

**PHD THESIS**

**SEPTEMBER 2017**

**GESTUI: A MODEL-DRIVEN  
METHOD TO INCLUDE  
GESTURE-BASED INTERACTION  
IN USER INTERFACES**

**BY OTTO PARRA GONZALEZ**

*Supervisors:*

Oscar Pastor López

Sergio España Cubillo

Ignacio Panach Navarrete



**UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA**



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA

## **PhD Thesis**

# **gestUI: a model-driven method to include gesture-based interaction in user interfaces**

**Otto Parra González**

**Supervisors:**

**Óscar Pastor López**

**Sergio España Cubillo**

**Ignacio Panach Navarrete**

September 2017



A thesis submitted by Otto Parra González in partial fulfilment of the requirements for the degree of Doctor of *Philosophy in Computer Science* by *Universitat Politècnica de València, Spain*.



# gestUI: a model-driven method to include gesture-based interaction in user interfaces

## This report was prepared by:

Otto Parra González

[otpargon@posgrado.upv.es](mailto:otpargon@posgrado.upv.es), [otto.parra@ucuenca.edu.ec](mailto:otto.parra@ucuenca.edu.ec)

## Supervisors

Óscar Pastor López, Universitat Politècnica de València

Sergio España Cubillo, University of Utrecht

Ignacio Panach Navarrete, Universitat de València

## External reviewers of the thesis:

Antoni Granollers Saltiveri, Universidad de Lleida

Lourdes Moreno López, Universidad Carlos III

Victor Manuel Ruiz Penichet, Universidad de Castilla-La Mancha

## Members of the Thesis Committee:

Jean Vanderdonckt, Université Catholique de Louvain

Antoni Granollers Saltiveri, Universidad de Lleida

Jose Antonio Macías Iglesias, Universidad Autónoma de Madrid

## Centro de Investigación en Métodos de Producción de Software

Universitat Politècnica de València

Camí de Vera s/n, Edificio 1F

46022, Valencia, España

Tel. (+34) 963 877 007 ext. 83533

Fax: (+34) 963 877 359

Web: <http://www.pros.upv.es>



Centro de Investigación en Métodos  
de Producción de Software

---

**Release date:** september-2017

**Comments:** A thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Computer Science by Universitat Politècnica de València.

**Rights:** © Otto Parra González, 2017.



## **Acknowledgements**

First of all, I would like to thank the supervisors Dr. Oscar Pastor López, Dr. Sergio España Cubillo and Dr. Ignacio Panach Navarrete for their support, rigor and guidance. I am very grateful for the trust placed in me. It has been a pleasure to have the opportunity to learn from them.

I want to thank Oscar for the opportunity he gave me to be part of PROS and because he was always ready to assist me when I needed it, with his advice and guidance to do my work.

Sergio, for his valuable work since the beginning of the thesis, because he was always ready to guide me along the long path of development of the thesis.

Ignacio, thanks for all the work done in the final part of the thesis corresponding to the evaluation of the process, for the review of the thesis and for the constant support that has given me since its incorporation to the "thesis supervisors team". In addition, for his help in translating the thesis summary to the Valencian.

I want to thank PROS staff, especially Ana Ciudad, Paco Valverde and Verónica Buriel because they were always willing to help me every time I went to them, they always knew how to help me. To the friends I made in this time of study in the UPV: José Reyes, Carlos Iñiguez, Julio Sandobalin, Sonia Cárdenas, Mauricio Loachamin, Nelly Condori-Fernandez, and Alejandro Catalá. To them, thank you for the support given me in every moment.

This work has been supported by Universidad de Cuenca and SENESCYT of Ecuador, and received financial support from the Generalitat Valenciana under Project IDEO (PROMETEOII/2014/039) and the Spanish Ministry of Science and Innovation through the DataMe Project (TIN2016-80811-P).

Finally, thanks to my wife Maria Fernanda and our children Maria Paula, Luis Felipe and Maria Emilia for their support and for being always close to me. For having been our faithful companions in this journey and adventure that has meant for them our permanence in Valencia.





## **Abstract**

The research reported and discussed in this thesis represents a novel approach to define custom gestures and to include gesture-based interaction in user interfaces of the software systems with the aim of help to solve the problems found in the related literature about the development of gesture-based user interfaces.

The research is conducted according to Design Science methodology that is based on the design and investigation of artefacts in a context. In this thesis, the new artefact is the model-driven method to include gesture-based interaction in user interfaces. This methodology considers two cycles: the main cycle is an engineering cycle where we design a model-driven method to include interaction based on gestures. The second cycle is the research cycle, we define two research cycles: the first research cycle corresponds to the validation of the proposed method with an empirical evaluation and the second cycle corresponds to the technical action research to validate the method in an industrial context.

Additionally, Design Science provides us the clues on how to conduct the research, be rigorous, and put in practice scientific rules. Besides Design Science has been a key issue for organising our research, we acknowledge the application of this framework since it has helps us to report clearly our findings.

The thesis presents a theoretical framework introducing concepts related with the research performed, followed by a state of the art where we know about the related work in three areas: Human-computer Interaction, Model-driven paradigm in Human-Computer Interaction and Empirical Software Engineering.

The design and implementation of gestUI is presented following the Model-driven Paradigm and the Model-View-Controller design pattern. Then, we performed two evaluations of gestUI: (i) an empirical evaluation based on ISO 25062-2006 to evaluate usability considering effectiveness, efficiency and satisfaction. Satisfaction is measured with perceived ease of use, perceived usefulness and intention of use, and (ii)

a technical action research to evaluate user experience and usability. We use Model Evaluation Method, User Experience Questionnaire and Microsoft Reaction cards as guides to perform the aforementioned evaluations.

The contributions of our thesis, limitations of the tool support and the approach are discussed and further work are presented.

## **Resumen**

La investigación reportada y discutida en esta tesis representa un método nuevo para definir gestos personalizados y para incluir interacción basada en gestos en interfaces de usuario de sistemas software con el objetivo de ayudar a resolver los problemas encontrados en la literatura relacionada respecto al desarrollo de interfaces basadas en gestos de usuarios.

Este trabajo de investigación ha sido realizado de acuerdo a la metodología Ciencia del Diseño, que está basada en el diseño e investigación de artefactos en un contexto. En esta tesis, el nuevo artefacto es el método dirigido por modelos para incluir interacción basada en gestos en interfaces de usuario. Esta metodología considera dos ciclos: el ciclo principal, denominado ciclo de ingeniería, donde se ha diseñado un método dirigido por modelos para incluir interacción basada en gestos. El segundo ciclo es el ciclo de investigación, donde se definen dos ciclos de este tipo. El primero corresponde a la validación del método propuesto con una evaluación empírica y el segundo ciclo corresponde a un Technical Action Research para validar el método en un contexto industrial.

Adicionalmente, Ciencia del Diseño provee las claves sobre como conducir la investigación, sobre cómo ser riguroso y poner en práctica reglas científicas. Además, Ciencia del Diseño ha sido un recurso clave para organizar la investigación realizada en esta tesis. Nosotros reconocemos la aplicación de este marco de trabajo puesto que nos ayuda a reportar claramente nuestros hallazgos.

Esta tesis presenta un marco teórico introduciendo conceptos relacionados con la investigación realizada, seguido por un estado del arte donde conocemos acerca del trabajo relacionado en tres áreas: Interacción Humano-Ordenador, paradigma dirigido por modelos en Interacción Humano-Ordenador e Ingeniería de Software Empírica.

El diseño e implementación de gestUI es presentado siguiendo el paradigma dirigido por modelos y el patrón de diseño Modelo-Vista-Controlador. Luego, nosotros hemos realizado dos evaluaciones de gestUI: (i) una evaluación empírica basada en ISO 25062-2006 para evaluar la usabilidad considerando efectividad, eficiencia y satisfacción. Satisfacción es medida por medio de la facilidad de uso percibida, utilidad percibida e intención de uso; y, (ii) un Technical Action Research para evaluar la experiencia del usuario y la usabilidad. Nosotros hemos usado Model Evaluation Method, User Experience Questionnaire y Microsoft Reaction Cards como guías para realizar las evaluaciones antes mencionadas.

Las contribuciones de nuestra tesis, limitaciones del método y de la herramienta de soporte, así como el trabajo futuro son discutidas y presentadas.

## Resum

La investigació reportada i discutida en aquesta tesi representa un mètode per definir gests personalitzats i per incloure interacció basada en gests en interfícies d'usuari de sistemes de programari. L'objectiu és ajudar a resoldre els problemes trobats en la literatura relacionada al desenvolupament d'interfícies basades en gests d'usuaris.

Aquest treball d'investigació ha sigut realitzat d'acord a la metodologia Ciència del Disseny, que està basada en el disseny i investigació d'artefactes en un context. En aquesta tesi, el nou artefacte és el mètode dirigit per models per incloure interacció basada en gests en interfícies d'usuari. Aquesta metodologia es considerada en dos cicles: el cicle principal, denominat cicle d'enginyeria, on es dissenya un mètode dirigit per models per incloure interacció basada en gestos. El segon cicle és el cicle de la investigació, on es defineixen dos cicles d'aquest tipus. El primer es correspon a la validació del mètode proposat amb una avaluació empírica i el segon cicle es correspon a un Technical Action Research per validar el mètode en un context industrial.

Adicionalment, Ciència del Disseny proveeix les claus sobre com conduir la investigació, sobre com ser rigorós i ficar en pràctica regles científiques. A més a més, Ciència del Disseny ha sigut un recurs clau per organitzar la investigació realitzada en aquesta tesi. Nosaltres reconeixem l'aplicació d'aquest marc de treball donat que ens ajuda a reportar clarament les nostres troballes.

Aquesta tesi presenta un marc teòric introduint conceptes relacionats amb la investigació realitzada, seguit per un estat del art on coneixem a prop el treball realitzat en tres àrees: Interacció Humà-Ordinador, paradigma dirigit per models en la Interacció Humà-Ordinador i Enginyeria del Programari Empírica.

El disseny i implementació de gestUI es presenta mitjançant el paradigma dirigit per models i el patró de disseny Model-Vista-Controlador. Després, nosaltres hem realitzat dos avaluacions de gestUI: (i) una avaluació empírica basada en ISO 25062-2006 per avaluar la

usabilitat considerant efectivitat, eficiència i satisfacció. Satisfacció es mesura mitjançant la facilitat d'ús percebuda, utilitat percebuda i intenció d'ús; (ii) un Technical Action Research per avaluar l'experiència del usuari i la usabilitat. Nosaltres hem usat Model Evaluation Method, User Experience Questionnaire i Microsoft Reaction Cards com guies per realitzar les avaluacions mencionades.

Les contribucions de la nostra tesi, limitacions del mètode i de la ferramenta de suport així com el treball futur són discutides i presentades.

# Contents

Chapter 1. Introduction .....	3
1.1    Motivation .....	3
1.1.1    Human-computer interaction .....	3
1.1.2    Software systems and development tools .....	5
1.2    Problem Statement .....	10
1.3    Research Questions .....	13
1.4    Thesis Objectives .....	14
1.5    Research Methodology .....	16
1.6    Expected Contributions .....	18
1.7    Thesis Context .....	20
1.8    Thesis Outline .....	20
Chapter 2. Theoretical Framework.....	25
2.1    Overview.....	25
2.2    A theoretical framework for Human-Computer Interaction. 26	
2.2.1    Gestures related definition.....	26
2.2.2    Classification of gestures .....	27
2.2.3    Gesture recognition algorithms.....	29
2.2.4    Gesture-based interaction.....	30
2.3    A theoretical framework of Model-Driven paradigm.....	31
2.3.1    Model related definition .....	31
2.3.2    MDA Conceptual framework.....	33
2.3.3    Model Transformations .....	34
2.3.4    Transformation Language.....	36
2.4    Summary.....	37
Chapter 3. State of Art.....	41



3.1	Motivation.....	41
3.2	Gesture representation.....	42
3.3	Gesture recognition tools .....	50
3.4	The role of gesture-based interfaces in Information Systems Engineering .....	53
3.5	Model-driven engineering in Human-Computer Interaction	56
3.6	Evaluation between model-driven paradigm and other methodologies .....	59
3.7	Technical action research to validate software systems .....	65
3.8	Range of Improvements.....	66
3.9	Summary .....	67
Chapter 4.	gestUI: A Model-Driven Method .....	71
4.1	Overview .....	71
4.2	Why a Model-Driven method? .....	73
4.3	Why a Model-View-Controller design pattern?.....	74
4.4	Determining needed resources .....	76
4.5	gestUI: our proposal.....	78
4.5.1	Features of gestUI .....	79
4.5.2	Metamodel of the gesture catalogue modelling language	80
4.5.3	Components of gestUI .....	91
4.5.4	Model transformations .....	95
4.6	Personalization of gesture definition.....	99
4.6.1	Introduction .....	99
4.6.2	Enhancing the metamodel.....	100
4.7	Overview of gestUI to include gesture-based interaction in a user interface .....	103

4.7.1	Introduction.....	103
4.7.2	Including gesture-based interaction in a user interface 104	
4.7.3	Redefining a gesture during the execution time .....	111
4.8	Summary.....	116
Chapter 5.	gestUI Tool Support .....	119
5.1	Introduction.....	119
5.2	Components of the tool support.....	120
5.2.1	Subsystem “Gesture Catalogue Definition Module” ...	121
5.2.2	Subsystem “Gesture-Action Correspondence Definition Module”	122
5.2.3	Subsystem “Model Transformation Module” .....	125
5.3	Development methodology of the tool support .....	126
5.4	Implementation of the tool support .....	126
5.4.1	Option 1: “Gesture catalogue definition” .....	127
5.4.2	Option 2: “Specific catalogue” .....	130
5.4.3	Option 3: “Gesture-action correspondence definition” 131	
5.4.4	Module to redefine gesture .....	134
5.5	Demonstration of the tool support.....	135
5.5.1	Applying the method and tool to testing a gesture catalogue	135
5.5.2	Applying the method and the tool to integrate gestUI into user interface development.....	137
5.6	Summary and Conclusions .....	140
Chapter 6.	Empirical Evaluation .....	145
6.1	Introduction.....	145

6.2	Experimental planning .....	146
6.2.1	Goal .....	146
6.2.2	Research Questions and Hypothesis Formulation .....	147
6.2.3	Factor and Treatments.....	149
6.2.4	Response variables and metrics.....	149
6.2.5	Experimental Subjects.....	153
6.2.6	Experiment design .....	154
6.2.7	Experimental objects .....	157
6.2.8	Instrumentation .....	158
6.2.9	Experiment procedure .....	159
6.2.10	Threats of validity.....	168
6.2.11	Data analysis .....	172
6.3	Results.....	174
6.3.1	RQ1: Effectiveness in the inclusion of gesture-based interaction.....	174
6.3.2	RQ2: Effectiveness in the definition of custom gestures	177
6.3.3	RQ3: Efficiency in the inclusion of gesture-based interaction.....	180
6.3.4	RQ4: Efficiency in the definition of custom gestures..	182
6.3.5	RQ5: Perceived Ease of Use .....	185
6.3.6	RQ6: Perceived Usefulness .....	187
6.3.7	RQ7: Intention to Use .....	190
6.3.8	Effect-size calculation .....	192
6.4	Discussion.....	195
6.4.1	Effectiveness .....	195
6.4.2	Efficiency .....	197

6.4.3	Satisfaction .....	199
6.5	Conclusions.....	200
Chapter 7.	Technical Action Research .....	205
7.1	Introduction.....	205
7.2	Background: Capability Design Tool .....	208
7.3	Validation using Technical Action Research .....	210
7.3.1	Goal of the TAR.....	211
7.3.2	Experimental subjects .....	211
7.3.3	Research questions.....	212
7.3.4	Factor and Treatment.....	212
7.3.5	Response variables .....	214
7.3.6	Instruments for the TAR .....	214
7.3.7	Experimental Object.....	214
7.4	Action Research Procedure .....	215
7.5	Analysis and Interpretation of results .....	218
7.6	Threats to validity.....	222
7.7	Conclusions.....	223
Chapter 8.	Conclusions, Contributions and Future Work .....	227
8.1	Summary of the thesis.....	227
8.2	Contribution of this thesis .....	229
8.3	Future work .....	231
8.4	Conclusion .....	232
8.5	Publications .....	233
Appendix A.	A code-centric method for develop user interfaces with gesture-based interaction .....	239
A.1	Introduction .....	239

A.2 The code-centric method .....	239
References .....	243

## List of Figures

<i>Figure 1. Design science research iterates over two problem-solving activities (taken from [30])</i>	17
<i>Figure 2. Overview of the research methodology</i>	19
<i>Figure 3. Types of semaphoric gestures</i>	28
<i>Figure 4. MDA Layers</i>	34
<i>Figure 5. MDA Transformations</i>	35
<i>Figure 6 Model-to-Model Transformation</i>	36
<i>Figure 7. Model-to-text transformation</i>	37
<i>Figure 8. quill's main interface</i>	51
<i>Figure 9. \$N's main interface</i>	51
<i>Figure 10. iGesture main interface</i>	52
<i>Figure 11. Software System with traditional interaction</i>	75
<i>Figure 12. Modifying the controller to support gesture-based interaction</i>	76
<i>Figure 13. Metamodel of the gesture catalogue modelling language</i>	81
<i>Figure 14. States of a posture</i>	87
<i>Figure 15. Precedence relation between postures</i>	88
<i>Figure 16. Interval of time between postures</i>	89
<i>Figure 17. A general excerpt of any method for develop user interfaces</i>	91
<i>Figure 18. gestUI method overview (Taken from [31])</i>	92
<i>Figure 19. Platform-independent gesture definition</i>	93
<i>Figure 20. An excerpt of Figure 18 showing the M2M transformation</i>	96
<i>Figure 21. An excerpt for the M2M transformation</i>	97
<i>Figure 22. An excerpt of Figure 18 showing the M2T transformation to obtain the gesture-based user interface</i>	98
<i>Figure 23. An excerpt of Figure 18 showing the M2T transformation to obtain the test gesture</i>	98
<i>Figure 24. An excerpt for the M2T transformation</i>	99
<i>Figure 25. An excerpt of gestUI showing the redefinition of a gesture</i>	100
<i>Figure 26. Enhanced version of the metamodel</i>	101
<i>Figure 27. Users defining their own gestures catalogue to apply it in the same user interface</i>	102
<i>Figure 28. An excerpt of the map representation of gestUI</i>	103
<i>Figure 29. MAP representation of gestUI</i>	105
<i>Figure 30. User defining a gesture</i>	108
<i>Figure 31. Platform-independent gesture catalogue</i>	109
<i>Figure 32. A specific-platform gesture catalogue</i>	110

<i>Figure 33. An excerpt of the source code of a user interface containing widget definition and keywords</i>	110
<i>Figure 34. Map representation of the software system with the redefinition feature included</i>	114
<i>Figure 35. gestUI tool support</i>	121
<i>Figure 36. An excerpt of Figure 35 showing the subsystem "Gesture Catalogue Definition Module"</i>	122
<i>Figure 37. An excerpt of Figure 35 showing the subsystem "Gesture-action Correspondence Definition Module"</i>	123
<i>Figure 38. Excerpt of Figure 35 showing the subsystem "Model Transformations Module"</i>	126
<i>Figure 39. Main interface of the tool support</i>	127
<i>Figure 40. Screenshot of the interface of gestUI to sketch gestures</i>	128
<i>Figure 41. User sketching a gesture and storing it in a repository</i>	129
<i>Figure 42. Screenshot of the user interface to obtain the platform-independent gesture catalogue</i>	129
<i>Figure 43. An excerpt of a rule of the M2M transformation</i>	130
<i>Figure 44. M2M transformation parameters</i>	130
<i>Figure 45. Interface for defining gesture-action correspondence and to generate source code</i>	131
<i>Figure 46. SWT components to define actions</i>	132
<i>Figure 47. JFace and SWT components used to define an action in a user interface</i>	133
<i>Figure 48. Interface to execute a model-to-text transformation</i>	134
<i>Figure 49. An example of the module to redefine custom gestures</i>	135
<i>Figure 50. Gesture catalogue defined by gestUI</i>	136
<i>Figure 51. Gesture description files: \$N (left), quill (centre), iGesture (right)</i>	136
<i>Figure 52. Importing the gesture catalogue to the quill framework</i>	137
<i>Figure 53. Examples of multi-stroke gestures: \$N (left) and quill (centre) and iGesture (right)</i>	137
<i>Figure 54. UML class diagram of the demonstration case</i>	138
<i>Figure 55. Screen mockups (gestures are shown in red, next to action buttons)</i>	138
<i>Figure 56. Using gestures to execute actions on the interfaces</i>	139
<i>Figure 57 Software system supporting traditional interaction</i>	157
<i>Figure 58 Software system supporting gesture-based interaction</i>	161
<i>Figure 59 Gesture-action correspondence definition using tool support</i>	167
<i>Figure 60 Box-and-whisker plot of PTCCI</i>	175

<i>Figure 61</i>	<i>Box-plot-whisker of PTCCG</i>	178
<i>Figure 62</i>	<i>Box-plot for TFTI</i>	181
<i>Figure 63</i>	<i>Box-plot of TFTG</i>	183
<i>Figure 64</i>	<i>Box-plot for PEOU</i>	186
<i>Figure 65</i>	<i>Box-plot of PU</i>	188
<i>Figure 66</i>	<i>Box-plot of ITU</i>	191
<i>Figure 67</i>	<i>An excerpt of User Experience Questionnaire (taken of <a href="http://www.ueq-online.org">www.ueq-online.org</a>)</i>	207
<i>Figure 68</i>	<i>An excerpt of the 118 positive and negative phrases of Microsoft Reaction Cards</i>	208
<i>Figure 69</i>	<i>CDT with traditional interaction using keyboard and mouse</i>	210
<i>Figure 70</i>	<i>CDT with gesture-based interaction</i>	210
<i>Figure 71</i>	<i>Excerpt of a model defined in Everis</i>	215
<i>Figure 72</i>	<i>UEQ results: custom gesture definition interaction</i>	219
<i>Figure 73</i>	<i>UEQ results: inclusion of gesture-based interaction</i>	219
<i>Figure 74</i>	<i>Reaction cards positive results</i>	221
<i>Figure 75</i>	<i>Reaction cards negative results</i>	222
<i>Figure 76</i>	<i>A code-centric method for develop user interfaces with gesture-based interaction</i>	240





## List of Tables

<i>Table 1. Examples of application of gesture-based interaction outside the office</i>	6
<i>Table 2. Some software development kit including toolbox to design user interfaces</i>	8
<i>Table 3. Objectives for the research questions</i>	14
<i>Table 4. Related areas in the thesis</i>	25
<i>Table 5. A summary related with gesture representation</i>	47
<i>Table 6. Summary of works related with role of gesture-based interfaces in Information Systems Engineering</i>	55
<i>Table 7. Summary of works related with Model-driven engineering in Human-Computer Interaction</i>	58
<i>Table 8. Summary of works related with Evaluation between model-driven paradigm and other methodologies</i>	62
<i>Table 9. Detected problems vs. Benefits of model-driven paradigm to solve them</i>	80
<i>Table 10. Business rules for the "Catalogue" class</i>	81
<i>Table 11. Business rules of the "Gesture" class</i>	82
<i>Table 12. Business rule of the "Action" class</i>	83
<i>Table 13. Business rule of the "Stroke" class</i>	83
<i>Table 14. Business rules for the "Posture" class</i>	84
<i>Table 15. Business rule for the "Precedence" class</i>	84
<i>Table 16. Business rule for the "Point" class</i>	84
<i>Table 17. Data structure of a gesture</i>	86
<i>Table 18. Constraints and business rules of gesture definition</i>	90
<i>Table 19. Business rules for the "User" class</i>	101
<i>Table 20. Business rules for the "UserInterface" class</i>	102
<i>Table 21. Strategies of gestUI</i>	106
<i>Table 22. Strategies of the software system with gesture-based interaction</i>	115
<i>Table 23. Platform-independent gesture catalogue definition</i>	139
<i>Table 24. Factor and treatments of the experiment</i>	149
<i>Table 25. Response variables to evaluate effectiveness and efficiency of gestUI</i>	151
<i>Table 26. Responses variables to measure satisfaction of use gestUI</i>	152
<i>Table 27. Summary of RQ's, hypotheses, response variables and metrics</i>	153
<i>Table 28. Summary of demographic questionnaire</i>	155
<i>Table 29. Crossover design</i>	156

<i>Table 30 Operators and average time on KLM</i>	<i>156</i>
<i>Table 31 Estimating time for the experiment</i>	<i>157</i>
<i>Table 32 Instruments defined for the experiment</i>	<i>159</i>
<i>Table 33 Gesture catalogue defined in the experiment</i>	<i>160</i>
<i>Table 34 An excerpt of the Task Description Document containing the sequence of steps for custom gesture definition using the code-centric method</i>	<i>163</i>
<i>Table 35 An excerpt of the Task Description Document containing the sequence of steps for gesture-based interaction inclusion using the code-centric method</i>	<i>165</i>
<i>Table 36 An excerpt of the Task Description Document for custom gesture definition using gestUI</i>	<i>166</i>
<i>Table 37 Gesture-action correspondence step-by-step definition</i>	<i>167</i>
<i>Table 38 Non-parametric Levene's test for the variables in the experiment</i>	<i>174</i>
<i>Table 39 Descriptive statistics for PTCCI</i>	<i>175</i>
<i>Table 40 Spearman's Rho correlation coefficient of PTCCI</i>	<i>176</i>
<i>Table 41 Wilcoxon Signed-rank test for PTCCI</i>	<i>176</i>
<i>Table 42 Wilcoxon Signed-rank test statistics for PTCCI</i>	<i>177</i>
<i>Table 43 Descriptive statistics for PTCCG</i>	<i>177</i>
<i>Table 44 Spearman's Rho correlation coefficient of PTCCG</i>	<i>179</i>
<i>Table 45 Wilcoxon Signed-rank test for PTCCG</i>	<i>179</i>
<i>Table 46 Wilcoxon Signed-rank test statistics for PTCCG</i>	<i>179</i>
<i>Table 47 Descriptive statistics for TFTI</i>	<i>180</i>
<i>Table 48 Spearman's Rho correlation coefficient of TFTI</i>	<i>181</i>
<i>Table 49 Wilcoxon Signed-rank test for TFTI</i>	<i>182</i>
<i>Table 50 Wilcoxon Signed-rank test statistics for TFTI</i>	<i>182</i>
<i>Table 51 Descriptive statistics for TFTG</i>	<i>183</i>
<i>Table 52 Spearman's Rho correlation coefficient of TFTG</i>	<i>184</i>
<i>Table 53 Wilcoxon Signed-rank test for TFTG</i>	<i>184</i>
<i>Table 54 Wilcoxon Signed-rank test statistics for TFTG</i>	<i>185</i>
<i>Table 55 Descriptive statistics for PEOU</i>	<i>185</i>
<i>Table 56 Spearman's Rho correlation coefficient of PEOU</i>	<i>185</i>
<i>Table 57 Wilcoxon Signed-rank test for PEOU</i>	<i>187</i>
<i>Table 58 Wilcoxon Signed-rank test statistics for PEOU</i>	<i>187</i>
<i>Table 59 Descriptive statistics for PU</i>	<i>188</i>
<i>Table 60 Spearman's Rho correlation coefficient of PU</i>	<i>189</i>
<i>Table 61 Wilcoxon Signed-rank test for PU</i>	<i>189</i>
<i>Table 62 Wilcoxon Signed-rank test statistics for PU</i>	<i>190</i>

<i>Table 63 Descriptive statistics for ITU</i>	<u>190</u>
<i>Table 64 Spearman's Rho correlation coefficient of ITU</i>	<u>191</u>
<i>Table 65 Wilcoxon Signed-rank test for ITU</i>	<u>191</u>
<i>Table 66 Wilcoxon Signed-rank test statistics for ITU</i>	<u>192</u>
<i>Table 67 Summary of the results obtained in the experiment</i>	<u>193</u>
<i>Table 68 Effect size of the metrics</i>	<u>194</u>
<i>Table 69. Instruments defined for the validation</i>	<u>213</u>
<i>Table 70. Gesture catalogue defined by the subjects</i>	<u>214</u>
<i>Table 71. A summary of the experiment procedure</i>	<u>217</u>
<i>Table 72. Results obtained from the UEQ</i>	<u>218</u>
<i>Table 73. Reaction cards positive results</i>	<u>220</u>
<i>Table 74. Reaction cards negative results</i>	<u>220</u>



---

INTRODUCTION

1

The topics covered in this chapter are:

- 1.1 Motivation
- 1.2 Problem Statement
- 1.3 Research Questions
- 1.4 Thesis Objectives
- 1.5 Research Methodology
- 1.6 Expected Contributions
- 1.7 Thesis Context
- 1.8 Thesis Outline



# Chapter 1. Introduction

## 1.1 Motivation

Computers have evolved, in recent decades, since the advent of the personal computer towards current mobile devices. Two factors have contributed to the wide diffusion of computing devices [1]: (a) an appropriate human-computer interaction which resulting in the ease of use of services and software systems available for the devices, and (b) the availability of a wide variety of services, software systems and development tools. These two factors are analysed in this section.

### 1.1.1 Human-computer interaction

First of all, we analyse *human-computer interaction (HCI)*. Since the advent of the personal computer, HCI has changed, we first had a simple interface using a command line (CLI) through which the user entered orders that were based on operating system commands. The interaction between computer and the user was complicated because the number of commands and the complexity were increased in the next years.

Then came the development of graphical user interface (GUI) that uses keyboard and mouse, these two elements have been for many years the devices employed by the user to entering information to the computer [2]. The interaction between computer and the people was improving. New operating systems were developed (e. g. Microsoft Windows<sup>1</sup>, Mac OS<sup>2</sup>, Linux<sup>3</sup>) which helped to improve the human-computer interaction. WIMP (Window, Icon, Menu, Pointer) appeared in the scenario of the computers and the user interface design started to include elements that helped to the users to interact in a better way with the computers.

---

<sup>1</sup> <https://www.microsoft.com/en-gb/windows>

<sup>2</sup> <http://www.apple.com/uk/osx/>

<sup>3</sup> <http://www.ubuntu.com/>



With the development of the mobile devices, besides of keyboard and mouse, other elements appeared in the technology market, for example a pointer that was used for data entry in Palm<sup>4</sup> devices [3], then with the development of touch screens began to use the fingers of the user's hand, which led to the emergence of gesture as a natural interaction, whose primary goal is better communication between the user and the computer [4].

The next step was the development of hardware tools that the user employs to perform actions with gestures using more than a finger. Devices vary in their type and features, there are some types of computers (e.g. desktop computer, notebook, netbook) that include additional technologies to interact with the users. For instance, there are computers such as desktop and notebooks supporting touch-based interaction and the devices of reduced size (tablet, smartphone) have included by default the touch-based interaction. Devices such as Microsoft Kinect<sup>5</sup>, Nintendo Wii<sup>6</sup> were primarily aimed at allowing the user to play using the body as an "instrument" for the movement to be carried out in the games [4] [5]. This fact allowed the development of tools that capture gestures made by users and process them to perform actions on other activities in addition to the games. This leads to the concept of natural user interface (NUI) [3].

NUI refers to interfaces that allow the user to interact with a system based on the knowledge learnt from using other systems [6]. NUI promises to reduce barriers to compute even more than GUI, while simultaneously increases the power of the user and allows the computing access to more niches of use [3].

The gesture which is an element used for interaction in NUI has caught the attention of end users and developers. It is a movement made by a user, either with his/her fingers, hand or with whole body [7], with

---

<sup>4</sup> [https://en.wikipedia.org/wiki/Palm\\_\(PDA\)](https://en.wikipedia.org/wiki/Palm_(PDA))

<sup>5</sup> <https://developer.microsoft.com/es-es/windows/kinect>

<sup>6</sup> <https://www.nintendo.es/Wii/Wii-94559.html>

the main objective of establishing a communication with an electronic device to perform some action [8]. A gesture can be applied on a touch-sensitive surface, or carried in front of a device that captures the movements made by the user. The use of gesture has created a type of interaction called gesture-based interaction.

The methods of gesture-based interaction that have been developed in recent years are related mainly to two types of technologies: touch-based and vision-based. Regarding the former, its development is due to the great popularity of devices that support touch-based interaction (smartphones, tablets, etc.). Also, researchers have developed a series of investigations about this type of interaction. The second type of interaction that is based on vision is considered like a type of more natural interaction since the user “do not touch any element or surface” to interact with the device.

The development of computing devices makes possible that two main interaction styles are available in the field of user interfaces: (i) WIMP [9] supporting traditional interaction based on keyboard and mouse and using the graphical user interface (GUI) on desktop computers and notebooks, and (ii) post-WIMP [3] supporting natural user interaction with other interaction techniques, mainly gesture-based, gaze-based, and voice-based. Post-WIMP is associated with non-conventional interaction employing other interaction styles that are currently available resulting in NUI.

The interest of this thesis is to consider the touch-based gesture as an element of communication between the user and devices that supports gesture-based interaction.

### **1.1.2 Software systems and development tools**

The second aforementioned aspect is related to *software systems and development tools* available on the diverse computing platforms.

In the most recent years, the requirements to include new interaction techniques in user interfaces is increasing dramatically because new

devices come together with new types of interfaces (e.g. based on gaze, gesture, voice, haptic, brain-computer interfaces) [3]. Their aim is to increase the naturalness of interaction, although this is not exempt from risks. Due to the popularity of touch-based devices, gesture-based interaction is slowly gaining ground on mouse and keyboard in domains such as video games and mobile apps.

The increasing scope of the application areas suggests the importance of undertaking more work in gesture-based interaction research [10]; furthermore the gesture-based interaction has many applications in a variety of fields in the society, too. The prevalence of gesture-based commercial products has increased since five years ago, as the technology has improved and become commercially viable. For instance, gestures have been used in projects that attempt to create novel or improved interactions for appliance control and home entertainment systems [11]. Table 1 includes some examples about the trend in supporting tasks performed outside the office by means of gesture-based interaction.

**Table 1. Examples of application of gesture-based interaction outside the office**

Authors	Description
Yang et al. [12]	It describes the inclusion of gestures in the Building Information Model (BIM) technology in order to make more intuitive its manipulation. This research focuses on developing an interactive interface for site workers to retrieve information from BIM models by means of gestures.
Fujitsu Laboratories [13]	It announces that they have developed a wearable device in the form of a glove supporting gestures for maintenance and other on-site operations.
Song et al. [14] [15]	It presents a unified framework for body and hand tracking in order to apply in the aircraft field.
Kim et al. [16]	The authors describe a method for hand gestures recognition under varying illumination conditions. The application is oriented to places with different levels of illumination.
Cardoso et al. [17]	It describes the development of an application with a 3D sensor included, in order to implement the interaction based on swipe gestures to navigate through options, menu and operations of selection and deselection.
Weiss et al. [18]	It describes a proposal to implement a service robot capable of safely navigation in densely populated environments supporting hand gestures to execute actions.

People need to interact with multiple mobile devices (e.g. notebook, tablet, and mobile phone) at the same time using non-conventional interaction, typically gesture-based interaction. This, in turn, implies that the software engineers must be prepared for a major change in user interfaces development considering diversity of platforms and applications available in those devices. This situation involves that software engineers require (i) tools to specify and to implement custom gestures that people use in their daily tasks, and (ii) tools to design and to implement user interfaces supporting custom gesture-based interaction.

Nowadays there are many software development kits (SDK) that allow the development of software systems for device platforms available in the market. Many of these SDKs are specific to some manufacturers of the device platform which allows developing software system specific for that platform (proprietary software). There are others that allow software system development for two or three platforms; but they require to acquire licenses and the cost is associated with the number of platforms/number of developers. Some SDK are free but they have restricted access in relation to available characteristics of the tools and target platforms. Finally, the free version of some manufacturers is a trial version with a limited number of days.

The variety of SDK's and platforms had forced the developers to focus on a specific device platform with the aim of achieving a suitable domain of programming languages and integrated development environment (IDE) for developing software systems. Hence, the wide range of SDK's available in the technology market implies a wide range of tools (programming languages, compilers, IDE's, etc.) for the construction of software systems.

Table 2 contains details about SDK's for the most significant platforms available in the technology market.

**Table 2. Some software development kit including toolbox to design user interfaces**

SDK name	Device Platform	Software Platform	Included basic services	Custom gestures definition?	License/platform version	Free version	Programming Language
Microsoft Visual Studio <sup>a</sup>	Desktop, Tablet, Smartphone	Windows	Editor, Debugger, Interface Builder, Source Control	No	Yes	Yes	C++, C#, Visual Basic
Genexus <sup>b</sup>	Desktop, Tablet, Smartphone	Windows, Mac, Android	Editor, Simulator, Debugger, Interface Builder, Source Control	No	Yes	Trial version	HTML5, Java for Android, Objective-C for iOS
Xamarin <sup>c</sup>	Desktop, Tablet, Smartphone	iOS, Mac, Android	Editor, Simulator, Debugger, Interface Builder, Source Control	No	Yes	Yes	C#
Cocoa + XCode and Cocoa Touch <sup>d</sup>	Apple	iOS, Mac	Editor, Simulator, Debugger, Interface Builder, Source Control	No	No	Yes	ANSI C, C++, Objective-C
Eclipse Framework <sup>e</sup>	Desktop, Tablet, Smartphone	Windows, Mac, Linux	Editor, Simulator, Debugger, Interface Builder, Source Control	No	No	Yes	Java
PeopleSoft Fluid (ORACLE) <sup>f</sup>	Desktop, Tablet, Smartphone	Any platform with Web support	Editor, Simulator, Debugger, Interface Builder	No	Yes	Trial version	HTML5, JavaScript, CSS3
Wavemaker <sup>g</sup>	Desktop, Smartphone	Any platform with Web support	Editor, Simulator, Debugger, Interface Builder	No	Yes	Trial version	HTML, JavaScript, CSS3

<sup>a</sup> <https://www.microsoft.com/visualstudio>

<sup>b</sup> <http://www.genexus.com>

<sup>c</sup> <https://www.xamarin.com>

<sup>d</sup> <https://developer.apple.com/xcode/interface-builder/>

<sup>e</sup> <http://www.eclipse.org>

<sup>f</sup> <http://www.oracle.com>

<sup>g</sup> <http://www.wavemaker.com>

With the increasing prevalence of computers and other related technologies in many facets of today's society, it becomes increasingly important to create software systems capable of interacting with humans properly [19]. One important aspect to consider is that, depending on the device, there is a variety of operating platforms that imply separate standards, programming languages, development tools, and in some cases, even distribution markets (i.e. Web portals) through which users can purchase and download applications [20].

However, some complications are present when software engineers decide to implement user interfaces [9] with gesture-based interaction by means of traditional software tools (e.g. Microsoft Visual Studio Enterprise [21], Eclipse Window Builder [22]), for instance, software engineers require additional knowledge and experience in: (i) the specification and implementation of custom gestures, (ii) the design and implementation of gesture-based user interfaces, (iii) the use of software tools depending of the platform selected to implement gesture-based user interfaces.

Therefore, building software systems with gesture-based interaction is complicated yet due to the diversity of devices, software platforms and development tools to design and to implement user interfaces that supports custom gestures [9]. Typically, the software engineer requires skills to implement custom gestures and to write the methods required to support them in a user interface.

This thesis proposes a solution that helps to resolve this situation: the inclusion of gesture-based interaction independently of the platform in user interfaces of software systems. For this aim, we propose a method to define custom gestures and to include gesture-based interaction, which is independent of the technology of the devices.

The remainder of this chapter is organized as follows: Section 1.2 details the problem statement. Section 1.3 includes the research questions proposed for this thesis. Section 1.4 introduces the objectives of the thesis. Section 1.5 introduces the research

methodology that has been followed in the thesis. Section 1.6 describes the expected contributions of the thesis. Section 1.7 explains the context in which the thesis has been performed. Finally, Section 1.8 gives an overview of the structure of this thesis.

## 1.2 Problem Statement

The development of user interfaces, ranging from early requirements to software obsolescence, has become a time-consuming and costly process in the software development life cycle (SDLC) [23]. In this process, it would be more effective to include the interaction specifications so that the software fulfils the requirements of users and also provides an interaction according to the type of task to be performed with the software.

During the SDLC, specifically in the code stage, the software engineers have some software tools and software development paradigms to implement user interfaces, typically, some of them use event-driven programming to build the user interface (e.g. Microsoft Visual Studio<sup>7</sup>, Eclipse Window Builder<sup>8</sup>). In this context, the process to obtain a user interface is based on the selection, insertion and customization of each component of the user interface from a toolbox included in the software tool.

This situation forces developers to build software systems for specific platforms, furthermore, developing an independent software product for each platform requires conducting significant parts of the software life cycle several times for each released software system, which may become redundant and expensive [20]. This situation complicates the work of software developers, as they are required to have a wide domain of development tools, programming languages, and the processes that comprise the software systems development life cycle to be able to develop and maintain software systems to any platform.

---

<sup>7</sup> <https://www.visualstudio.com/>

<sup>8</sup> <https://eclipse.org/windowbuilder/>

Now, if the user requires using custom gestures to do tasks then the process is more complicated. In this case, in order to obtain software systems supporting gesture-based interaction, the software engineer must have experience in two fields: (i) in the custom gesture definition and (ii) in the inclusion of source code to manage custom gestures in the user interface in different platforms. Complexity could be increased if the users require custom gestures in user interfaces to different devices and different platforms. Therefore, implementation process of gesture-based user interfaces is complicated and costly.

Several concerns may delay the wide adoption of gesture-based interaction in complex software systems. Gesture-based interfaces have been reported to be more difficult to implement and test than traditional mouse and pointer interfaces [24]. Gesture-based interaction is supported at the source code level (typically third-generation languages) [25]. This involves a great coding and maintenance effort when multiple platforms are targeted, it has a negative impact on reusability and portability, and it complicates the definition of new gestures.

Some of the aforementioned challenges (e.g. complexity, cost, expertise required, reusability, portability, and multiple platforms) can be tackled by following a model-driven development (MDD) approach [26] provided that gestures and gesture-based interaction can be modelled and that it is possible to automatically generate the software components that support them. If a model-driven development is intended, it is essential that the models include complete requirements to create the software product using model-transformation and code-generation tools.

MDD has been fairly popular in the academic community [27] in recent years, and a number of different proposals have been introduced to develop software systems. MDD is a software development paradigm which is based on models and model transformations in order to obtain a final product by means of automatic code generation considering some transformation rules.



In a field where technology changes rapidly, a model-driven methodology is a valid option for some reasons:

- The domain of the knowledge is represented in models, which are independent of technology [28],
- The solution for the development of a software system is not affected by the evolution of the hardware platform.
- When a new technology is considered as a target platform to develop software, it is not necessary to describe the whole system again but only to generate a new platform-specific model (PSM) including the changes in the target platform.
- Tasks related with the development life cycle (maintenance, new versions, documenting process) are less complicated to make them [29].

This thesis introduces an MDD method for development of gesture-based user interfaces and a tools that supports it. The method aims to allow software engineers focusing on the key aspects of gesture-based user interfaces; namely, defining custom gestures and specifying gesture-based interaction. Coding and portability efforts are alleviated by means of model-to-text (M2T) transformations

In summary, the problem statement in this thesis is:

*In the context of devices supporting gesture-based interaction, there is a fast technological development of devices and there are SDK's, specific for each platform, to build software systems. This situation makes difficult a better development of the software systems with gesture-based interaction in aspects related with availability, portability and their distribution.*

*The growing need for development methodologies for software systems that are adaptable to new demands of service users, permits that Model-Driven Development is presented as an alternative to meet the need to develop software systems with gesture-based interaction, in minimal time and for any hardware and software platforms.*

Our work aims to define a method for improving the process of defining custom gestures and the inclusion of gesture-based interaction in user interfaces of software systems in an MDD environment.

### 1.3 Research Questions

In this thesis, we aim at gathering new knowledge and producing useful artefacts; thus, we opt for a design science approach [30].

The **main goal** of this thesis is **to provide a methodological approach to define custom gestures and to include gesture-based interaction in software systems user interfaces.**

In order to accomplish this goal, we must answer research questions defined in this section. We classify these research questions as either knowledge problems (KP) or design problems (DP), based on the definitions by Wieringa [31]:

**RQ1 (KP):** *What elements should be considered for the definition of a method to include gesture-based interaction in user interfaces?*

**RQ2 (KP):** *What model-driven methods exist to include gesture-based interaction in user interfaces with human--computer interaction based on gestures?*

**RQ3 (DP):** *Is it possible to define a model-driven method for the inclusion of gesture-based interaction in software systems user interfaces?*

**RQ4 (KP):** *What advantages and disadvantages has the model-driven method for the inclusion of gesture-based interaction in software systems user interfaces?*

Section 1.4 describes how to tackle the answer for each research question through the thesis.

## 1.4 Thesis Objectives

The main goal of the thesis is to define a method to help improving the process of definition of custom gestures and the inclusion of gesture-based interaction in user interfaces by means of a MDD environment. Since MDD has proved to be effective in managing the complexity of the software systems development process [32], we will apply it to the particular domain of interest for this thesis.

The general objective and the specific objectives established for this thesis are shown in Table 3.

**Table 3. Objectives for the research questions**

Research Questions	Objectives
Overall RQs	Define a method to help improving the process of define custom gestures and the inclusion of gesture-based interaction in user interfaces of software systems in an MDD environment.
RQ1	Determine the elements that should be considered to define a method for the inclusion of gesture-based interaction in user interfaces.
RQ2	Determine the existing methods to include gesture-based interaction in user interfaces with included gesture-based interaction.
RQ3	Define a model-driven method for the inclusion of gesture-based interaction in software systems user interfaces. In this case, we consider three sub goals: <ul style="list-style-type: none"><li>i. Determine which gesture characteristics are representative to be used as descriptors of human-computer interaction.</li><li>ii. Define a tool to represent a gesture in the specification of the interaction in user interface.</li><li>iii. Establish techniques and tools to facilitate the use of the proposed method.</li></ul>
RQ4	Evaluate advantages and disadvantages of the method for the inclusion of gesture-based interaction in user interfaces.

First of all, with regard to research question 1 (**What elements should be considered for the definition of a method to include gesture-based interaction in user interfaces?**), one of the goals of this thesis is determine which elements should be considered in the definition of the aforementioned method. With the aim of determining these elements, we perform a review of the related literature available on Internet. Also, we do a bibliographic review of related topics in order to know how to structure this type of method.

Regarding research question 2 (**What model-driven methods exist to include gesture-based interaction in user interfaces with human-computer interaction based on gestures?**), we need to do a review of the related literature in order to know which methods have been developed to include gesture-based interaction in user interfaces. We are interested in methods that include concepts related with Human-computer Interaction and Model-driven Development because there is a variety of platforms of hardware and software involved in this context. Since MDD claims to be independent of any technology platform, it has been considered for the definition of a method that helps in developing gesture-based software systems

With regard to research question 3 (***Is it possible to define a model-driven method for the inclusion of gesture-based interaction in software systems user interfaces?***), we consider the three sub-goals included in Table 1.3 to answer this question. The first sub-goal (**Determine which gesture characteristics are representative to be used as descriptors of human-computer interaction**) is related with the *study of the gesture representation* because nowadays the gesture is considered an important element of interaction between human and computer. To achieve the mentioned goal, we propose performing an analysis of the related literature to determine the features that we need to be captured to use them as descriptors of the gesture. Additionally, we use gestures in a software development environment with the aim of capturing the data related with each gesture by means of a debug process. Using this information, we establish the gesture representation in a conceptual model and include it in the design process of software systems with gesture-based interaction.

Regarding second sub-goal (**Define a tool to represent a gesture in the specification of the interaction in user interface**), another goal of this thesis is to define a tool that permits to represent a gesture based on the conceptual model specified with the previously obtained information. We consider MDD in this thesis as a solution to minimize the problems of diversity of platforms that imply separated standards,

programming languages and development tools to develop gesture-based interaction.

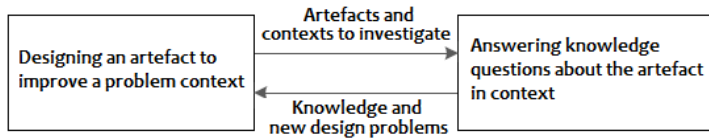
With regard to third sub-goal (**Establish techniques and tools to facilitate the use of the proposed method**), it is necessary to define techniques and a tool support in order to implement the proposed method. In this case, we use Java programming language and Eclipse Modelling Framework to define components of the proposed method.

Regarding research question 4 (**What advantages and disadvantages has the model-driven method for the inclusion of gesture-based interaction in software system user interfaces?**), we need to verify if the proposed framework facilitates the inclusion of gesture-based interaction in user interfaces of software systems. Empirical evaluation helps us to validate the proposed method in the gesture-based interaction field. Some mechanisms must be provided in order to obtain valuable feedback regarding the user experience in the developed process.

## 1.5 Research Methodology

The type of research of this thesis corresponds to the design science framework since it aims to design a new artefact, by means of acting and deciding on the basis of a systematic body of evidence [33]. Design science is a methodology based on the design and investigation of artefacts in a context. The artefacts we study are designed to interact with a problem context in order to improve something in that context [31]. In this thesis, the new artefact is the model-driven method to include gesture-based interaction in user interfaces.

The two parts of design science, design and investigation, correspond to two kinds of research problems in design science, namely, design problems and knowledge questions (Figure 1) [30].



**Figure 1. Design science research iterates over two problem-solving activities (taken from [30])**

This methodology proposes: (1) to perform an initial problem investigation that characterizes the problem to solve; (2) to provide a solution design suitable to solve those problems; and (3) to validate if the proposed solution satisfies the problematic phenomena previously analysed.

The research methodology is explained by means of regulative cycles [30] that were conceived in order to answer the research questions indicated above. Figure 2 presents the research methodology described, where the regulatory cycle can be observed.

- The main cycle of the research methodology is an engineering cycle (EC1: Design a model-driven method to include interaction based on gestures) since this proposal focuses on the development of a new artefact (method). Some tasks are related with this cycle:
  - i. Problem investigation, described in chapter 2 (“Theoretical Framework”), permits to obtain an answer to RQ1 and RQ2 research questions,
  - ii. Solution design, described in chapter 3 (“State of art”), chapter 4 (“gestUI, a model-driven method”) and chapter 5 (“A tool support”), this task permits to obtain an answer to RQ3 research question.

Two research cycles have been defined:

- The first research cycle (RC1: Validation of the proposed method with an empirical evaluation) describes the process that will be developed to validate the proposed method by means of an empirical evaluation, described in chapter 6 (“An empirical evaluation of gestUI”). It permits to obtain an answer to RQ4 research question.

- The second research cycle (RC2: Technical action research to validate the method in an industrial context) corresponds to the process where the proposed method is applied to a case study provided in the project “Capability as a Service” (CaaS), described in chapter 7 (“Technical Action Research”). It permits to obtain an answer to RQ4 research question.

## 1.6 Expected Contributions

Model-driven engineering (MDE) is a software development approach that provides an environment that ensures the use of models throughout the development process of software systems [34]. The essential idea of MDE is to shift the centre of attention from code to models [35] [36]. The software systems, or part of them, can be automatically generated using an abstract description and transformation rules. Using MDE is possible to abstract the technological diversity that there is in the application level of devices with gesture-based interaction.

In this thesis, a process to define custom gestures and to include gesture-based interaction based on the foundations of MDE is proposed. Specifically, this thesis provides as contributions:

Contribution 1: **a model-driven method** to define custom gestures and to include gesture-based interaction in user interfaces of software system. The process is carried out from the initial specification of custom gestures, based on metamodels, and using model transformations to obtain a gesture-based user interface by means of automatic generation of source code.

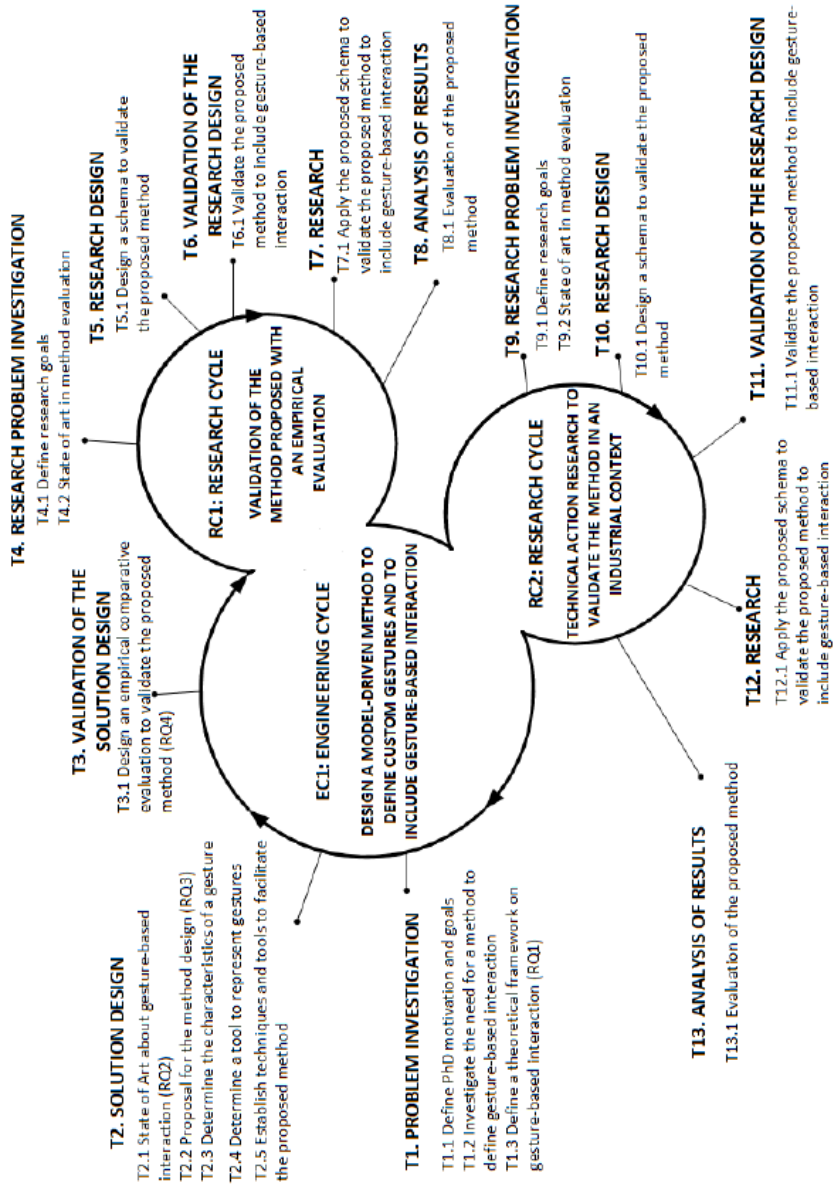


Figure 2. Overview of the research methodology



Contribution 2: a **tool** to support the model-driven method described in Contribution 1.

Contribution 3: the **validation** of the method proposed by means of an empirical comparative evaluation

Contribution 4: the validation in an industrial-context by applying a technical action research (TAR).

A more wide description about the contributions is included in Chapter 8.

## 1.7 Thesis Context

This research work has been developed in the context of the PROS Research Centre (Centro de Investigación en Métodos de Producción de Software), and DSIC (Departamento de Sistemas de Información y Computación) of the Universitat Politècnica de València, Spain.

This work has been supported by the Universidad de Cuenca and Secretaría Nacional de Educación Superior, Ciencia y Tecnología - SENESCYT of Ecuador, and received financial support from the Generalitat Valenciana under Project IDEO (PROMETEOII/2014/039) and the Spanish Ministry of Science and Innovation through the DataMe Project (TIN2016-80811-P).

## 1.8 Thesis Outline

The thesis comprises three parts, according to Design Science Methodology: Part I (Problem Investigation), Part II (Solution Design) and Part III (Validation of the Solution). Therefore, this thesis has been structured as follows.

### Part I: Problem Investigation

Chapter 1 (**Introduction**). This chapter describes the problem statement, objectives of the thesis, research questions, and research goals. Additionally, we describe the research methodology applied to the thesis, and the thesis context.

Chapter 2 (**Theoretical Framework**). In this chapter we include a theoretical framework in order to establish a commitment about the terminology defined in this thesis. This chapter includes a description of the technologies and concepts in which is based the development of this thesis.

Chapter 3 (**State of Art**). This chapter includes a review of the literature to highlight relevant advances in respect of development of user interfaces with gesture-based interaction considering the model-driven paradigm. Additionally, this chapter facilitates a common understanding around the topics of the thesis.

#### Part II: Solution Design

Chapter 4 (**gestUI: A model-driven method**). In this chapter we describe the **model-driven method** proposed to include gesture-based interaction in user interfaces of system software. We specify the model-driven method architecture considering details and formalisms like metamodel, business rules applied in the model and the constraints included in the method.

Chapter 5 (**gestUI Tool Support**). In this chapter we outline the tool support that has been developed for gestUI. It includes a description about the components of the tool support and a user guide explaining their functionalities.

#### Part III: Validation of the proposal

Chapter 6 (**Empirical evaluation**). This chapter describes the validation performed for the gestUI method. We detail the validation process highlighting the use of empirical software engineering to formally validate the proposed method. The empirical evaluation is performed as a comparative process between a code-centric method and the model-driven method.

Chapter 7 (**Technical Action Research**). This chapter presents the process to follow in the validation of gestUI method performed using Technical Action Research.

Part IV: Final Part

Chapter 8 (**Conclusions, Contributions and Future Work**). This chapter summarizes the contributions of this work and presents the conclusions of the thesis. Additionally, in this chapter we describe the future work.

---

THEORETICAL  
FRAMEWORK

2

The topics covered in this chapter are:

- 2.1 Overview
- 2.2 A theoretical framework for Human-computer interaction
- 2.3 A theoretical framework of Model-driven paradigm
- 2.4 Summary



## Chapter 2. Theoretical Framework

### 2.1 Overview

With the aim of maintaining a better comprehension with the people that read this thesis, we need to obtain a same conceptual commitment. Hence, a theoretical framework becomes vital.

Theoretical frameworks have been widely used and proposed aiming at defining the concepts that relies on a certain theory in order to facilitate conceptual commitment.

Since this thesis deals with the definition of a model-driven method to define custom gestures and to include gesture-based interaction in software systems user interfaces, we establish a theoretical framework for model-driven development and we also provide a theoretical framework for human-computer interaction. Therefore, this work is placed in the intersection of three research areas that have some aspects in common (Table 4):

**Table 4. Related areas in the thesis**

Included definitions	Research Area
Model-driven method	Model-driven Development
Gesture-based Interaction	Human-Computer Interaction
Software system	Software Engineering

This thesis is based on different concepts and technologies from these areas. With the aim of clarifying the foundations in which this thesis relies and to provide an adequate theoretical framework for understanding the overall work, different concepts and techniques are introduced in this chapter.

It is important to clarify that in this thesis the word software refers to software systems and gesture refers to custom gestures.

The remainder of this chapter is organized as follows: Section 2.2 describes a theoretical framework about Human-Computer

Interaction, including a brief description about gesture and gesture-based interaction. Section 2.3 includes a theoretical framework about concepts related with Model-driven paradigm. Finally, the summary of this chapter is included in Section 2.4.

## 2.2 A theoretical framework for Human-Computer

### Interaction

In this section we define a set of terms that describes how Human-Computer Interaction is applied to obtain gesture-based interfaces. This section includes definitions related with gestures, algorithms and tools available to recognise gestures, which permit to have a better comprehension about this topic.

Human-computer Interaction (HCI) is the study of the interaction between users and computers [37]. The interaction is mainly done in the user interface. According to Karray et al. [38], an interface mainly relies on number and diversity of its inputs and outputs which are communication channels that enable users to interact with computers via this interface. There are three categories of modalities: visual-based, audio-based and sensor-based. Considering types of interaction that have been developed in the last years, we take into account two of them: (i) traditional, using keyboard and mouse (sensor-based), and (ii) gesture-based, using gestures sketched by the users with their fingers or a pen/stylus on a touch-based surface.

#### 2.2.1 Gestures related definition

A gesture is considered as a primary element in the architecture of devices with support of gesture-based interaction. We consider two definitions about gestures:

- Oxford English Dictionary [39] defines gesture as “a movement of part of the body, especially a hand or the head, to express an idea or meaning”.
- Gesture is referred as a motion of the body (o some part of the body) that somebody does with the aim of communicating with other person.

A gesture catalogue is a list of gestures including descriptive information of each gesture.

Gesture description languages are significant for the correct execution of interactions by end-users, the preservation of techniques by designers, the accumulation of knowledge for the community, and the engineering of interactive systems [40].

Gesture recognition: According to Gillian et al. [41], gesture recognition is a powerful tool for human-computer interaction. Gesture recognition in most systems has been done by writing code to recognize the particular set of gestures used by the system [7].

### **2.2.2 Classification of gestures**

Some classifications of gestures are reported in the related literature. We consider those that help us to define a gesture representation to use it in our work.

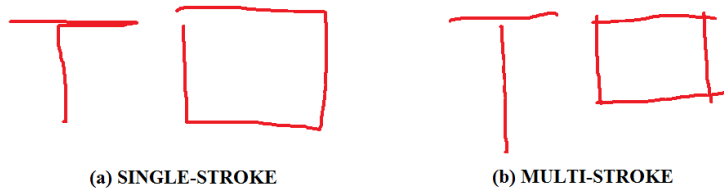
According to Kaushik et al. [42], a gesture can be classified as static or dynamic. A dynamic gesture changes over a period of time while static gesture is observed at the short interval of time.

Additionally, a gesture can be classified as discrete or continuous. A gesture is considered as discrete if the start and stop of the gesture is defined, usually with the press and hold of a widget while the gesture is carried out [43], for example, a double tap. A gesture is considered as continuous if it takes place over a period of time, for example, a scroll or a custom gesture [44].

According to Nacenta et al. [45], there are three types of gestures that can be included in a user interface: predesigned, stock and user-defined (custom). They demonstrate that users prefer user-defined gestures rather than stock and pre-defined gestures. Although user-defined gestures offer better memorability, efficiency and accessibility than pre-defined gestures, they have received little attention in the literature [46].



According to the taxonomy of gestures proposed by Karam et al. [11], semaphoric gestures refer to strokes or marks made with a mouse, pen or finger. This type of gesture is further classified as single-stroke and multi-stroke, according to the number of strokes required to sketch them (Figure 3).



**Figure 3. Types of semaphoric gestures**

According to Zhai et al. [47], a stroke gesture is commonly encoded as a time-ordered sequence of two-dimensional points with coordinates  $(x, y)$ . Optionally, stroke gestures can also have time stamps as the third dimension so the sampling points are encoded as  $(x, y, t)$  if the temporal aspect of a gesture is to be preserved and used.

Karam et al. [11] propose a style-based gesture classification that includes the next types: deictic, manipulations, semaphores, gesticulation, and language gestures.

- (a) Deictic gestures involve pointing to establish the identity or spatial location of an object within the context of the application domain. The application domain can include desktop computer, virtual reality applications or mobile devices for example.
- (b) A manipulative gesture is one whose intended purpose is to control some entity by applying a tight relationship between the actual movements of the gesturing hand/arm with the entity being manipulated. Manipulations are mainly dynamic and can occur either in a 2-D interaction using a device direct manipulation, or in a 3-D interaction.
- (c) A semaphoric gesture is any gesturing system that employs a stylized dictionary of static or dynamic hand or arm gestures. Semaphoric gestures can involve static poses or dynamic

movements. These types of gestures can be performed using a hand, fingers, arms, the head, feet.

- (d) The act of gesticulating is regarded as one of the most natural forms of gesturing and is commonly used in combination with conversational speech interfaces.
- (e) Gestures used for sign languages are often considered independent of other gesture styles since they are linguistically based and are performed using a series of individual signs or gestures that combine to form grammatical structures for conversational style interfaces.

Many of the existing systems do not focus on a single style of gesture interaction but rather employ a variety of gestures that are result of combining two or more gestures.

In this thesis, we consider gestures defined as: dynamic, continuous, user-defined and semaphoric (multi-stroke). We consider custom gestures that can be performed on a touch-based surface using one finger of the user or a pen/stylus. These gestures are used to issue commands, which are the names of the executable computing functions issued by the user.

### **2.2.3 Gesture recognition algorithms**

The ways of recognizing the gesture can be considered as a significant progress of the technology. Progress of image processing technology has played an important role here. Gestures have been captured by using infrared beams, data glove, still camera, wired and many interconnected technologies like gloves, pendant, infrared signal, network server, etc., in the past [10]. Computer application operating was the main target in the early stage. But now it is widely accepted for ambient device and ubiquitous computing.

In our work, we consider algorithms for touch-based gesture recognition because we use custom touch-based gestures to include gesture-based interaction in user interfaces of software systems. This section describes in a brief way, some of the well-known algorithms.

- Rubine algorithm: This algorithm was created in 1991 by D. Rubine. It is one of the first algorithms developed to recognise mouse and pen-based gestures [7]. An important feature of this algorithm is that gestures are not described programmatically but they are learned by examples. It employs a statistical method of gesture recognition based on a set of 13 geometric features [48]. It has been used for recognising single-stroke gestures like the unistroke or Grafitti alphabets [49]. It also allows the user to define a gesture through demonstration.
- \$N algorithm: The \$N Multistroke Recognizer [50] is a 2-D stroke recognizer designed for rapid prototyping of gesture-based user interfaces. Simple geometry and trigonometry are used to perform template matching between stored templates and entered candidates, giving \$N a deterministic quality whereby candidates that look most like their templates are usually recognised as such [50]. Other algorithms with a similar philosophy are: \$1 [51], \$N-Protractor [52], \$P [53].
- SiGeR: The SiGeR (Simple Gesture Recogniser) algorithm was developed by W. Swigart for the Microsoft Developer Network [54]. This algorithm classifies gestures based on regular expressions and describes them according to the eight cardinal points and statistical information. These regular expressions are then applied to input gestures and, in the case that a class description matches the input string, the corresponding gesture class is returned as a result [48].

In this thesis we adopt \$N as gesture recognition algorithm because it is a simple algorithm that is a good solution to prototyping of user interfaces. \$N does not require many computer resources which is a very important feature when the target device is a mobile one.

#### **2.2.4 Gesture-based interaction**

Natural User Interface is an emerging computer interaction methodology which focuses on human abilities such as touch, vision, voice, motion and higher cognitive functions such as expression,

perception and recall [55]. Facial expressions, posture, and gestures in particular have been recognized as an important modality for non-verbal communication [19].

Broadly speaking, there are two extremes of interaction: one in which the user interacts consciously and explicitly with the system; and at the other extreme, the user interacts unconsciously or implicitly. In explicit interaction, a user interacts with a software application directly by manipulating a GUI, running a command in a command window or issuing a voice command. In short, the user intentionally performs some action [56].

In this thesis, we consider the gesture as an element of communication between the user and devices that support an explicit interaction by means of gestures, because the interaction between user and computer is done by the user in a manner explicit by means of actions on the screen.

### 2.3 A theoretical framework of Model-Driven paradigm

In this section we define a set of terms that describes how Model-driven paradigm is applied to the work described in this thesis. This section includes definitions related with models and model transformations, which permit to have a better comprehension about this topic.

As the MDD paradigm promotes to specify software systems by means of models, we start this theoretical framework defining what a model is. Then, we describe other basic concepts related with the model-driven paradigm used in this thesis. The definitions are based on the Object Management Group definitions [57].

#### 2.3.1 Model related definition

A system is a collection of parts and relationships among these parts that may be organized to accomplish some purpose [58].

A model is a description of a system or part of a system written in a well-defined language [59]. A metamodel: A model of models [58].

A well-defined language: it is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer [59]. A model, both source and target, is expressed in a language, for example, XML.

Model-driven: it describes an approach to software development whereby models are used as the primary source for documenting, analysing, designing, constructing, deploying and maintaining a system [57]. In late 2000, the MDA (Model-Driven Architecture) initiative was launched by OMG (Object Management Group) to promote using models as the essential artefacts of software development. A new paradigm in software development where models are the primary software artefacts and transformations are the primary operation on models [60] is available.

MDA is based on four principles [61]:

- The models expressed in well-defined notation are a key to understanding software systems.
- The implementation of software systems can be organized around a set of models making it necessary to carry out a series of transformations between models, organized in an architectural framework of layers and transformations.
- A formal basis for describing models in a set of metamodels facilitates meaningful integration and transformation among models, and is the basis for automation through tools.
- Acceptance and broad adoption of this method, based on models, requires industry standards to provide openness to consumers and fostering competition among providers.

There are two main flavours based on this paradigm: MDD (Model-Driven Development) and MDE (Model-Driven Engineering).

- Model-Driven Engineering (MDE) describes software development approaches in which abstract models of software are created and systematically transformed into concrete implementations [62].
- Model-driven Development (MDD) focuses on the construction of models, specification of transformation rules, tool support and automatic generation of code and documentation [63]. In this case, the software development process can be viewed as a sequence of model transformations. Recent studies indicate that the adoption of Model-Driven Development (MDD) is widespread [64].

### **2.3.2 MDA Conceptual framework**

OMG provides a conceptual framework and a vocabulary for MDA and it defines a specific set of layers and transformations. In this schema, it identifies four layers (see Figure 4) that raise the level of abstraction of traditional platform dependent design [65]: Computation Independent Model (CIM), Platform Independent Model (PIM), Platform Specific Model (PSM) and Implementation Specific Model (ISM).

In this work, we use the following three layers:

- Platform Independent Model (PIM) is a model with a high level of abstraction that is independent of any implementation technology [59]. System developers use this language for precisely describing the system using a technology independent view [66].
- Platform Specific Model (PSM) is a model that adds to the PIM the technological aspects of the target platforms [67].
- Implementation Specific Model (ISM): that describe the last detail of programming [67].

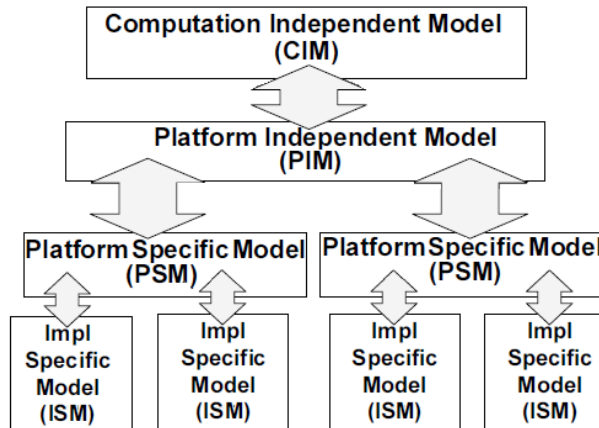


Figure 4. MDA Layers

The definition of PIM and PSM is motivated by the constant change in implementation technologies and the recurring need to port software from one technology to another [68].

In summary, MDA is organized around a PIM which is a specification of a system in terms of domain concepts. These domain concepts exhibit a specified degree of independence of different platforms of similar type. The system can then be translated towards any of those platforms by transforming the PIM to a PSM. The PSM specifies how the system uses a particular type of platform [69].

### 2.3.3 Model Transformations

Model transformation: It is a process consisting in to convert one or more source models (input) to one target model (output) [59].

Transformation rule: It is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language [59].

A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description

of how one or more constructs in the source language can be transformed into one or more constructs in the target language [59].

OMG has defined two transformations (see Figure 5):

- PIM to PSM transformation (it defines how a PIM can be converted to a PSM)
- PSM to ISM transformation (it is the code generation from the PSM). Due to the fact that PSM is expressed using technological terms, transformation to ISM is immediate.

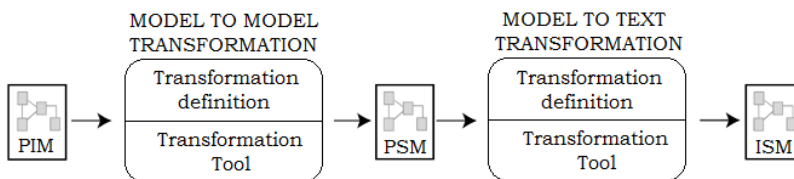


Figure 5. MDA Transformations

The mentioned transformations can be identified as:

- Model-to-Model transformation (M2M): A model transformation is a mapping of a set of models onto another set of models or onto themselves, where a mapping defines correspondences between elements in the source and target models [36]. The important role of model transformations motivates the effort that OMG took to define the standard language for model transformations called QVT MOF 2.0 Language [68].
- Model-to-text transformation (M2T) which generates source code from models, thus lowering the abstraction level of modelling artefacts and making them executable [70]. The standard MOF M2T Language (OMG) [71] specifies the M2T transformation, which is an important type of model transformation [72]. M2T is used to implement code and documentation generation in development of software systems [73]. In the transformation process, the source is a PSM (platform specific model) and the target is a source code, such as Java, C#, etc.



### 2.3.4 Transformation Language

In this thesis, we consider two types of transformation language: (i) model-to-model transformation (MMT) language, and (ii) model-to-text transformation (MTT) language.

#### MMT Language: ATL

ATL (ATLAS Transformation Language) is a MMT language specified as both a metamodel and a textual concrete syntax. In the field of MDE, ATL provides developers with a means to specify the way to produce a number of target models from a set of source models (see Figure 6).

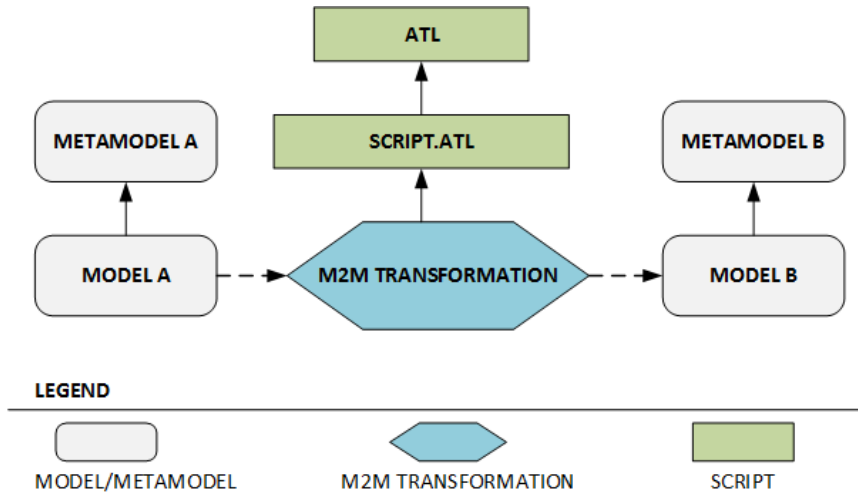
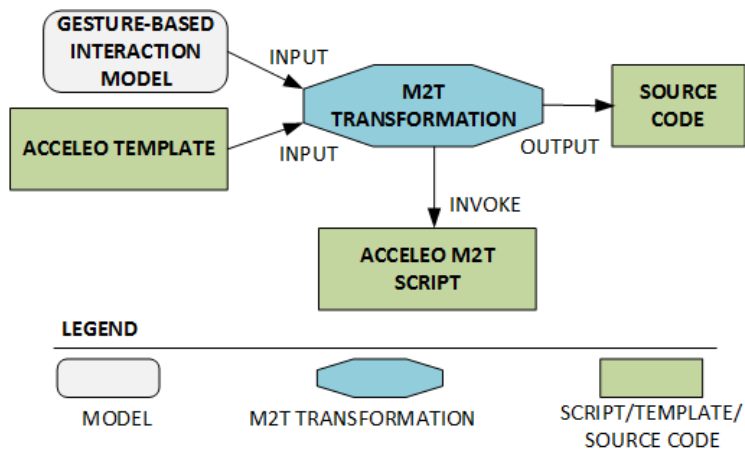


Figure 6 Model-to-Model Transformation

The ATL language is a hybrid of declarative and imperative programming. Declarative style is the preferred of transformation writing. In this thesis, we use ATL to write the model-to-model transformation.

#### MTT Language: Acceleo

Acceleo is a MTT template-based language for defining code-generation templates. The language supports OCL as well as additional operations helpful for working with text-based documents in general.



**Figure 7. Model-to-text transformation**

A model-to-text transformation in Acceleo basically consists in a mapping between each object in the input model and a string of characters that represents the output (Figure 7).

In this thesis, we use Acceleo to write the model-to-text transformation and to obtain the new version of the user interface source code containing gesture-based interaction.

## 2.4 Summary

In this chapter, we have presented two theoretical frameworks to use in this thesis: in first place, we introduce a theoretical framework for Human-Computer Interaction including concepts related with human-computer interaction including: concepts related with gestures, and concepts related with gesture-based interaction (Section 2.2). In second place, we introduce a theoretical framework for Model-driven paradigm including concepts by describing standards related with model-driven paradigm, proposed by the OMG (Section 2.3).

The terms defined in this section are used in Chapter 4 and Chapter 5, where the model-driven method design is presented and the tool support is described. Also Chapter 6 uses the terms for describing the validations performed in this thesis.

These terms are included because these domains are related with the objectives of this thesis and also with the aim of giving a better comprehension about this work.

STATE OF  
ART

3

The topics covered in this chapter are:

- 3.1 Motivation
- 3.2 Gesture representation
- 3.3 Gesture recognition tools
- 3.4 The role of gesture-based interfaces in Information Systems Engineering
- 3.5 Model-driven Engineering in Human Computer Interaction
- 3.6 Evaluation between model-driven paradigm and other methodologies
- 3.7 Technical action research to validate software systems
- 3.8 Range of Improvements
- 3.9 Summary



## Chapter 3. State of Art

### 3.1 Motivation

This chapter describes the most important approaches that support the design and development of software systems with gesture-based interaction. Once we have analysed in Chapter 2 the general application domains in which this work is based, we analyse the specific proposals in the domains that are closely related to our thesis. This analysis allows us to determine the way in which each proposal addresses the aspects that are central in our approach.

The work related to this thesis can be analysed regarding three research areas mentioned in Chapter 2: (i) gesture representation (research area: Human-Computer Interaction); (ii) the definition of a model-driven method to generate user interfaces with gesture-based interaction (research area: Model-driven paradigm in Human-Computer Interaction) and, (iii) the type of method employed to perform the verification (research area: Empirical Software Engineering).

In this chapter we review the previous work regarding three aforementioned dimensions.

The remainder of this chapter is organized as follows:

- i. Regarding first dimension, Section 3.2 presents related works about gesture representation and gesture recognition tools.
- ii. Regarding second dimension, Section 3.3 describes the role of the gesture-based interfaces in Information Systems Engineering and, Section 3.4 details the related work about Model-driven Engineering in Human-Computer Interaction.
- iii. Regarding third dimension, Section 3.5 describes related work about the evaluations between model-driven paradigm and other methodologies and, Section 3.6 presents related work about the use of technical action research to validate software systems in an industrial context.

Section 3.7 describes the range of improvements to solve the problems found in the related work. Finally, Section 3.8 presents the summary of this chapter.

### 3.2 Gesture representation

Firstly, we decide consider in our work touch-based gestures to perform actions on a touch surface of any device supporting this type of interaction.

Techniques available in the literature for gesture representation are important in our thesis because we are interested in to know the different ways employed by other authors to describe touch-based gestures in order to adopt the more adequate to our work. This adoption depends of the simplicity and accuracy considered to define a gesture.

Our work is not related with the definition of a gesture representation nor with a method to recognize gestures. We are interested in find a solution to the problems mentioned in Chapter 1 when the software engineers decide to include gesture-based interaction in user interfaces of software systems.

Hence, in this section, we take in account methods reported in the related literature to represent this type of gestures. Three categories are considered about this topic (Table 5):

- (i) Based on regular expressions. A regular expression is an expression formed by elements such as ground terms (basic buildings blocks), operators, symbols, etc.
- (ii) Based on a language specification. A gesture is represented using a language specification, typically XML.
- (iii) Based on a demonstration. A gesture is represented by means of information obtained when it is sketched on a touch-based surface. Then, the gestures are tested and refined, and, once the users are satisfied with them, include their definition in the applications.

In the next paragraphs, we describe works reported about gesture representation using each aforementioned method.

Based on regular expressions.

There are several works reported in the related literature that represent gestures by means of regular expressions:

Spano et al. [74] propose a compositional, declarative meta-model for gesture definition based on Petri Nets. This proposal allows constructing complex gestures from a well-defined set of building blocks and composition operators. The definition starts with ground terms representing the set of basic features observable through a specific device. The user interface behaviour can be associated to the recognition of the whole gesture or to any other subcomponent, addressing the problem of granularity for the notification events. Sample applications have been developed for supporting multitouch gestures on iOS and full body gestures with Microsoft Kinect.

Lascarides and Stone [75] present a formal semantic analysis of iconic gestures employing a multidimensional matrix whose rows contain values that describe aspects of a gesture's form. The contribution of their work is to meet the challenge, implicit in descriptive work on nonverbal communication, of handling gesture within a theoretical framework that's continuous with and complementary to purely linguistic theories.

Giorgolo [76] has a complementary proposal of Lascarides [75] to represent iconic spatial gestures based on formal semantic. Giorgolo provides a more precise description of the mechanism of gesture meaning determination.

Kin et al. [77] propose Proton, a framework which allows a declarative and customised definition of multi-touch gestures using regular expressions composed of touch event symbols. A gesture can be represented as a regular expression describing a sequence of touch events.



Kin et al. [78] describe Proton++, a declarative multi-touch framework that allows developers to describe multi-touch gestures as regular expressions such event symbols. Their work includes a custom declarative gesture definition; it is based on the Proton framework.

Spano et al. [79] describe GestIT, a framework to represent a gesture as a declarative and compositional definition for different platforms. This framework shares with Proton++ the compositional and declarative approach. A gesture is defined through an expression that can be composed with a set of operators and a set of ground terms.

Swigart [54] developed SiGeR (Simple Gesture Recogniser) for the Microsoft Developer Network to describe gestures with the eight cardinal points (N, NE, E, SE, S, SW, W and NW) and some statistical information. A regular expression is created out of this description representing a gesture.

#### Based on a language specification

There are several works reported in the related literature that represent gestures by means of some language specification:

Signer et al. [48] describe iGesture, a Java framework for the development and deployment of stroke-based gesture recognition algorithms. iGesture has two schemas to store the gesture definition: (i) an open source object database as a primary storage container and, (ii) XML which simply serialise the data object into a document based on the x-stream Java library. Additionally, this framework has included a functionality to import or export gestures definition written in XML.

Puype [80] extended the iGesture framework in order to include support to multi-modal composite gestures. In this context, gestures can be defined using XML and XML Schema.

Gesture ML [81] or Gesture Markup Language (GML) developed by Ideum is an extensible language, based on XML, used to define multi-

touch gestures that describes interactive object behaviour and the relationships between objects in an application.

Görg et al. [82] in their work adapt Labeled Deductive System (LDS) to represent a gesture. LDS provides a framework for expressing logics by using a pair (label: formula). This schema permits define multi-touch gestures by means of a parametrised formula.

Hachaj et al. [83] introduce a new approach for human body poses and movement sequences recognition using Gesture Description Language (GDL). This language consists of rules set, where each rule has the logical expression and conclusion enabling the description of any body's poses and gestures with assumption that gesture can be partitioned into sequences of poses. The description is contained in a script using a proprietary language.

GDML (Gesture Definition Markup Language) allows a declarative description of the sequence of events that the device senses for recognising a custom touch gesture. GDML is a proposed XML dialect that describes how events on the input surface are built up to create distinct gestures [84].

Kammer et al. [85] describe GeForMT (Gesture Formalization for MultiTouch) which is defined using semiotic with three components and their scope: syntactic (symbols), semantics (meaning) and pragmatics (interpretation). This language permits the representation of multi-touch gestures.

#### Based on demonstration

There are several works reported in the related literature about gesture representation by means of demonstration:

Lü et al. [86] describe Gesture Coder which is a tool for programming multi-touch gestures by demonstration. Instead of writing code or specifying the logic for handling multi-touch gestures, a developer can demonstrate these gestures on a multi-touch device, such as a tablet.

Gesture Coder automatically generates user-modifiable code that detects intended multi-touch gestures and provides callbacks for invoking application actions. Gesture Coder allows to the developers define a gesture by demonstration, test the generated code, refine it, and, once they are satisfied with this definition, integrate the code in their applications. Multi-touch gestures are defined using this specification by means of information supplied by multi-touch interaction on a two-dimensional Surface.

Lü et al. [87] describe Gesturermote, a technique for interacting with remote displays through touch gestures on a handheld touch surface. Gesturermote supports a wide range of interaction behaviours, from low pixel-level interaction such as pointing and clicking, to medium-level interaction such as structured navigation of a user interface, to high-level interaction such as invoking a function directly (e.g. shortcuts).

Wobbrock et al. [51] describe \$1 that is easy, cheap, and usable almost anywhere in about 100 lines of code. The \$1 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.

Anthony et al. describe \$N-Protractor [52] and \$N [50] to add support for multi-stroke gestures. Specifically, \$N gesture recogniser algorithm is a lightweight, concise multistroke recogniser that uses only simple geometry and trigonometry to perform template matching between stored templates and entered candidates. \$N is a significant extension of the \$1 unistroke recogniser by Wobbrock et al. [51].

**Table 5. A summary related with gesture representation**

Category	Author	Type of representation	Application	Type of gestures	Platform
Based on regular expressions	Spano et al. [74]	A compositional, declarative meta-model for gesture definition based on Petri Nets.	Applications using touch-based and full body gestures.	Multi-touch gestures	iOS Kinect
	Lascarides and Stone [75]	A multidimensional matrix that describe aspects of a gesture's form.	Authors analyse the combination of hand gesture with the speech of a user.	Iconic gestures	Not mentioned
	Giorgolo [76]	Based on formal semantic. It is a complementary proposal of the Lascarides' work	Not mentioned	Iconic spatial gestures	Not mentioned
	Kin et al. [77]	A declarative and customised definition of gestures based on regular expressions.	Three proof-of-concept applications using Proton.	Multi-touch gestures	Not mentioned
	Kin et al. [78]	Describing multitouch gestures as regular expressions of touch event symbols.	Some basic applications to demonstrate the flexibility of Proton++.	Multi-touch gestures	Not mentioned
	Spano et al. [79]	Using GestIT, a gesture can be represented using an expression composed with as set of operators and a set of ground terms.	To recognise gestures included in WIMP style user interfaces.	Multi-touch and full body gestures	Different recognition platforms
	Swigart [54]	Using SiGeR, a gesture can be described using the eight cardinal points and some statistical information.	To recognise gestures included in a Tablet PC.	Multi-touch gestures	Tablet PC

Based on a language specification					
Signer et al. [48]	Using some existing algorithms included in iGesture describes gestures with a grid-based signature.	Palm Graffiti letters and numbers, the Microsoft Application Gestures and a customised set of multi-stroke gestures	Single and multi-stroke gestures.	Not mentioned	
Puype [80]	It is an extended version of iGesture. Gesture can be defined using XML and XML Schema.	Examples of applications are: multimedia player, PaperPoint presentation tool.	Composite and Multi-modal gestures	Touch tables	
Ideum [81]	Using GestureML a multi-touch gesture can be described.	Examples of application are: Windows 8 games	Multi-touch gestures	Post-WIMP NUI	
Görg et al. [82]	Using the format Labeled Deductive System (LDS) can be described a multi-touch gesture.	A navigation system	Multi-touch gestures	A multi-touch input device (e.g. iPhone)	
Hachaj et al. [83]	GDL language is used for syntactic description human body poses and movement sequences.	Application related with detection of movement, recognition of hand clapping and recognition of waving gestures	Human body poses and movement sequences	Kinect sensor and OpenNI Framework software.	
NUI Group [84]	Gesture Definition Markup Language is used to describe gestures.	Not mentioned	Standard gestures	Not mentioned	
Kammer et al. [85]	Gesture Formalization for Multi-touch is used to describe multi-touch gestures.	Not mentioned	Multi-touch gestures	Not mentioned	

Based on demonstration		Gesture Coder which is a tool for programming multi-touch gestures by demonstration.	An example of a typical software development project.	Multi-touch gestures	Two-dimensional Surface
Lü et al. [86]	Gesture Coder is described to define touch gestures on a handheld touch surface.	Some basic applications to demonstrate the feasibility of the tool.	Touch gestures	Gesturemote could be used to control a TV interface.	
Lü et al. [87]	Gesturemote is described to define touch gestures on a handheld touch surface.	User interface prototypes	Single-stroke gestures	Touch surfaces	
Wobbrock et al. [51]	\$1 is used to define single-stroke gestures. A user's gesture results in a set of candidate points C, and we must determine which set of previously recorded template points T <sub>i</sub> it most closely matches.	User interface prototypes	Multi-stroke gestures	Touch surfaces	
Anthony et al. [50]	\$N and \$N-Protractor can be used to define multi-stroke gestures. \$N is a lightweight, concise multi-stroke recognizer that uses only simple geometry and trigonometry.	User interface prototypes	Multi-stroke gestures	Touch surfaces	
Vatavu et al. [53]	\$P is used to define multi-stroke gestures as a points cloud. \$P is a point-based recognizer instead of a stroke-based one, as are \$1 (one stroke) and \$N (N strokes).	User interface prototypes	Multi-stroke gestures	Touch surfaces	

Vatavu et al. [53] describe \$P as a gesture recognizer for user interface prototypes. \$P performs similarly to \$1 on unistrokes and is superior to \$N on multistrokes. In this case, a gesture is defined as a points cloud.

If we analyse the different representations of the gestures of each category, we can see the diversity of proposals reported by the related literature depending their application. Therefore, a solution to this problem is to use a single representation to the different possible scenarios where the gestures can be used.

Hence, in this thesis, we propose a model-driven approach in order to represent gestures with a high-level of abstraction, enabling platform-independence and reusability. By providing the proper transformations, it is possible to target several gesture recognition technologies. We focus on user-defined, multi-stroke, semaphoric gestures according to the taxonomy proposed by Karam in [11]. Also, we adopt \$N [50] as the gesture recognizer algorithm to include in the model-driven method proposed. Then, in this thesis we use XML to represent gestures because it is a standard language and it permits get multi-platform feature in the definition of gesture catalogue.

### 3.3 Gesture recognition tools

According Ruffieux et al. [88], gesture recognition precisely refers to a subset of the human activity and action recognition field and can be defined as the process by which specific gestures, intentionally performed by a user, are recognized and interpreted by a machine.

Three of the most known gesture recognition tools are the following:

- quill [89]: It is a gesture design toolkit that supports Rubine gesture recognition algorithm. quill allows to the user define a gesture by demonstration. Figure 8 shows a screenshot of this tool which is divided in three areas: (1) training set, containing the gesture catalogue, (2) training example, containing the gesture sketched

by the user and, (3) the area showing the result of the gesture recognition.

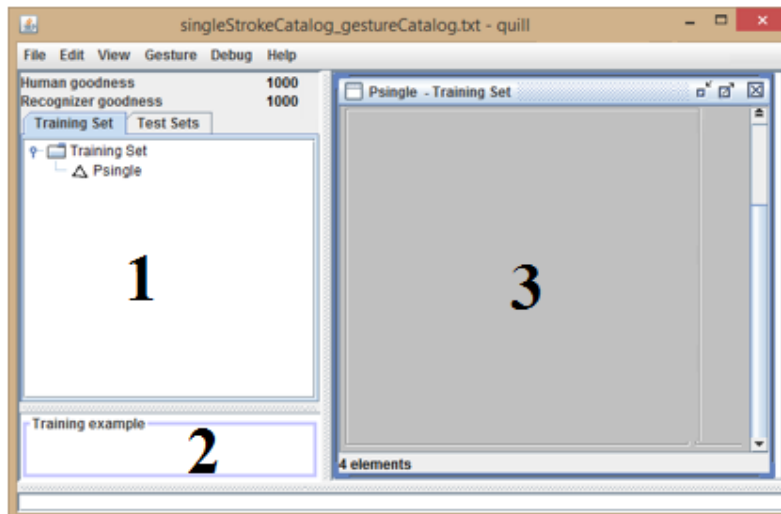


Figure 8. quill's main interface

- \$N [50]. The goal of \$N is to provide a useful, concise, easy-to-incorporate multi-stroke recogniser deployable on almost any platform to support rapid prototyping. \$N is based on demonstration to represent gestures.

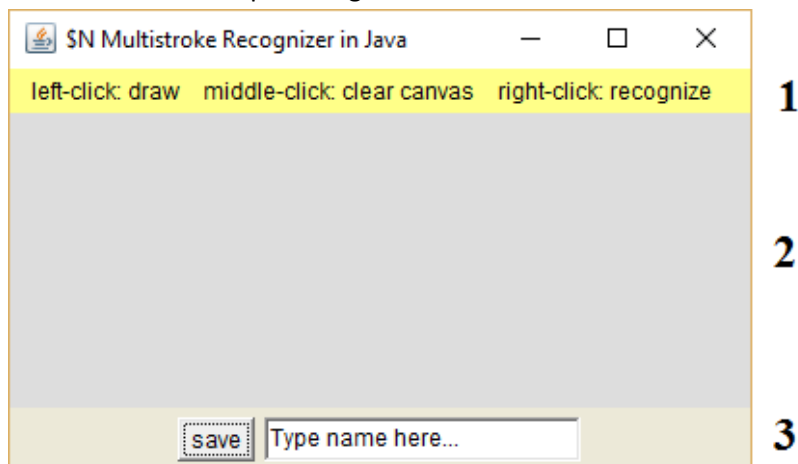


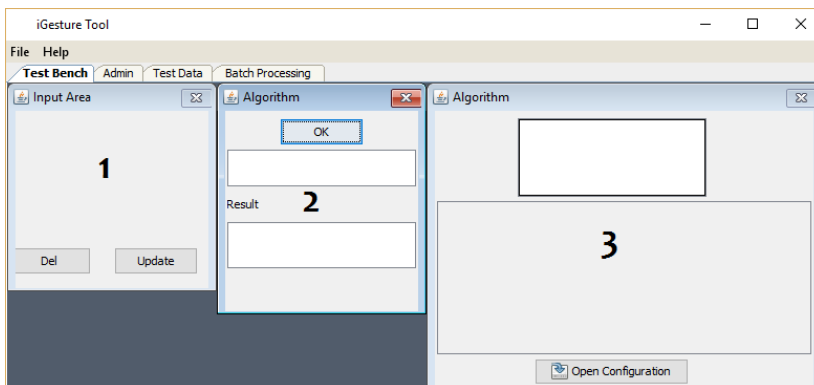
Figure 9. \$N's main interface

In this case, \$N main interface is divided in three zones (Figure 9): has an area where the user sketches gestures: (1) area in the upper



side of the interface containing messages about the process (e.g. messages for the user related with the sketched and recognised gesture), (2) area in the middle side of the interface containing a panel to draw gestures, and (3) area in the down side containing a text field and a button to save the gesture.

- iGesture [48]: It is a framework which supports the SiGeR algorithm and it is based on a set of modular components and some common data structures. Figure 10 shows a screenshot of this framework containing its components: (1) the input area where the user sketches the gesture, (2) the algorithm specification zone, where the user can specify the algorithm to use in the process of gesture recognition and, (3) the area showing the result of the gesture recognition.



**Figure 10. iGesture main interface**

In this thesis, we use these gesture recognition tools in order to test the gesture definition obtained through model transformation and code generation. Additionally, we use \$N as gesture recognition tool adapting it in the user interface modified to support gesture-based interaction.

### 3.4 The role of gesture-based interfaces in Information Systems Engineering

Gesture-based interfaces can play two major roles in information systems engineering, depending on whether we intend to incorporate this natural interaction into (i) CASE tools or (ii) into the information systems themselves. In the former case, the interest is to increase the information systems developers' efficiency, whereas in the latter the aim is to increase information system usability, especially in operations in the field, where the lack of a comfortable office space reduces the ergonomics of mouse and keyboard. In both cases, gesture-based interface development methods and tools are needed.

Some studies reporting on the definition of methods to generate a user interface are described in the following paragraphs (Table 6):

UsiGesture, proposed by Beuvens et al. [90], allows a designer to integrate gesture-based interaction in an interface considering 2D pen-based gestures, but it lacks techniques to model, analyse or recognise gestures. The authors applied the method to developing a restaurant management tool.

Guimaraes et al. [91] proposes a process model for development of gesture-based applications. The proposed development process is defined in four steps inter-related and executed interactivity. The steps are: requirements definition, design definition, implementation and evaluation. The authors apply it to creating a puzzle game using a 3D coordinates system to move and fit the pieces of the puzzle using the hand of the user.

Nielsen et al. [92] describe in their work a method with two variants (technology-based and human-based) and provides guidelines for the definition and selection of gestures, based on ergonomic principles. The authors perform a study to decide which types of gestures are required for arbitrary applications. Therefore, they define a gesture vocabulary containing gestures performed by the hand of the user. The process consists in the following steps: (i) find the functions to each

gesture, (ii) find logical gestures, (iii) process the data, (iv) benchmark of the gestures.

Bragdon et al. [93] describes GestureBar embeds gesture disclosure information in a familiar toolbar-based user interface. GestureBar's simple design is also general enough for use with any recognition technique and for integration with standard, non-gestural user interface component. The authors began the design process with a simple mockup, using Windows Presentation Foundation (WPF). The process consists of three steps: (i) prototype and iterative design, (ii) final design, (iii) content development.

Bhuiyan et al. [10] report the development of a Gesture Controlled User Interface (GCU) prototype application called Open Gesture to facilitate inclusive interface designs that are usable by the elderly and disabled. Open Gesture uses simple hand gestures to perform diverse range of tasks via a television interface. Therefore, authors apply it to an interactive television project.

If we analysed the related works included in this Section (summarised in Table 6), we can see the diversity of proposals reported by the related literature depending of some factors such as: development life cycle, methodology, type of gesture supported, application, tool support and platform. Therefore, this scenario complicates the process to define custom gestures and to include gesture-based interaction in user interfaces in software systems.

Hence, in this work, we propose a similar flow to that described in [91], but we define custom gestures by using models and we automate the implementation of gesture-based interfaces by means of model transformations.

**Table 6. Summary of works related with role of gesture-based interfaces in Information Systems Engineering**

Author	Stages in the development of user interfaces	Methodology	Type of gesture supported	Application	Tool support	Platform
Beuvenet al. [90]	Seven steps are defined to obtain a gesture-based user interface.	Based on a Java rendering engine to obtain final user interface (FUI) by using a visual editor	2D pen-based gesture	A system to manage a menu in a restaurant	Based on XWT	Not mentioned
Guimaraes et al. [91]	Four steps are considered: requirements definition, design, implementation and evaluation.	An adaptation of the conventional development process	3D gestures performed with the hand of the user	KIPuzzle, a puzzle using a 3D coordinates system	Not mentioned	Microsoft Windows
Nielsen et al. [92]	Four steps are included in the process: find the functions, find logical gestures, process the data, benchmark.	A conventional development process of user interfaces	Gestures performed with the hand of the user	A software to use in a pub	Not mentioned	Not mentioned
Bragdon et al. [93]	Three steps are defined: design principles, prototype and iterative design, final design, content development	A conventional development process of user interfaces	2D pen-based gestures	Lineogrammer, an application to draw simple diagrams	Gesture Bar	Microsoft Windows
Bhuiyan et al. [10]	Not mentioned	Not mentioned	Simple hand gesture	Open Gesture	Not mentioned	Programmable digital Television equipment

### 3.5 Model-driven engineering in Human-Computer Interaction

This section reviews the role of model-driven engineering (MDE) in Human-Computer Interaction in which models are used to create a user interface that includes user interaction. Several studies have reported on the use of model-driven engineering in HCI to design user interfaces with this type of interaction (Table 7).

Aquino et al. [26] emphasize the importance of interaction modelling on the same level of expressiveness as any other model involved in the development life cycle of an interactive application. They define the presentation model of the OO-Method [94] as an abstract model from which the model compiler can automatically generate a user interface for different interaction modalities and platforms, although they do not include an explicit reference to a type of interaction modality (e. g., graphical, vocal, tactile, haptic, and multimodal).

Deshayes et al. [95] propose the use of MDE and model execution in the context of human-computer interaction (HCI) by means of heterogeneous models obtained with the ModHel'X modelling environment for developing a simple HCI application for gesture-based interaction. Their application makes it possible to browse a virtual book using gestures (e.g., swiping, moving) in Microsoft Kinect.

Coutaz et al. [96] include a report regarding user interface plasticity and MDE, in which three information spaces are defined: the user model, environment model, and platform model. The platform model considers the possible interactions that can be included in a user interface. This report also includes a description of models that have been defined with the aim of creating user interfaces. It also mentions the variety of interaction modalities currently available thanks to the diversity of technological spaces that can be included in the definition of concrete user interfaces.

Calvary et al. in [97] describe the relation between MDE and HCI in implementing user interfaces. In this context, they introduce the

models contained in a number of frameworks (e.g., UsiXML [98], CTTe [99]), one being the interaction model considered in the process of defining user interfaces. However, the interaction modality is not considered.

Valverde et al. [100], propose the Abstract Interaction Model that is added to the Presentation Model in the OO-Method. Two sub-models are considered to define the Interaction Model: the user model and abstract interface model. A set of interaction components is defined in the abstract interface model that define its interface with the software system. These components are conceptual modelling elements that describe the interaction behaviour expected by the user but not how it is implemented, so that this system does not include the interaction modality in the process of user interface generation.

Vanderdonckt [101] describes a MDA-compliant environment which considers a set of variables related to the development of user interfaces, one of which is related with interaction devices and styles. Interaction styles include the gesture recognition. However, he points out that an abstract user interface is independent of any interaction modality [102] so that an explicit reference to a specific type of interaction style is not considered.

All the works cited above mention the importance of using MDE and HCI to obtain user interfaces in a short time at a low cost. Although they also point out the need for a specification of an interaction modality, they do not include gestures in their proposals. We considered gesture-based interaction in this proposal in order to obtain a complete definition of user interfaces using MDE and HCI.

**Table 7. Summary of works related with Model-driven engineering in Human-Computer Interaction**

Author	Modelling notation	Methodology used in the process	Tool Support	Specified interaction type?	Code generation?
Aquino et al. [26]	Interaction modelling	Model-driven Engineering	OO-Method	Not included an explicit reference to interaction type	Automatic code generation
Deshayes et al. [95]	Static models of computation	Model-driven Engineering and Human-Computer Interaction	ModHel'X modelling environment	3D gestures	Not mentioned
Coutaz et al. [96]	User Model, Environment model, Platform model	MDE combined with code-centric method	Not specified	Not included an explicit reference to interaction type	Not mentioned
Calvary et al. [103]	User Model, Task Model, Interaction Model, Platform Model	Model-driven Engineering and Human-Computer Interaction	Not mentioned	Not included an explicit reference to interaction type	Yes
Valverde [100]	Abstract Interaction Model, Presentation Model, Interface Model	OO-Method	OlivaNova	Not included an explicit reference to interaction type	Not mentioned
Vanderdonckt [101]	Abstract user interface	MDA-compliant method	MDA-compliant environment	Not included an explicit reference to interaction type	Not mentioned

### 3.6 Evaluation between model-driven paradigm and other methodologies

The main goal of this section is to know how were performed other comparative evaluation processes between two or more methods of software development regarding the parameters defined for each process (e.g. variables, metrics, instruments).

In this sense, we analyse the related work about comparative evaluation between methodologies based on model-driven paradigm and others existing methodologies (e.g. traditional software development methodology) to develop software.

There are several works which report experiments to do this comparison, some of them are described in the following paragraphs (Table 8):

Kapteijns et al. [104], describe a case study of the development of a small middleware application in order to do a comparison between Model-Driven Development (MDD) implementation with regular third-generation programming. The MDD framework used, which is called XuWare, permits to generate “create-remove-update-delete” – CRUD functionality for Web applications from UML models. Results obtained show that MDD is well applicable to small-scale development projects under easily satisfactory conditions.

Bunse et al. [105], in their work describe a case study in order to compare MARMOT (based on MDD and CBD – component-based development) with RUP and Agile Development. In this evaluation, the subjects developed a small control system for an exterior car mirror. The metrics employed in the evaluation are: model-size, amount of reused elements, defect density, development effort. Their evaluation reveals that model-driven, component-oriented development performs well and leads to maintainable systems and a higher-than-normal reuse rate.



Ricca et al. [106], describe in their work a controlled experiment with the aim of investigating the effectiveness of Model-driven development during software maintenance and evolution activities. Participants (bachelor students) used two software systems (Svetofor and Telepay) and by means of UniMod obtained two new versions of these software systems. In this experiment, the results showed a marked reduction in time to complete the maintenance tasks, with no important impact on correctness, when UniMod is used instead of conventional programming.

Papotti et al. [27] describe a quantitative study in order to evaluate the impact of using model-driven code generation vs. traditional development of software systems to implement a web application. Results show that the development time to code generation is shorter than time required using traditional development.

Condori-Fernandez et al. [107] describe an empirical approach for evaluating the usability of model-driven tools. They propose a framework to evaluate the usability applying it to INTEGRANOVA, an industrial tool that implements a MDD software development method called the OO-Method. The authors report results about the usability evaluation in terms of efficiency, effectiveness and satisfaction within an experimental context.

Martinez et al. [108] describe a quasi-experiment in order to compare three methods (Model-driven, Model-based and Code-centric) developing the business layer of a Web 2.0 application. Results show that MDD approaches are the most difficult to use but, at the same time, are considered as the most suitable in long term. Additionally, these authors in [109] report a quasi-experiment in order to evaluate productivity and satisfaction when a group of Master students develop a Web application using three methods: code-centric, model-based (UML) and model-driven (OOH4RIA). Results show that the use of Model-driven Engineering practices significantly increase both productivity and satisfaction of junior Web developers, regardless of the particular application. Other work reported by these authors [110]

is about an empirical study on the maintainability of the Web applications. In this work, they compare Model-driven Engineering with Code-centric method by using OOH4RIA and Visual Studio .NET respectively. The results show that maintaining Web applications with OOH4RIA clearly improves the performance of the subjects.

Cervera et al. [111], in their work describe an empirical evaluation using TAM and Think Aloud methods with the aim of assessing usefulness and ease of use of MOSKitt4ME. In this evaluation the results were favourable, that is, MOSKitt4ME was highly rated in perceived usefulness and ease of use; the authors also obtained positive results with respect to the users' actual performance and the difficulty experienced.

Panach et al. [112] describe in their work an experiment in order to compare quality, effort, productivity and satisfaction of MDD and traditional development. Participants (last-year master students) built two web applications from scratch. Results obtained show that for small systems and less programming-experienced subjects, MDD does not always yield better results than a traditional method, even considering effort and productivity.

All these works describe comparative evaluations in order to check whether or not model-driven produces better results than other methods (e.g. code-centric method, method based on RUP and Agile methodology). The types of study used in these evaluations are mainly case studies, empirical evaluations and quantitative studies. As far as we know, there are no previous experiments that dealt with the comparison of a model-driven versus a code-centric method in the context of generating gesture-based interaction. So, this work is a step forward in the process of covering this gap.

**Table 8. Summary of works related with Evaluation between model-driven paradigm and other methodologies**

Author	Year	Type of study	Goal	Application field	Variables	Experimental subjects	Tool employed in the experiment
<b>Kapteijns et al.</b> [104]	2009	Case study	To compare an MDD implementation with regular third generation programming	A small middleware application	Development productivity, development time, application complexity.	Novice and expert developers	XuWare
<b>Bunse et al.</b> [105]	2009	Case study	To compare MARMOT (based on MDD+CBD) with RUP and Agile Development.	A small control system for an exterior car mirror.	Development effort, model size, defect density, amount of reused elements, model size.	Graduate students	MARMOT
<b>Ricca et al.</b> [106]	2012	Controlled experiment	To compare UniMod programming with code-centric programming	Two new versions of Svetofor and Telepay: Svetofor+ and Telepay+.	Effectiveness of UniMod	Bachelor degree students	UniMod versions of Svetofor and Telepay
<b>Martinez et al.</b> [108]	2012	Quasi-experiment	To compare the productivity and satisfaction of	Web 2.0 application	Productivity and satisfaction	Master's degree students	A set of tools to implement the



<b>Condori-Fernandez et al. [107]</b>	2013	Empirical approach	Usability of model-driven tools	Not mentioned	Efficiency, Effectiveness and satisfaction	PROS-UPV Researchers	INTEGRANOVA
<b>Martinez et al. [110]</b>	2014	Empirical study	Maintainability	Web application	Performance and satisfaction	Graduate students	OOH4RIA
<b>Cervera et al. [111]</b>	2015	Empirical evaluation	To evaluate usefulness and ease of use of MOSKitt4ME	Not mentioned	Perceived usefulness, ease of use	Master's and PhD students, a post-doc, industrial software engineers	MOSKitt4ME
<b>Panach et al. [112]</b>	2015	Experiment	To verify some of the most cited benefits of MDD.	Web application	Quality, effort, productivity and satisfaction.	Final-year Master's degree students	INTEGRANOVA

We will use an empirical evaluation to compare a model-driven method and code-centric method in order to evaluate performance and acceptance of our proposal. In this experiment, the participants will define custom gestures and they will include gesture-based interaction in an existing user interface with source code written in Java.

### **3.7 Technical action research to validate software systems**

In a similar way than the previous section, we include this section in order to know how was applied the technical action research in the industrial context to validate software products.

In this sense, some works related with the applicability of technical action research in the field of software engineering are included in this section. However, we found few reports about the application of TAR to validate software systems:

Morales-Trujillo et al. [113] describe the validation of a software engineering framework employing Technical-Action Research and case study methods. They report that the combination of TAR and case studies was a successful experience and that it is a feasible resource for bridging the gap between academy and industry.

Morali et al. [114] report the use of TAR to validate a method to specify confidentially requirements in an outsourcing relation. They used CRAC++ to specify confidentially requirements that could be included in an outsourcing SLA.

Abelein [115] describes in his work the application of technical action research to validate iPeople Case Study applying the User-Developer Communication-Large-Scale IT Projects (UDC-LSI) method. His evaluation showed a positive effect of the UDC-LSI method on effectiveness and efficiency.

Antinyan et al. [116] report a complementary empirical method for validating software measures. The method is based on action research principles and it can be combined with theoretical validation methods.

The industrial experiences reported in their work show that in many practical cases the method is effective.

In this work, we will employ technical action research in order to validate our proposal in an industrial context to determine its benefits in the field of human-computer interaction related with gesture-based interaction.

### 3.8 Range of Improvements

At the end of the revision of the related literature, we can describe some problems found in the context of this thesis related with the research areas identified in the beginning of this chapter:

#### *Research area 1: Gesture representation*

Our search on the related literature about gesture representation has demonstrated that there are a variety of techniques to define gestures that are platform-specific. This fact can represent a problem when the software engineers have planned to develop software to several platforms of devices because the engineers require have a good level of skills and experience to develop software with quality and efficiency.

#### *Research area 2: Definition of a model-driven method to generate user interfaces with gesture-based interaction*

Currently, computing devices and their software systems are in great demand. This creates a problem that needs attention: it is necessary updated versions of software systems for the most number of platforms as well as development methodologies of software systems to allow the implementation of these systems in the shortest possible time, at the least cost and with quality.

Another aspect that should be considered is the widespread use of gestures to perform actions on the device, the application fields of software systems with gesture-based interaction is growing and

software developers build software systems that do not allow easy addition of new gestures or user-defined (custom) gestures.

The development of software systems for devices with gesture-based interaction is largely based on the source code, forcing developers to achieve mastery of a platform and its tools and learn to deal with all the pros and cons that brings this trend, according to mentioned in Chapter 2 of this research work.

*Research area 3: the type of method employed to perform the verification of the proposal*

In this case, we analyse the works related with comparative evaluation of two methods to develop user interfaces. Some works had used empirical evaluation with the aim of validating the usability of a solution to obtain software systems, but we do not find any work related with the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces.

### 3.9 Summary

This chapter presents the state of the art of the disciplines that are related to this thesis.

In the field of software systems development methodologies in this thesis have described the model-driven development which has been considerate to use in this thesis because it satisfies the requirements specified in this document.

In the field of gesture-based interaction there is a series of works with proposed techniques for gestural representation, a work that is closest to the proposal of this thesis is presented in [74], which is based on Petri networks, however work focuses on gesture recognition rather than the representation of a gesture based on MDA, such as the proposal for this thesis.

We will try to solve these problems with our work that is described in this document.





# gestUI: A MODEL-DRIVEN METHOD

# 4

The topics covered in this chapter are:

- 4.1 Overview
- 4.2 Why a Model-driven method?
- 4.3 Why a Model-View-Controller design pattern?
- 4.4 Determining needed resources
- 4.5 gestUI: our proposal
- 4.6 Personalization of gesture definition
- 4.7 Overview of gestUI to include gesture-based interaction in a user interface
- 4.8 Summary



## Chapter 4. gestUI: A Model-Driven Method

### 4.1 Overview

Currently, there are two topics to consider regarding the development of user interfaces supporting gesture-based interaction:

- (i) The current tendency to adopt new human-computer interaction techniques considering the development of the technology, specifically the gesture-based interaction. In the current technology market, there is a wide range of devices platforms supporting gesture-based interaction.
- (ii) The high demand of gesture-based interaction in the software systems. There is a variety of SDK's to develop software systems for these devices with gesture-based interaction.

However, when a software engineer employs a code-centric method to include gesture-based interaction in user interfaces of software systems, some of the following problems are involved [117] [118] [119]: (i) the software engineer has two options to obtain the source code: writing the methods required to implement the software from scratch or adapting existing source code; (ii) the gesture specification is not multi-platform; (iii) it is hard to reuse the source code to support gesture-based interaction in other platforms; (iv) software engineers require skills in the programming language of each platform employed in the implementation of software systems user interfaces; (v) in some cases, the integrated development environment (IDE) is not available in all platforms required by users.

These facts allow the introduction of new challenges and opportunities in the design and development of software systems with gesture-based interaction for any platform as well as the ability to define custom gestures in the design stage of a software system.

Considering this situation, we motivated to define a methodology that allows improving processes to include gesture-based interaction in user interfaces of software systems for any available platform in the

market, and a mechanism to simplify the definition of custom gestures according to the needs of end users without requiring extensive skills or knowledge of a programming language.

This chapter introduces a new model-driven method, called gestUI that aims to overcome the limitations identified and described in this thesis to implement user interfaces with gesture-based interaction. Thus, to meet this challenge, our proposal advocates the use of Model-Driven Engineering (MDE) techniques. We believe that the use of MDE reduces the complexity of the implementation of gesture-based user interfaces because it allows software engineers to work at a high level of abstraction and it also increases automation and reuse.

In our work, the level of abstraction is raised by allowing software engineers to design gestures as models. On the other hand, automation is increased by means of model transformations.

The objectives of this approach are:

- (a) To provide a methodology to include gesture-based interaction in software systems user interfaces, which is platform-independent of computing devices.
- (b) To provide a mechanism that helps the developer to specify a catalogue of gestures that can be included in a software system user interfaces for devices with gesture-based interaction.

We apply gestUI in user interfaces of existing software system with the aim of modifying the component related with the interaction to add the possibility of using gestures to perform actions. Under this situation, we consider the Model-View-Controller (MVC) design pattern because it describes the logical components of a user interface in an independent way of any technology.

In summary, the method proposed is based on Model-driven paradigm and on the Model-View-Controller design pattern with the aim of generating gesture-based user interfaces. Therefore, the reasons about their use is given in the following sections: Section 4.2 gives an

answer about the use of Model-Driven paradigm in this thesis. Section 4.3 explains the use of Model-View-Controller design pattern in this thesis. Additionally, Section 4.4 presents an explanation about the needed resources to implement the method.

The subsequent sections of this chapter are organized as follows: Section 4.5 introduces the method proposed called gestUI. Section 4.6 describes the personalization of the gestures. Section 4.7 includes the summary of this chapter.

## 4.2 Why a Model-Driven method?

The major challenges that Software Engineering has to deal with are represented by two questions: how sustainability can increase productivity? And how you can shorten the period of time to have new software or new versions of existing software? [34].

Unfortunately these questions do not have answers based on the traditional methodologies of software development, they employ a paradigm focused on the code and third generation languages, and its role is defined to be in the field of the solution instead of in the field of problem [34] [36]. This implies that software systems developers require more effort in the process of implementing software systems [94], consequently affecting their work performance, and time to have new versions of software systems.

MDD is a software development approach that has the potential to deal with the identified challenges of Software Engineering mentioned above. MDD proposes the use of models to specify the desired details of a system (requirements) and using transformations rules to generate the source code automatically to the hardware platform specified in the process [120]. MDD offers an environment that ensures the use of models throughout the development process of software systems [34]. Another feature of MDD is abstraction, a fundamental feature for describing components of software system development without considering the target technology platform [121]. In our thesis, abstraction is an important feature since the

proposed framework is platform independent of hardware and/or software of the devices.

In summary, in this thesis, techniques related with MDD are applied. This help to raise the level of abstraction of the process, which enhance the development process of software systems with gesture-based interaction, and facilitate the definition of gesture-based interaction in software systems. By applying these techniques, we get improved productivity by managing the similarities and differences of the devices with gesture-based interaction, promoting reuse and automation in the software development process.

### 4.3 Why a Model-View-Controller design pattern?

The software systems must work on various types of devices, including devices installed in different environments and devices that users carry with them [122]. The aim is to improve the usability of a software system [123] because they must provide convenient access to the services offered, and allow users to learn the functionalities of the application and to produce results in a short time. Software systems must be adapted to interact with the user in an appropriate way based on the context where the user is located.

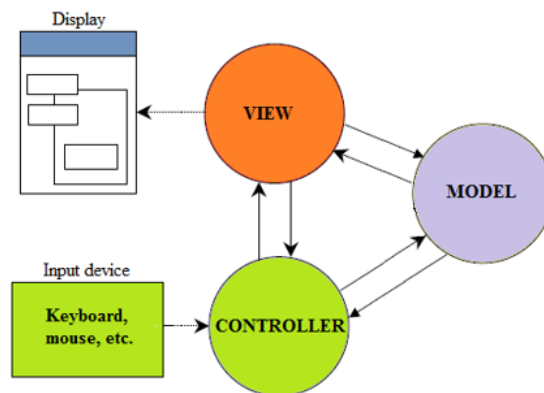
In the specification of the architecture of these software systems, the functional core should remain independent of the user interface. This core, based on the functional requirements of the system, usually remains stable. The user interface may need some change or adaptation. For instance, (i) the software systems may have to support different user interface standards, or be configured to suit the client's business process. This leads to the need of using architectures that support the adaptation of user interface components without causing significant impact on software system-specific functionality or data model used [124]; (ii) the software systems that are based on traditional interaction (using keyboard and mouse) could be migrated to use a gesture-based interaction but maintaining the same

functionalities, that is, now the user employs a gesture to do an action that before required using a keyboard or mouse.

The mentioned possibilities can be realized using a MVC design pattern in two instances of the software system lifecycle:

- a) When the software system is in the design stage. In this case, the software engineers include features to specify the type of interaction to use in the user interface.
- b) When the software system is implemented. In this case, software engineer can modify existing features in the user interface in order to include other type of interaction in the user interface. We consider this situation in this thesis.

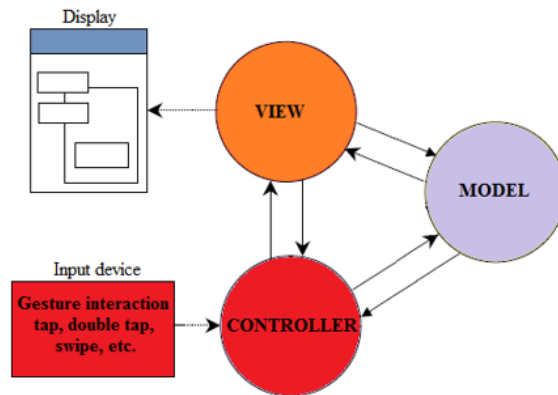
Figure 11 shows the software system based on the MVC design pattern where the controller of the software system receives signals done with traditional interaction using keyboard and mouse to perform actions.



**Figure 11. Software System with traditional interaction**

Figure 12 is shown the same software system based on the MVC design pattern but with other controller that supports gesture-based interaction where the user can employ gestures to perform actions on the system (gesture-based interaction). In this latter case, the “new” controller is prepared to receive actions by means of gestures instead of actions by means of keyboard and mouse.





**Figure 12. Modifying the controller to support gesture-based interaction**

In this thesis, the MVC design pattern is used because it allows to treat the logical components in an independent way of the technology components in a device [123]; furthermore, from the logical point of view, knowing the hardware platform which will operate the software system resulting in the code generation process is not considered essential. The independence in the components defined by this design pattern is also complemented with platform independence achieved using MDE in the process of definition of the methodological framework.

#### 4.4 Determining needed resources

In order to develop the method for including gesture-based interaction in software systems for any device, it is necessary to consider some questions that help us in the construction of the solution:

- What kind of gesture can be specified?
- What kind of gesture-based interaction will be supported?
- What kind of actions might be performed through gesture-based interaction?

We consider that the following resources are required to obtain an answer of each of these questions:

- An **artefact to define the type of gestures** that the user can perform independently of target devices;

- The specification of the **supported interaction type** independently of target devices;
- A mechanism to specify the **actions that can be performed** through gesture-based interaction.

A brief explanation about these resources is included in the following paragraphs:

- Regarding the *type of gestures* to use in the process, we consider touch gestures because currently in the market there are devices that support these types of gestures. The touch gestures are used in devices with touch screen such as smart phones, tablets, computers, and so on. It is needed to define the features that characterize a gesture to include them in the artefact that allows the gesture definition. Using the model-driven paradigm is possible to specify some tasks to perform in the software development life cycle: (i) to specify types of gestures, and even to establish patterns of gestures so that information regarding gestures is available for inclusion in software development processes considering gesture-based interaction; (ii) to define the relation between gesture and the command (or action) included in the user interface.
- The task related with the specification of **supported interaction type** involves two aspects: (i) the definition of the device because the type of interaction is directly related to the type of device and (ii) the context of use because this feature defines how will be specified the actions in a device. For instance, if the user has a device with touch screen, he/she uses a touch gesture implying touch gesture-based interaction. If the user has the hands and eyes occupied with other tasks such as driving, then the speech interaction is appropriate.
- Finally, regarding the **action that can be performed** with touch gesture-based interaction, it is related with the definition of the gesture-action correspondence. The actions are specified by means of commands assigned to the widgets (image, text field,

button, etc.) included in a user interface. The type of actions depends of the type of software system, if we consider a form-based software system, the actions are related with typical operations of a database (e.g. CRUD<sup>9</sup> operations), but if we consider a CASE tool to draw diagrams, the actions are related with sketching primitives of a diagram (e.g., using a finger to sketch a rectangle and to obtain a class in a UML diagram). Therefore, the gesture-action correspondence consists in the specification of a gesture to perform any action when a user touches some widget in the user interface. For example, if the item is an image then the actions can be: reducing the size (zoom in), increasing the size (zoom out), rotating right, rotating left, sending by e-mail, etc. If the element is a button, it will only be possible to press the button to execute some action specified in this widget (e.g. to save a record in a database). Additionally, users can sketch custom gestures on a touch surface to execute some action, e.g., a user can sketch a gesture to print a document instead of select the corresponding option in a menu.

In this thesis, the specification of these resources is performed using metamodels and models that are employed as part of the proposed methodology using MDE. A detailed description of the mentioned resources (metamodels and transformations) in the scope of aforementioned methodologies can be found in the following pages.

#### 4.5 gestUI: our proposal

This chapter introduces a new model-driven method, called gestUI that aims to overcome the limitations that are identified in Chapter 1, in the implementation of user interfaces with gesture-based interaction.

In order to obtain a better understanding about our proposal, this section describes the following topics related with gestUI: (i) features of gestUI, (ii) metamodel to represent a gesture catalogue modelling

---

<sup>9</sup> CRUD operations are referred to Create, Read, Update and Delete.

language including the description of each class of the metamodel with its business rules<sup>10</sup>, (iii) components of gestUI and, (iv) model transformations defined to obtain gesture-based user interface.

#### **4.5.1 Features of gestUI**

gestUI [125] is defined with the aim of helping in the definition of custom gestures and in the inclusion of gesture-based interaction in user interfaces.

gestUI is model-driven since its main artefacts are conceptual models which are compliant with the Model-Driven Architecture, a generic framework of modelling layers that ranges from abstract specifications to the software code. gestUI is composed of three layers according to the model-driven method: a platform-independent layer, a platform-specific layer and source code.

gestUI is user-driven because the users are the main actors in the definition of custom gestures and in the inclusion of gesture-based interaction in information systems user interfaces.

gestUI is iterative because if the users are not satisfied with the definition of the gestures or maybe they have problems sketching some gesture, then they can repeat the process of gesture definition to redefine such a gesture.

Through MDD, gestUI aims to tackle the problems indicated in Section 4.1, as Table 9 shows.

The scenario where gestUI can be included to implement gesture-based user interfaces consists of the following steps:

Step 1: Stakeholders (e.g. software engineers, end-users, software analysts) obtain the requirements specification of the software system

---

<sup>10</sup> According to Kardasis et al. [163], business rules have been defined as 'declarations of policy or conditions that must be satisfied' in a software system or in an organization.

containing human-computer interaction (gesture-based interaction) to include in the user interface.

Step 2: The software system, including the user interfaces, is designed.

Step 3: The developers implement the software system, including user interfaces.

Step 4: By using gestUI, the stakeholders include the gesture-based interaction in the user interface and finally, by applying model transformations they obtain the gesture-based user interface source code. gestUI modifies the source code of the user interfaces in order to include custom gesture definitions and gesture-based interaction according to the requirements specified by the end-users. As a result of this, a new version of the existing user interface is automatically generated by using model transformations.

Step 5: By executing the software system, the user can perform actions using this recently implemented user interface with gesture-based interaction.

**Table 9 Detected problems vs. Benefits of model-driven paradigm to solve them**

Problems described in Section 4.1	Benefit of model-driven paradigm [29]
Problem (iv)	Productivity
Problems (ii)	Portability
Problem (v)	Interoperability
Problem (iii)	Reusability
Problem (i)	Source code automatic generation

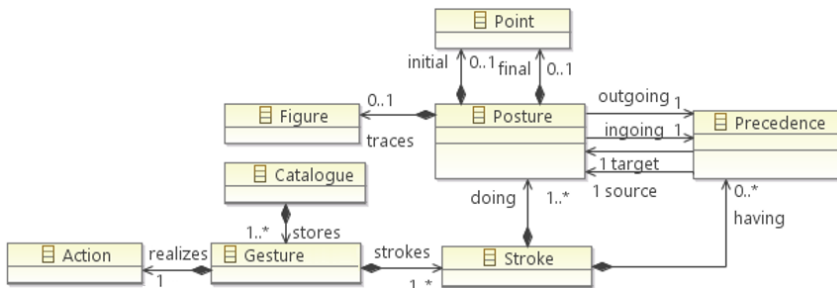
Then, considering this sequence of steps, gestUI permits also the inclusion of gesture-based interaction in legacy systems with user interfaces supporting traditional interaction using keyboard and mouse. After the application of gestUI, such user interfaces will support the gesture-based interaction.

#### **4.5.2 Metamodel of the gesture catalogue modelling language**

We consider that a user interface of a software system supporting gesture-based interaction requires a gesture catalogue containing

custom gestures to execute actions (commands) specified in the user interface. This type of user interface is called gesture-based user interface.

In this thesis, we define a metamodel (Figure 13) to specify the gesture catalogue modelling language. The description of the proposed metamodel consists of the classes that define the gesture catalogue, the business rules and constraints defining custom gestures. These business rules will be applied in the definition of the gesture catalogue using validation rules by means of OCL (Object Constraint Language) sentences, which will be included in the stage of metamodel definition.



**Figure 13. Metamodel of the gesture catalogue modelling language**

A description of each class is included in the following paragraphs:

**Catalogue:** It represents a gesture catalogue which contains all the gestures defined by the user, and that are available in a system or a device. It has an attribute that describes the name of the gesture catalogue.

**Table 10. Business rules for the "Catalogue" class**

Class	Business rule	OCL constraint
Catalogue	The name of the catalogue must be unique	context Catalogue inv: self.contains -> isUnique(Name)
	A gesture catalogue has at least one gesture	context Catalogue inv: self.stores -> size >0;

The business rules associated to this class are “*The catalogue name must be unique*” and “*A catalogue contains at least one gesture*”. These business rules are validated using OCL sentences as is shown in Table 10.

**Gesture:** It represents the gestures that conforms a gesture catalogue. It has some attributes to describe a gesture: (i) the name of a gesture, (ii) if the gesture is discrete or continuous, (iii) the pressure applied on a touch surface when a gesture is sketched and, (iv) the duration time by sketching a gesture. The business rules associated to this class are: *“The gesture name must be unique”*, *“A gesture can be discrete or continuous”*, *“A multi-stroke gesture is formed by a sequence of postures”*, *“A touch-based gesture applies a pressure on the screen”* and, *“A gesture has duration time > 0”*. These business rules are validated using OCL sentences as is shown in Table 11.

**Table 11. Business rules of the "Gesture" class**

Class	Business rule	OCL constraint
Catalogue	The names of the gestures defined in the catalogue must be unique	context Catalogue inv: self.stores -> forall(g   g.isUnique(Name));
	A gesture can be discrete or continuous	context Catalogue inv: self.stores -> select(g   g.typeGesture=#discrete or g.typeGesture=#continuous)
Gesture	A multi-stroke gesture is formed by a sequence of postures	context Gesture inv: self.strokes>1 implies self.strokes.doing -> size>1
Gesture	<i>A touch-based gesture applies a pressure on the screen</i>	Context Gesture Inv: self. Pressure>0
Gesture	<i>A gesture has duration time &gt; 0</i>	Context Gesture Inv: self. duration Time>0

In this case, we define the enumeration *“GestureType”* containing the values *“Discrete”* and *“Continuous”*. It permits to specify the type of gesture according to the definition included in Chapter 2.

**Action:** It defines the action (command) to execute when the gesture is sketched by the user and recognised by means of a gesture recogniser algorithm. The business rule associated to this class is *“A gesture performs one or more actions”*. This business rule is validated using an OCL sentence as is shown in Table 12:

**Table 12. Business rule of the "Action" class**

Class	Business rule	OCL constraint
Gesture	A gesture performs one or more actions	context Gesture inv: self.realizes -> size>0

Stroke: It corresponds to the marks made with a mouse, pen or finger to define a gesture or a part of it. According to the number of strokes, a gesture can be single-stroke or multi-stroke. The business rule associated to this class is *"A gesture contains at least one stroke"*. This business rule is validated using an OCL sentence as is shown in Table 13:

**Table 13. Business rule of the "Stroke" class**

Class	Business rule	OCL constraint
Gesture	A gesture contains at least one stroke	context Gesture inv: self.strokes -> size>=1

Posture: It corresponds to the description of each posture that conforms a gesture. The attributes are: name of the posture which is used to identify it; the state of execution of the posture (initial, executing, final). The business rules associated to this class are *"A stroke contains at least one posture"* and *"The state of execution of the posture can be initial, executing or final"*. This business rules are validated using OCL sentences as is shown in Table 14.

In this case, we define the enumeration *"State"* that contains the values *"Initial"*, *"Executing"*, and *"Final"*. It is used to specify the cycle of execution of the postures of a gesture.

Precedence: It specifies the precedence relation between postures. It has an attribute (name) to identify the precedence defined between postures.

The business rule associated to this class is *"If a stroke has two or more postures then it is required to define a precedence relation"*. This business rule is validated using an OCL sentence as is shown in Table 15.



**Table 14. Business rules for the "Posture" class**

Class	Business rule	OCL constraint
Stroke	A stroke contains at least one posture	context Stroke inv: self.doing -> size>=1
Posture	The state of execution of the posture can be initial, executing or final	context Posture inv: self.state -> select(g   g.state=#initial or g.typeGesture=#executing or g.typeGesture=#final)

**Table 15. Business rule for the "Precedence" class**

Class	Business rule	OCL constraint
Stroke	If a stroke has two or more postures then it is required to define a precedence relation	context Gesture inv: self.strokes -> size>=2 implies self.stroke.having -> size > 0

Figure: It defines the type of figure that can be drawn using the points of a posture which is part of a gesture. In this case, we define two additional enumerations which contains definition of values of some attributes that complete the specification of a gesture: (i) Enumeration called "*Orientation*" containing the orientation (up, down, left, and right) in which a figure is drawn using the points defined in a posture; (ii) Enumeration "*FigureType*" that defines the type of figure (e.g. line, circle) that can be drawn between two points in order to obtain the drawing of a posture, and finally, the drawing of a gesture.

Point: It is related with the postures that conform a gesture. It contains the definition of coordinates (X, Y) to trace a posture and locate a gesture in a touch screen. The coordinates must take valid values in a touch screen depending on its size and its resolution. The business rule associated to this class is "*In a touch-based gesture, the values of coordinates (X, Y) must be greater than zero*". This business rule is validated using an OCL sentence as is shown in Table 16:

**Table 16. Business rule for the "Point" class**

Class	Business rule	OCL constraint
Posture	In a touch-based gesture, the values of coordinates (X, Y) must be greater than zero	context Posture inv: self. Initial.X >0; self. Initial.Y >0; self.final.X>0; self.final.Y>0

The data structure that defines a gesture is described in Table 17.

Therefore, according to the proposed metamodel (its classes, the business rules and the constraints specified with OCL sentences), we consider that:

A gesture catalogue (Catalogue class) contains one or more gestures (Gesture class) to execute actions (commands) (Action class) of a software system. Each gesture is formed by one or more strokes (Stroke class) defining single or multi-stroke gestures.

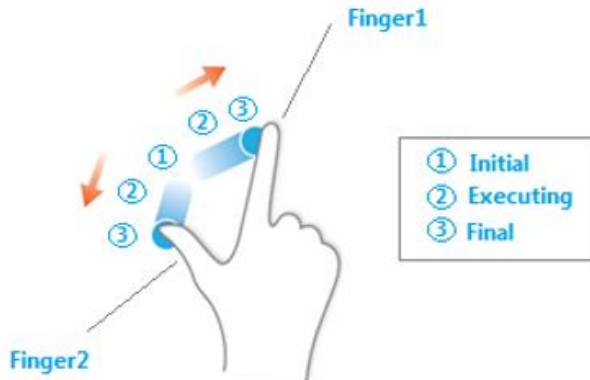
Each stroke is formed by postures (Posture class). A posture can be sequentially divided into three states (Figure 14): initial, executing, and final (State enumeration):

- The first state (Initial) occurs when the user begins sketching the gesture, for example, in a touch-based system the user puts a finger on the screen and the system detects it.
- The second state (Executing) can be formed by a set of postures depending of the type of gesture that is being sketched.

Class	Attribute	Description	Type	Observation
Catalogue	Name	Name of the catalogue gesture	String	
	Name	Name of the gesture.	String	
	Type	Specifies the type of a gesture.	Enum	Discrete or continuous.
Action	Name	Name of the action to execute	String	
Posture	Name	Name of the posture.	String	
	State	It specifies the state of execution of the posture.	Enum	Initial, Executing, Final.
Precedence	Name	Name of the precedence relation defined between postures in a gesture.	String	
Figure	Type	Type of figure that can be drawn between two points.	Enum	Line, Circle.
	Orientation	The attribute specifies the orientation of a figure drawn in a posture.	Enum	Up, Down, Left and Right.
Point	CoordX	It specifies the value of the coordinate X where a touch is detected.	Float	X>0
	CoordY	It specifies the value of the coordinate Y where a touch is detected.	Float	Y>0

**Table 17. Data structure of a gesture**

- The third state (Final) occurs when the user finishes the sketch of the gesture, for example, in a touch-based system occurs when the finger of the user doesn't touch the screen.



**Figure 14. States of a posture**

If a gesture consists of two or more postures is necessary to specify the order of execution of them. In this case, there are two possibilities to consider:

- Using a sequential number to specify the order of the postures that conform a gesture, or
- Using the concept of precedence relation to specify the order and relation of the postures that conform a gesture.

In this thesis, we consider the concept of precedence relation (Precedence class) because is more adequate to specify successor and predecessor in a set of postures rather than assigning a number of sequence to the postures. The precedence relation defines the order of execution of a set of postures that conforms a gesture, therefore, considering the concept of precedence relation between two postures  $P_A$  and  $P_B$ , posture  $P_A$  must necessarily occur before posture  $P_B$  occurs (see Figure 15).

Consequently, the sequence of strokes in the gesture is specified by means of precedence order.

Additionally, the concept of precedence relation considers the definition of source and target postures to specify the order and relation of the postures that conforms a gesture. Also, in this concept the definition of two states (ingoing and outgoing) is considered in order to specify the input and output precedence in the execution of a posture (see Figure 15).

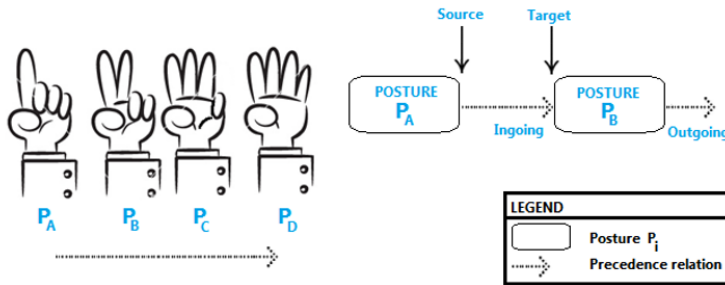


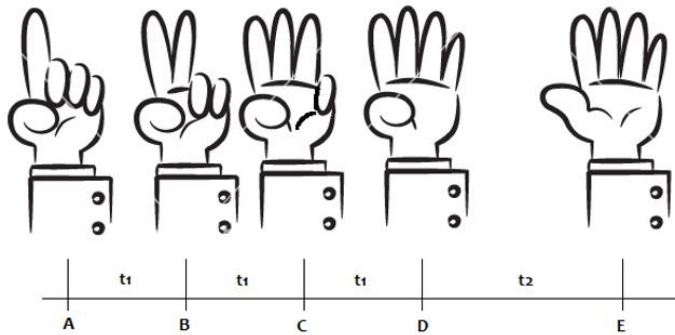
Figure 15. Precedence relation between postures

Each posture is composed of two points, initial and final, and each point (Point class) is described by coordinates (X, Y). Each posture draws a figure (e.g. line, circle) (Figure class) with an orientation (up, down, left, right) (Orientation enumeration). The set of postures (points) conforms the gesture.

If a gesture has two or more postures, it is necessary to define the precedence relations between postures in order to specify their order and relation. In this definition, *source* and *target* postures must be specified and, *ingoing* and *outgoing* precedence must be specified too.

During the trace of a multi-stroke gesture, it is necessary to consider the time interval between strokes that are being executed. For instance, if a gesture is formed by two strokes, then the time interval between strokes must be specified in order to recognize that both strokes belong to the same gesture.

In the process of definition of a gesture the elapsed time between postures must be similar<sup>11</sup> because the duration of one of them can define a gesture different to the gesture that the user wants to define. In Figure 16, a set of postures (A-B-C-D-E) that outline a gesture is shown, each posture has a duration time  $t_1$ , but the next posture (E) has a duration time  $t_2$  ( $t_1 \neq t_2$ ). In this example, we have two gestures instead of one gesture: first gesture (postures: A-B-C-D), and the second gesture (sequence E). If  $t_1 = t_2$ , would have only a defined gesture (postures: A-B-C-D-E). For instance, the action of pressing the touch screen during a short time defines the tap gesture, but if the time is greater the gesture can be traduced as “tap and hold”.



**Figure 16. Interval of time between postures**

Some of the constraints defined using OCL in the metamodel are included in Table 18.

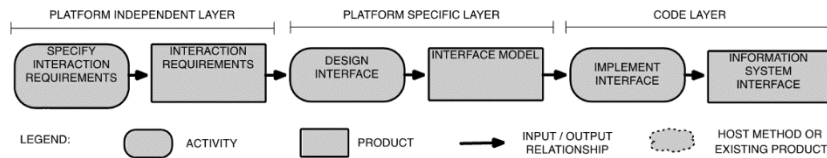
<sup>11</sup> When the user sketches a multi-stroke gesture, which is represented by a sequence of strokes, in the specification of the gesture, the user needs to specify a value of time between strokes so that it can determine that the trace of the gesture is complete, i.e., the execution of the gesture is finished.

**Table 18. Constraints and business rules of gesture definition**

Class	Description	OCL constraint
Catalogue	The name of the catalogue must be unique	context Catalogue inv: self.contains -> isUnique(Name)
Catalogue	The names of the gestures defined in the catalogue must be unique	context Catalogue inv: self.stores -> forAll(g g.isUnique(Name));
Catalogue	A gesture catalogue has at least one gesture	context Catalogue inv: self.stores -> size > 0;
Gesture	A gesture has at least one stroke	context Gesture inv: self.strokes -> size > 0;
Gesture	A gesture performs an action	context Gesture inv: self.realizes -> size = 0;
Stroke	A stroke has at least one posture	context Stroke inv: self.doing -> size > 0;
Stroke	If a stroke has two or more postures then it is required to define a precedence relation.	inv: self.strokes -> size >= 2 implies self.stroke.having -> size > 0
Posture	It is required at least two points to define a gesture	context Posture inv: self.traces -> size >= 2;
Point	The values assigned to the attributes CoordX and CoordY must be greater than zero.	context Point inv: self.CoordX > 0 and self.CoordY > 0;

### 4.5.3 Components of gestUI

Figure 17 shows an excerpt of the typical structure of an existing method based on model-driven paradigm to develop user interfaces. As is shown in Figure 17, a method defined by means of model-driven paradigm has three layers: platform-independent layer, platform-specific layer and source code layer. From the platform independent model (PIM) using successive transformations we obtain the platform-specific model (PSM) and then the source code is obtained. This source code of the user interface includes traditional interaction.



**Figure 17. A general excerpt of any method for develop user interfaces**

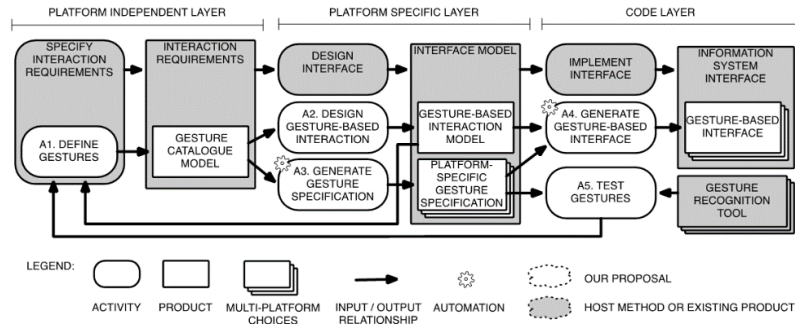
In general, the activities and products included in the layers of any existing MDD method are:

- In the platform-independent layer, the interaction requirements are detailed (*Activity “Specify Interaction Requirements”*) obtaining as a result the requirements specification to develop a user interface (*Product “Interaction Requirements”*).
- In the platform-specific layer, the interaction requirements are the input to perform the interface design (*Activity “Interface Design”*). As a result, in this layer the interface model is obtained (*Product “Interface Model”*).
- In the code layer, the interface model is the input to implement the interface (*Activity “Implement Interface”*) and to obtain the information system interface. As a result, in this layer the source code of the user interface is obtained (*Product “Information System Interface”*).

In order to obtain gesture-based interfaces, we propose including gestUI in the layers of any MDD method. Figure 18 shows the resultant



method consisting of three layers that contain existing activities and products, represented in grey colour and, activities and products contained in gestUI, represented in white colour.

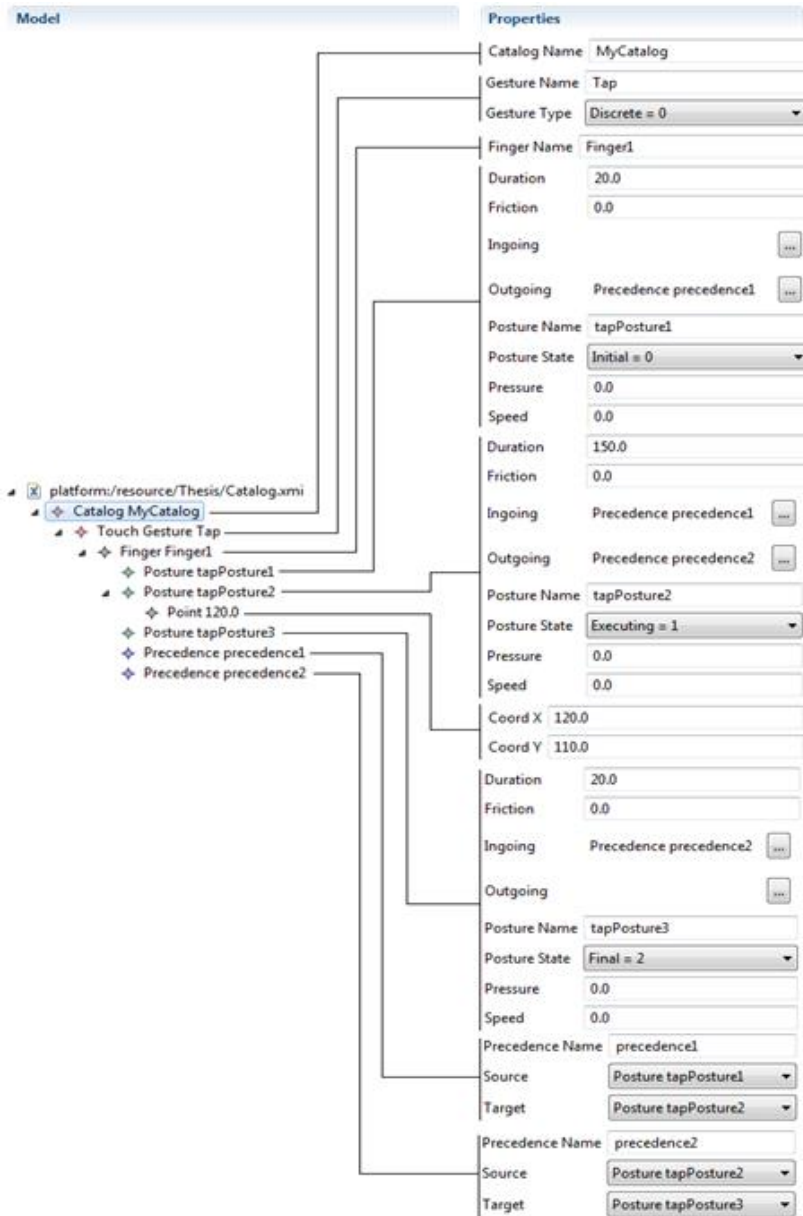


**Figure 18. gestUI method overview (Taken from [31])**

Each one of the modules contained in gestUI are described in the following paragraphs according to Figure 18:

### Platform-independent layer

1. Activity A1 (“Define gestures”) in which the developer specifies the gestures in collaboration with representative information system users. In our proposal, the gestures are defined by sketching on a canvas, then they are stored in the ‘Gesture catalogue model’ which conforms to the metamodel described in Section 4.5.2. Each gesture is formed by one or more strokes defined by postures, which in turn are described by means of coordinates (X, Y). The sequence of strokes of the gesture is specified by means of precedence. Each posture in a gesture is related to a figure (line, rectangle, circle, etc.) with an orientation (up, down, left, and right) and a state (initial, executing, final) qualifying the order of the strokes. The gesture catalogue definition could be part of a larger ‘Interaction requirements’ specification. The product obtained in this activity is the gesture-catalogue model. Figure 19 shows an example of platform-independent gesture definition using the gesture model.



**Figure 19. Platform-independent gesture definition**  
 In this figure, it is possible to check the classes and their attributes included in the metamodel described in this chapter.

## Platform-specific layer

In this layer, the activities A2 and A3 permit that the gesture catalogue can be defined from a previously defined gesture repository. That is, the gestures can be reused in other user interfaces in the same software system or in other software system. The description of each of these activities is as follows:

2. Activity A2, "*Generate gesture-based interaction*", since the user interface is designed in this layer, the gesture-based interaction is also defined in this layer in collaboration with the user by means of a code-centric method. The filename of the user interface source code is inserted as attribute in the "Gesture" class in the gesture catalogue model with the aim of processing the source code to obtain the actions defined in the user interface. In a model-based software system user interface development, the actions are specified in the interface model. In a code-centric interface development they are implemented on the source code of the interface itself. The procedure mainly consists of applying a parsing process on the source code to obtain the components included in the user interface, after which the correspondence between the gesture and action/command included in the user interface is allocated. This correspondence allows a set of sentences (action/command) to be defined in the same programming language as the source code of the user interface and enable it to be executed by each gesture previously defined. The product obtained in this activity is stored in the "gesture-based interaction model".
3. Activity A3, "*Generate gesture specification*", consists in an M2M transformation using ATL as model transformation language. The gesture catalogue model is required as input data and the result is the platform-specific gesture specification. In this case, we consider the structure of the

gesture definition according to \$N gesture recognition tool as the target platform in the model transformation.

### Source code layer

This layer contains two activities:

4. Activity A4, “*Generate gesture-based interface*” where the gesture-based interaction model and the gesture catalogue model are transformed into an executable and deployable code of the user interface written in the selected programming language. The tool generates components (e.g., Java code) that are embedded in the existing information system interface. ‘*Gesture based interface*’ is automatically generated by the platform-specific layer artefacts.
5. Activity A5, “*Test gestures*”, in this activity the gesture catalogue model is transformed into language supported by the gesture recognition tool (i.e. XML) so that both the developer and the user can test the gestures using the gesture recognition tool (we currently support three gesture testing platforms: quill [89], \$N [50] and iGesture [48]). We apply M2T transformation with transformation rules written in Aceleo to generate the platform-specific gesture catalogue for each gesture recognition tool.

#### 4.5.4 Model transformations

gestUI is a model-driven method to define custom gestures and to include gesture-based interaction in user interfaces. Following a model-oriented paradigm is possible to obtain user interfaces with gestural interaction for any platform and other benefits related with this paradigm.

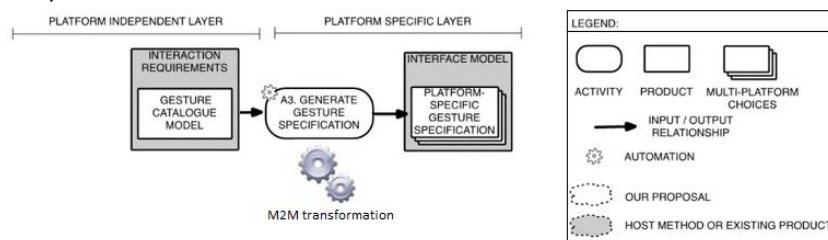
A model-driven method includes metamodels, models and model transformations [29]. The metamodel of the gesture catalogue is described in Section 4.5.2. In this section, with the aim of completing

the description of gestUI we describe the model transformations included in our work to obtain the gesture-based user interfaces.

In this thesis, we apply M2M and M2T model transformations in order to obtain a user interface including gesture-based interaction. With the aim of describing the model transformations we consider Figure 18 where the model transformations are represented by means of a symbol (a gear) in the upper left corner of the A3 and A4 activities of gestUI.

Firstly, a M2M transformation is performed during the activity A3 (Figure 18) to obtain the platform-specific gesture catalogue specification. This specification contains the gesture catalogue model according to the specification of the gestures to be used in the definition of the gesture-action correspondence to include gesture-based interaction in user interfaces. This specification is based on the definition of gestures included in the gesture recogniser algorithm considered in this thesis: \$N gesture recogniser. The model obtained in this model transformation conforms to the gesture catalogue metamodel described in Section 4.5.2.

Figure 20 shows this aforementioned M2M transformation that is executed by means of a transformation definition which contains the transformation rules written in ATL. Gesture catalogue model which conforms to gesture catalogue metamodel is the input to the M2M transformation. Platform-specific gesture specification (model) is the output in this transformation.



**Figure 20. An excerpt of Figure 18 showing the M2M transformation**

An excerpt of the transformation rules written in ATL is included in Figure 21 that contains the transformation rule to create the “Gesture”

class in the target model. In this transformation definition, the input is the gesture catalogue model and the output is platform-specific gesture specification.

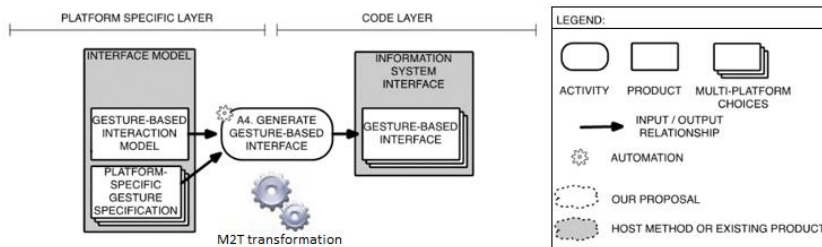
```
-- rule to create Gesture in multiStrokeGesture
rule Gesture {
  from
    s1: MMTG!Gesture(s1.employs->size()=1)
  to
    t1: MMMSG!Gesture (
      gestureName <- s1.gestureName,
      gestureType <- s1.gestureType,
      gestureDate <- s1.gestureDate,
      gestureTime <- s1.gestureTime,
      realizes <- thisModule.Action(s1.realizes),
      strokes <- s1.employs->collect(e|e.strokes
        ->collect(d|thisModule.Stroke(d)))
    )
}
```

**Figure 21. An excerpt for the M2M transformation**

Secondly, a M2T transformation is performed to obtain the source code of the gesture-based user interface (Figure 22). With the aim of supporting gesture-based interaction, this user interface source code contains the relation between gestures and actions where the gestures belong to the previously defined gesture catalogue and the actions are obtained from the same source code. This M2T transformation is included in the activity A4 of gestUI, described in Section 4.5.3. The filename containing the source code of the user interface and the name of the gesture catalogue are input data for this model transformation. In this case, the target platform is also specified by the user to generate the source code of the user interface.

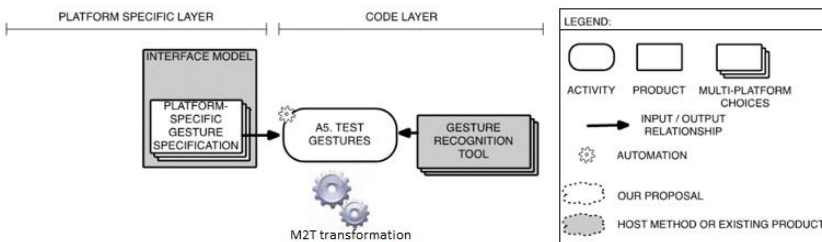
Additionally, using a second M2T transformation we obtain the gesture catalogue to be included in each of the three frameworks (gesture recognition tools) used in this thesis to test gestures (Figure 23): (i) quill [89] using GDT 2.0 to describe the gesture catalogue, (ii) iGesture [48] using XML to describe the gesture catalogue and (iii) \$N using XML to describe the gesture catalogue. In each transformation,

the specification of the target platform is required; in this case, each aforementioned framework.



**Figure 22.** An excerpt of Figure 18 showing the M2T transformation to obtain the gesture-based user interface

These M2T transformations are executed via a script containing the transformation rules written in Acceleo, applying a script that specifies information such as the classes and components participating in the generation, output folders, etc. The combination of the components that support the code generation process is depicted in Figure 24. The template definition, which drives code generation, constitutes the most important part of the transformation process. Appropriate templates have been defined for the platforms considered in our work: XML (\$N and iGesture), GDT (quill) and Java.



**Figure 23.** An excerpt of Figure 18 showing the M2T transformation to obtain the test gesture

Figure 24 includes an excerpt from the template written in Acceleo, for applying M2T transformation to obtain the gesture catalogue for the \$N gesture recognition tool. It also includes a header containing the general information of the gesture (gesture name, date and time when the gesture was sketched, number of strokes, number of points, etc.), the strokes contained in the gesture, and the set of points which conform the gesture.

In this thesis, the \$N gesture recognition algorithm [126] is adopted in order to apply it in the gesture recognize process. In this algorithm, the description of each gesture is stored in a file using XML, therefore, the transformation rules applied in the M2T model transformation consider the structure of the file containing each gesture in order to use it with the corresponding gesture recognize process.

```
[template public gestureM2T(aCatalog : Catalog)]
[comment @main/]
[for (g:TouchGesture|aCatalog.stores)]
[file (g.gestureName+'.xml', false, 'UTF-8')]
<Gesture Name = "[g.gestureName/]">
Subject = "test" Speed = "test" Milliseconds = "0" AppName = "NDollarRecognizer-java" AppVer =
"1.0" Date = "[g.gestureDate/]" TimeOfDay = "[g.gestureTime/]">
[for (f:Stroke|g.strokes)]
<Stroke index = "[f.strokeID/]">
[for (p:Posture|f.doing)]
<Point X = "[p.initial.CoordX/]" Y = "[p.initial.CoordY/]" T="0"/>
<Point X = "[p.final.CoordX/]" Y = "[p.final.CoordY/]" T="0"/>
[/for]
</Stroke>
[/for]
</Gesture>
[/file]
[/for]
[/template]
```

Figure 24. An excerpt for the M2T transformation

## 4.6 Personalization of gesture definition

### 4.6.1 Introduction

One of the main factors that could determine the success of gesture sets in user interfaces is whether the gestures can be effectively learned and remembered [45]. Personalization attempts to help the users to remember the gestures available in a user interface because the gesture is defined by the users themselves.

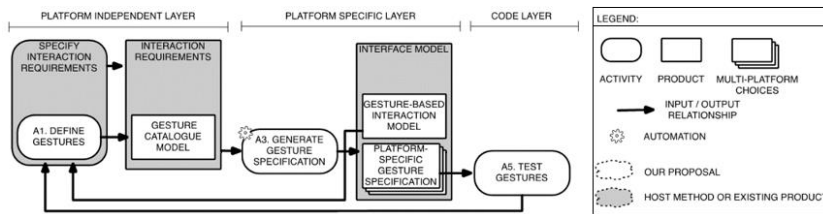
Personalization of gesture definition is related with a flexible gesture definition with no or minimal decrease of accuracy [127]. It is often desirable and necessary for users to create their own gestures, or personalized gestures [128].

gestUI is designed to support personalization of gestures by means of the definition of custom gestures, as described in Section 4.5. Additionally, if we consider this feature with the aim of redefining an



already defined gesture using gestUI then the user has two possibilities as is showed in Figure 25:

- (i) When the gesture-based interaction model containing the custom gesture definition is obtained.
- (ii) When the gestures are tested using the gesture recognition tools.



**Figure 25.** An excerpt of gestUI showing the redefinition of a gesture

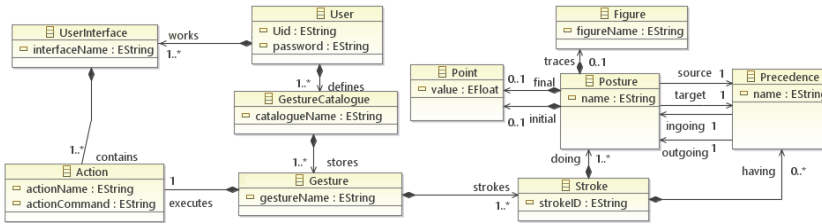
We describe how gestUI supports this feature of custom gesture redefinition by enhancing the metamodel described in Section 4.5.2.

In addition, the user has a third option to redefine custom gestures: when the system software containing user interfaces with gesture-based interaction is running. We have implemented a module that must be included in the software system with the aim of redefining custom gestures. This option is described in Section 4.7.

It is important to comment that by adding this feature in gestUI we give support to the user-centered design in the process of development of user interfaces including gesture-based interaction.

#### 4.6.2 Enhancing the metamodel

With the aim of implementing the personalization feature so that each user of gestUI can define/redefine custom gestures, we enhance the metamodel including two classes in the metamodel described in Section 4.5.2: *User* and *UserInterface* (Figure 26). These classes permit to complete the description of a user interface with gesture-based interaction.



**Figure 26. Enhanced version of the metamodel**

The description of each class and its business rules are described in the following paragraphs:

User: It represents a user of the user interface containing actions to execute by using gestures. It has an attribute that describes the user identification (UID) of the user. The business rules associated to this class are “*The user identification (UID) of the user must be unique*” and “*The user can define at least one gesture in a gesture catalogue*”. This business rules are validated using OCL sentences as is shown in Table 19.

**Table 19. Business rules for the "User" class**

Class	Business rule	OCL constraint
User	The user identification (UID) of the user must be unique	context UserInterface inv: self.contains -> isUnique(UID)
	A user can define at least one gesture in a gesture catalogue	context UserInterface inv: self.defines -> size >0;

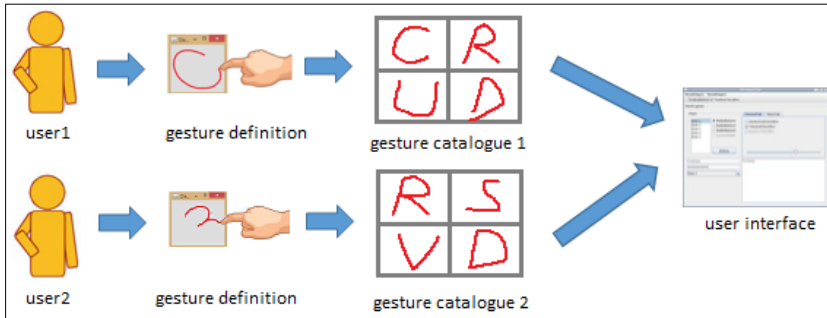
UserInterface: It represents a user interface which contains widgets (e.g. button, text field, canvas) containing actions to execute by the user, and that are available in a system or a device. It has an attribute that describes the name of the user interface.

The business rules associated to this class are “*The name of the user interface must be unique*”, “*A user interface is used at least by a user*” and “*The user interface contains at least one action to execute*”. This business rules are validated using OCL sentences as is shown in Table 20.

**Table 20. Business rules for the "UserInterface" class**

Class	Business rule	OCL constraint
UserInterface	The name of the user interface must be unique	context UserInterface inv: self.contains -> isUnique(Name)
	A user interface is used at least by a user	context User inv: self.works -> size>0
	A user interface has at least one action	context UserInterface inv: self.contains -> size >0;

The personalization feature is related with the enhanced version of the gestUI metamodel in order to include the user's definition, which permits individual users to define their own gestures catalogue to include gesture-based interaction in the user interface (Figure 27).



**Figure 27. Users defining their own gesture catalogue to apply it in the same user interface**

In this metamodel, the class *UserInterface* denotes the link to an existing user interface metamodel containing an element related with the action to execute using gesture-based interaction. Then, a user interface can be used by one or more users. Each user defines his own catalogue containing one or more gestures; each gesture permits to execute an action contained in the user interface. Each gesture is formed by one or more strokes defined by postures, and in turn described by means of coordinates (X, Y). The sequence of these strokes has an order of precedence. Each posture is related to a figure (e.g. line, circle) with an orientation (up, down, left, right), and is qualified by a state (initial, executing, final).

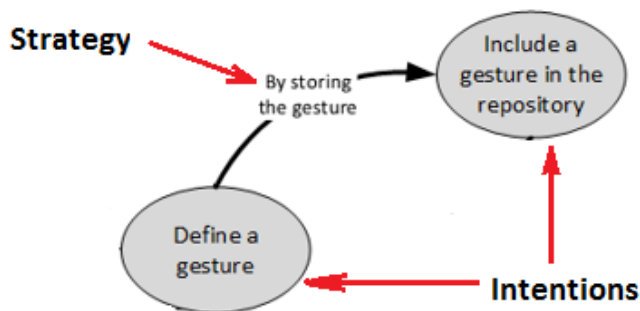
## 4.7 Overview of gestUI to include gesture-based interaction in a user interface

### 4.7.1 Introduction

In order to illustrate how to apply gestUI to include gesture-based interaction in a user interface we use MAP, a representation system which provides a non-deterministic ordering of intentions and strategies to model the multi-faceted purpose of a system [129].

An intention is a goal that can be achieved by performing a process [130]. For example, in the excerpt of the gestUI map shown in Figure 28 there are two intentions: “*Define a gesture*” and “*Include the gesture in a repository*”. Additionally, in a map there are two special intentions called ‘Start’ and ‘End’ to respectively start and end the process.

A strategy is an approach, a manner to achieve an intention [130]. In the same Figure 28 there is one strategy called “By storing the gesture”, defining a transition from “Define a gesture” to “Include the gesture in a repository”, is a manner to “Include a gesture in the repository” in a context of gesture-based interaction definition.



**Figure 28.** An excerpt of the map representation of gestUI

A map is graphically represented as a directed graph from Start to Stop. Intentions are represented as nodes and strategies as edges between nodes (see the map representation of the gestUI method in Figure 29). Dashed arrows represent strategies that have

methodological support but are not completely supported by the current version of the gestUI tool, described in the next chapter.

In Section 4.7.2 we explain the process to include custom gestures in a user interface (Step 3) and in Section 4.7.3 we explain how to redefine the existing custom gestures (Step 4).

#### **4.7.2 Including gesture-based interaction in a user interface**

In this section, we describe the process to include gesture-based interaction in a user interface using gestUI. We use the metamodel (Figure 26) and the map representation (Figure 29) to describe how is the process to include gesture-based interaction in a user interface.

Hence, in the following paragraphs, we describe the set of steps by means of intentions and strategies to include gesture-based interaction in a user interface with gestUI:

i. **A user opens a session in gestUI.** gestUI has two ways to allow users (e.g. developer, end-user, collaborative user) to establish a connection in a device (e.g. computer, notebook, smartphone) in order to define gestures:

- Intention: “*Open a local session*”. Users open a local session directly on the device.
- Intention: “*Open a remote session*”. Users open a remote session by means of an Internet connection.

The information of the user is stored in the “*User*” class.

ii. **The user defines gestures.** This definition can be performed by three ways:

- Intention: “*Directly sketching*” a gesture on the device in a local session and including it in the gesture catalogue (Figure 30).
- Intention: “*By sharing an existing definition*” of a gesture, that is, by importing a gesture definition in the gesture catalogue.
  - Intention: “*By sketching a gesture*” on the device in a remote session, that is, by using an Internet connection users sketch gestures and they are include in the gesture catalogue.

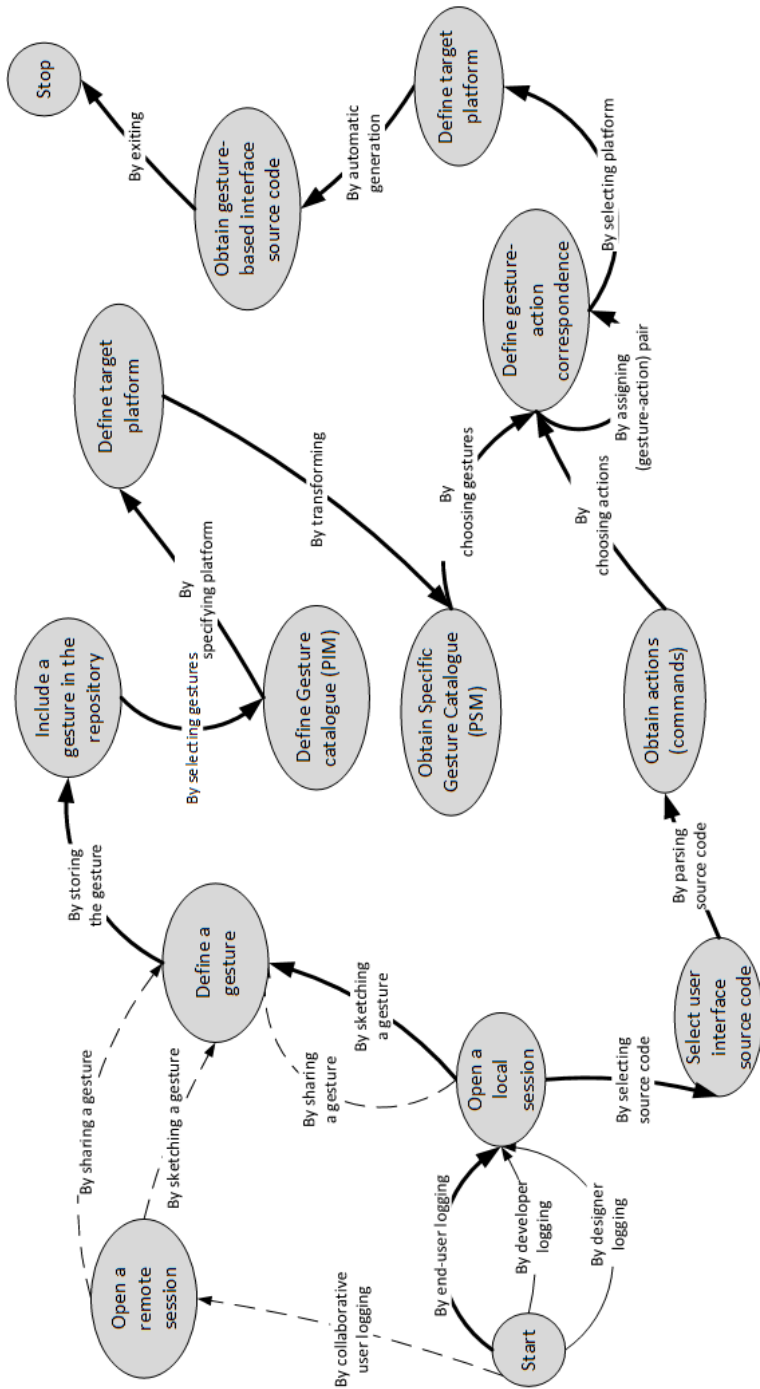


Figure 29. MAP representation of gestUI

**Table 21. Strategies of gestUI**

ID	Strategy	Description	Intention
1	Logging in as developer	The three types of users (developer, designer and end-user) can <i>log in to gestUI</i> tool support by means of a local connection using a user name and password. As result of this process, a user identification (UID) is assigned in order to identify the gestures definition of each user to obtain the gesture catalogue definition.	Open a local session
2	Logging in as end-user		
3	Logging in as designer user		
4	Logging in as collaborative user	This type of user can log in to gestUI by means of a remote connection using a username and password. As result of this process, a user identification (UID) is assigned in order to identify the gestures definition of each user to obtain the gesture catalogue definition.	Open a remote session
5	By sketching a gestures	Each user can use his/her fingers or a pen/stylus to <i>define a single or multi-stroke custom gesture</i> .	Define a gesture
6	By sharing a gesture	Each user can <i>define a custom gesture</i> by sharing a gesture which is stored in an external repository.	
7	By storing the gesture	Each gesture that is defined by a sketching task or shared from an external repository is <i>included in the repository</i> defined by gestUI.	Include a gesture in a repository
8	By selecting gestures	The user can select gestures from the repository to define <i>platform-independent gesture catalogue</i> .	Define gesture catalogue (PIM)
9	By specifying platform	The user <i>specifies a target platform</i> with the aim of applying a model transformation.	Define target platform
10	By transforming	Using a model-to-model transformation a <i>platform-specific gesture catalogue</i> is generated.	Obtain specific gesture catalogue (PSM)
11	By choosing gestures	The user is required to choose gestures to be considered in the <i>definition of gesture-action correspondence</i> .	Define gesture-action correspondence
12	By selecting source code	In order to include gesture-based interaction the <i>selection of a user interface source code</i> is required.	Select user interface source code

13	By parsing source code	Applying a parsing process, it is possible to <i>obtain the actions</i> (commands) included in the user interface source code.	Obtain (commands)	actions
14	By choosing actions	By choosing previously selected actions and gestures the user <i>defines the gesture-action correspondence</i> in order to include the gesture-based interaction.	Define	gesture-action
15	By assigning (gesture-action) pair	The user defines the gesture – action relation.		correspondence
16	By selecting platform	The user <i>selects the target platform</i> in order to apply an model-to-text transformation to obtain the source code	Define	target platform
17	By automatic generation	A new version of user interface source code is obtained by applying a model-to-text transformation considering previously defined transformation rules and source code generation. The result is the <i>gesture-based user interface</i> .	Obtain	gesture-based interface source code
18	By exiting	The process is finished when the user exits gestUI.	Stop	



The information obtained is stored in the “*Gesture*” class. Depending of the type of gesture (single-stroke or multi-stroke) the “*Stroke*” class contains one or more instances. The additional classes of the metamodel (*Posture*, *Precedence*, *Figure*, *Point*) are filled with information when the gesture is multi-stroke.

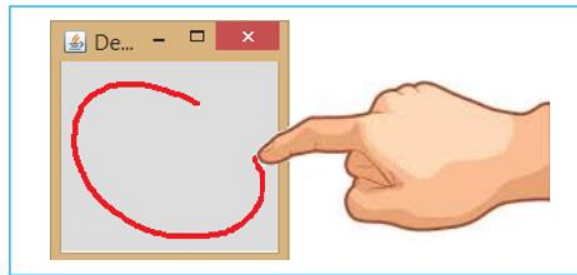


Figure 30. User defining a gesture

- iii. **The user includes a gesture in the repository.** A repository of gestures contains the gestures defined by the users.
  - Intention: “*Include a gesture in the repository*” by storing each gesture defined by the users in a repository.
- iv. **The user defines a platform-independent gesture catalogue.**
  - Intention: “*Define gesture catalogue (PIM)*” by selecting gestures from the repository according to the requirements specified by each user. The “*GestureCatalogue*” class is filled with information of each gesture included in the catalogue (Figure 31).
- v. **The user defines a target platform.**
  - Intention. “*The user defines a target platform*” to apply a model transformation.
- vi. **A platform-specific gesture catalogue (PSM) is obtained.**
  - Intention. “*Obtain a specific gesture catalogue*” as a result of apply a model-to-model transformation. In Figure 32 is described the user interface “*DrawingDiagrams*” with two users “*User1*” and “*User2*”. Each user has defined a gesture catalogue “*GestureCatalogueUser1*” and “*GestureCatalogueUser2*”. Each catalogue contains gestures defined for each user.

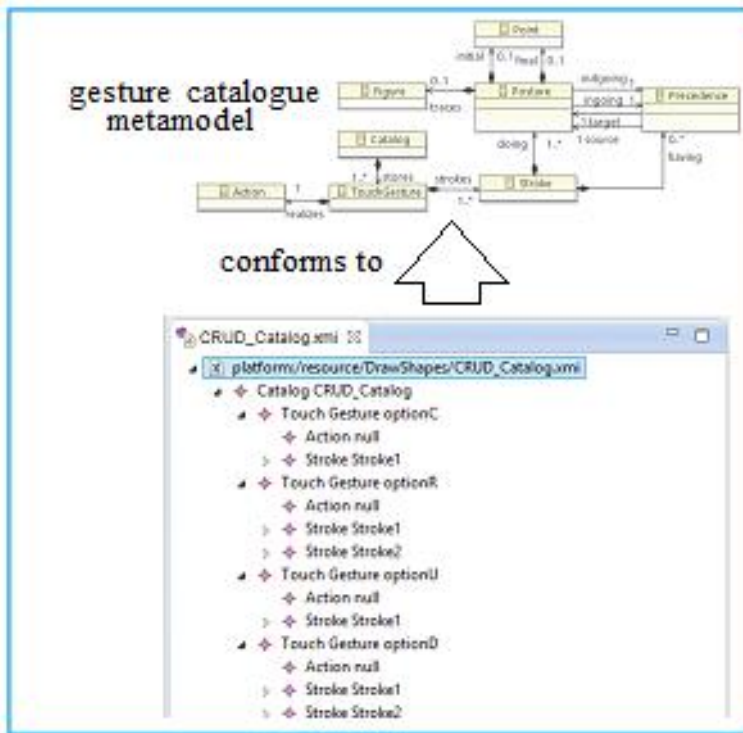


Figure 31. Platform-independent gesture catalogue

vii. **User selects the source code of a user interface.**

- Intention. “Select user interface source code” to include gesture-based interaction. This source code contains actions to perform the tasks involved with the user interface of the software system.

The “UserInterface” class is used in this intention to include the gesture-based interaction using gestUI.

viii. **We obtain the actions (commands) included in the user interface source code.**

- Intention. Obtain actions (commands) included in a user interface by applying a parsing process with the aim of searching keywords related with actions (Figure 33). The keywords depend on the programming language used to write the source code. For example, in Java, elements such as panel, button, label, etc. can be used to define actions in a user interface:

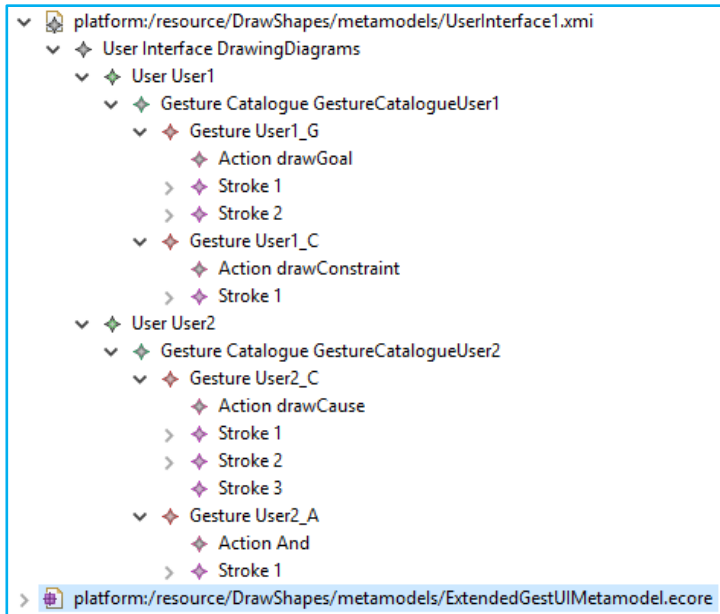


Figure 32. A specific-platform gesture catalogue

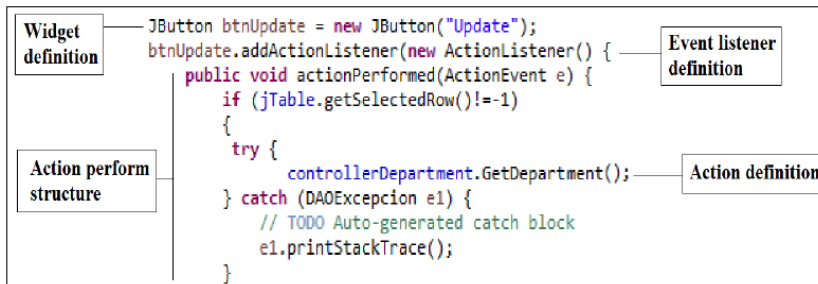


Figure 33. An excerpt of the source code of a user interface containing widget definition and keywords

The “Action” class is filled with information about the actions (commands) included in the user interface specified in the process.

- ix. **We define the gesture-action correspondence** to apply in the user interface.
  - Intention. “Define a gesture-action correspondence” as a one-to-one relation between a gesture of the gesture catalogue and an action included in the user interface source code.
- x. **The user defines a target platform.**

- Intention. *The user “defines a target platform” to apply a model transformation.*
- xi. **We obtain gesture-based interface source code** as a result of the model-to-text transformation.
- Intention. *“Obtain the gesture-based interface source code” corresponding to a user interface including gesture-based interaction.*

The strategies are a way of achieving an intention. In this case, we specify the strategies that permit to achieve each intention described in this section. Table 21 has four columns: “ID” column identifies the number of strategy. “Strategy” column contains the name of the strategy, the “Description” column includes a short explanation of each strategy contained in the map representation of gestUI, and the “Intention” column describes the intention related with the strategy.

#### **4.7.3 Redefining a gesture during the execution time**

If the user wants to change the initial specification of the gestures (redefine them) because he/she has problems to remind them or he/she has problems to sketch them, then it is needed to include some tools to permit the modification of the initial gesture catalogue specification in the user interface with the aim of improving the human-computer interaction.

In this section, we explain how a user can redefine an existing gesture directly in the software system during the execution stage (runtime). As is mentioned before, the process to redefine an existing gesture must be included in the software system containing the user interface with gesture-based interaction.

In this case, we use a map representation (Figure 34) to demonstrate how is the process to redefine custom gestures in the software system containing the gesture-based user interface. Then, this map shows the intentions and strategies to use custom gestures in the user interface and to redefine existing custom gestures in the software system.

The redefinition process consists in that the user again sketches the custom gesture according his/her preferences and then this gesture is included again in the gesture catalogue to be used in the software system to perform the same action defined in the beginning of the process when this gesture was defined with gestUI (as is explained in Section 4.7.2).

In the following paragraphs, we describe the set of steps by means of intentions involved in the map representation (see Figure 34) and the classes included in the metamodel (Figure 26) to redefine gestures included in a user interface supporting gesture-based interaction:

- i. **The user log in to the software system.** The software system has one way to allow that users (e.g. developer, end-user, collaborative user) establish a connection in a device (e.g. computer, notebook, smartphone) to use a user interface with gesture-based interaction included:
  - Intention: “*Log in to software*”. Users (developer, end-user) open a session directly on the software system.
  - Intention: “*Log in to software*”. User (collaborative user) opens a remote session on the software system.

When the user is logged in to the software system he/she obtains a user identification (UID). This UID is related with the previously defined gesture catalogue included in a user interface to support gesture-based interaction.

The “*User*” class of the metamodel contains the information required to log in to the software system.

- ii. **The user chooses a user interface** according to the task to perform in the software system:
  - Intention. “*Use gesture-based user interface*” to perform some task by means of gestures

The “*UserInterface*” class is referred to the user interface with gesture-based interaction included.

- iii. **The user performs some actions by drawing gestures** in the user interface of the software system.
- Intention. *“Use gestures in the user interface”* to perform some actions in the software system by means of gesture-based interaction.

The *“Gesture”* and the *“Action”* classes define the gesture-action correspondence to perform actions in the user interface by means of gestures.

- iv. **If the user has problems with the gestures** included in the user interface, then he/she can redefine them.
- Intention. *“Redefine custom gestures”*. This redefinition can be done by two ways: (a) by sketching gestures on a canvas in the software system, or (b) by sharing gesture definition through an Internet connection. In this case, the *“Gesture”* class is modified with the new information of the redefined gesture. The other classes (*“Stroke”*, *“Posture”*, *“Precedence”*, *“Figure”*, and *“Point”*) are also modified with the new information of the custom gesture.
  - Intention. *“By including gestures”*. When the redefinition of the gesture is ready, it is needed to include again this gesture in the gesture catalogue defined in the software system. The *“GestureCatalogue”* class is modified with the information of the recently redefined gestures.

The strategies included in the software system containing user interface supporting gesture-based interaction are described in Table 22 that has four columns: “ID” column identifies the number of strategy. “Strategy” column contains the name of the strategy, the “Description” column includes a short explanation of each strategy contained in the map representation of the software system, and “Intention” column describes the intention related with the strategy.

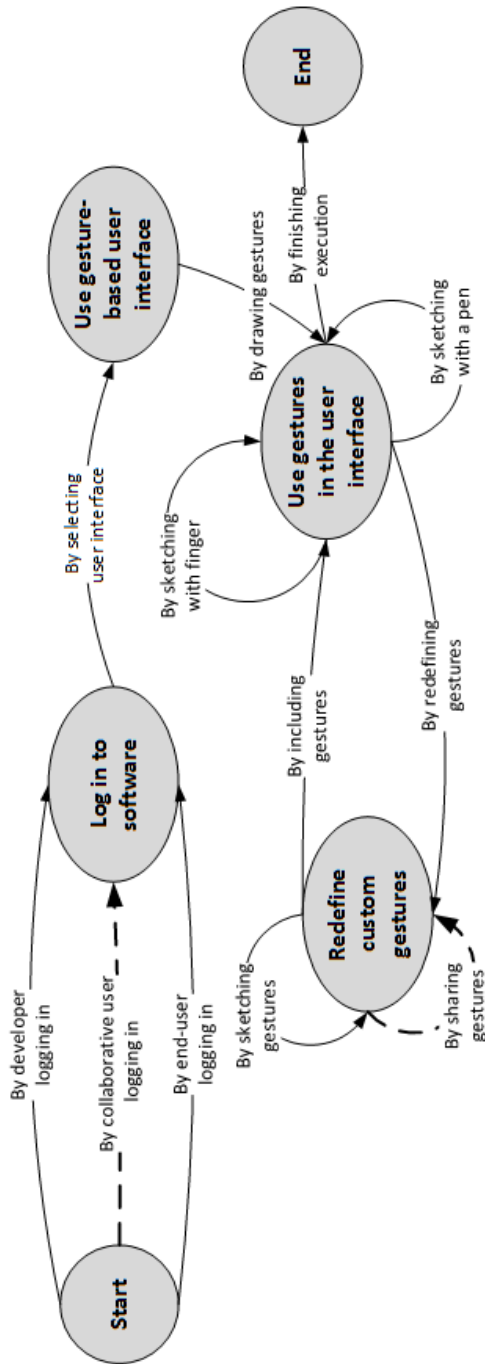


Figure 34. Map representation of the software system with the redefinition feature included

**Table 22. Strategies of the software system with gesture-based interaction**

ID	Strategy	Description	Intention
1	Logging in as developer	The two types of users (developer and end-user) can <i>log in to the software system</i> by means of a local connection using a user name and password. As result of this process, a user identification (UID) is assigned in order to identify the previously defined gesture catalogue.	Log in to software
2	Logging in as end-user		
3	Logging in as collaborative user	This type of user can log in to gestUI by means of a remote connection using a username and password. As result of this process, a user identification (UID) is assigned in order to identify the previously defined gesture catalogue.	Use gesture-based user interface
5	By selecting user interface	The user selects a user interface to perform some task in the software system.	
6	By drawing gestures	The user can sketch a <i>gesture</i> contained in the gesture catalogue with the aim of performing an action in the user interface.	Use gestures in the user interface
7	By sketching with a finger	The user sketches a gesture using his/her finger on a canvas in the interface.	
8	By sketching with a pen	The user sketches a gesture using a pen/stylus on a canvas in the user interface.	Redefine custom gestures
9	By redefining gestures	If the user has some problem with the gesture (e.g. it is hard to remind or it is hard to sketch) he/she can redefine it.	
10	By sketching gesture	The redefinition of the gesture can be done by sketching the gesture with a finger or a pen/stylus.	Use gestures in the user interface
11	By sharing gesture	The redefinition of the gesture can be done by sharing an existing definition in other gesture catalogue.	
12	By including gesture	When the user finishes the redefinition of the gesture, he/she requires to include the new definition of the gesture in the current gesture catalogue in order to do available this gesture in the user interface.	End
13	By finishing execution	The user finishes the execution of the software system.	



## 4.8 Summary

This chapter presents an integral proposal for the development of user interfaces of software systems with gesture-based interaction for any device platform. This proposal is based on the application of concepts (metamodel, model and model transformations) of model-driven paradigm.

The application of a model-based approach is justified by two aspects: the need to raise the level of abstraction of the process, and the possibility of applying a methodological approach. This model-based approach involves M2M and M2T transformations to convert PIM to PSM, and models to source code.

This chapter has described the features of the method to develop, the components and model transformations that comprise it and the relationship between them. Finally, it has presented an overview of the proposed method.

Additionally, we describe the gesture redefinition feature that permits to redefine gestures according to the needed and preferences of the users.

---

gestUI

# 5

TOOL  
SUPPORT

The topics covered in this chapter are:

- 5.1 Introduction
- 5.2 Components of the tool support
- 5.3 Development methodology of the tool support
- 5.4 Implementation of the tool support
- 5.5 Demonstration of the tool support
- 5.6 Summary and Conclusions



## Chapter 5. gestUI Tool Support

### 5.1 Introduction

Software development process is always a challenging activity, especially because software systems are becoming more and more complex with the introduction of the called natural user interaction in the user interfaces. This situation permits that software development process is shifting its attention towards MDD because it has demonstrated positive influences for reliability and productivity of the software development process.

The previous chapter first outlined a conceptual model for define custom gesture catalogue and then defined the model-driven method called gestUI to define custom gestures and to include the gesture-based interaction in user interfaces. This method has been defined following the MDD principles, as models drive its application, and the gestUI tool support has been built to support its models and activities. This method has been defined to guide the custom gesture definition and the inclusion of gesture-based interaction in the user interfaces of software systems.

In this context, the support of the tool is a valuable asset allowing the definition of the gesture catalogue model and supporting the necessary transformations to obtain the source code of the user interfaces supporting gesture-based interaction.

The remainder of this chapter is structured as follows. After this introduction, a description of each one of the components of the tool support is presented in Section 5.2. Section 5.3 describes the methodology adopted for the implementation of the tool support. Section 5.4 contains the description of the implementation of the tool support. Section 5.5 includes a demonstration of the applicability of the tool support in a form-based software system and in a Case Tool. Finally, Section 5.6 ends this chapter by presenting the conclusions.

## 5.2 Components of the tool support

The main idea behind the tool support is to facilitate a graphical environment for the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces. Then, in order to demonstrate the applicability of the proposed method we implemented with Java programming language and Eclipse Modelling Framework a prototype of the tool support structured into three systems (Figure 35):

- (i) The *information system* with interfaces where we aim to include gesture-based interaction.
- (ii) The *gestUI tool* to include the gesture-based interaction in user interfaces.
- (iii) A *framework to test the gestures* defined using gestUI (i.e. quill, iGesture, \$N).

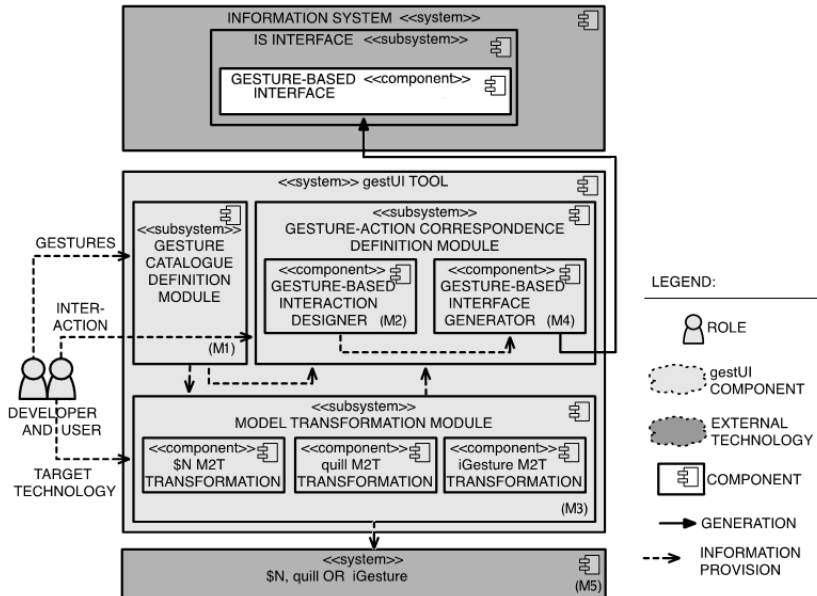
Regarding the second system (gestUI tool), by using the Java programming language and the Eclipse Modelling Framework we implement it.

The main features of gestUI tool support are:

- (i) The definition of custom gestures catalogue to execute actions in the user interfaces.
- (ii) The inclusion of gesture-based interaction in the user interfaces of a software system by specifying the gesture-action correspondence.
- (iii) The definition and the execution of model transformations to obtain PIM, PSM and source code of user interfaces of the software system.

The user interface of the tool support is composed by three options. Each option corresponds to one of the above main feature and it is related with one of the three subsystems, as shown in Figure 35: Gesture Catalogue Definition Module, Gesture-Action Correspondence Definition Module and Model Transformation Module. Next we describe these subsystems.

Each component of the tool support is implemented according to the corresponding component of *gestUI* described in Chapter 4. The implementation of each component is described in Section 5.4.



**Figure 35. *gestUI* tool support**

In the following sections are described each one of the subsystems included in *gestUI*:

### 5.2.1 Subsystem “Gesture Catalogue Definition Module”

The “Gesture Catalogue Definition Module” subsystem (Figure 36) provides functionalities for defining custom gestures and it is responsible for the execution of the model-to-model transformation to obtain the gesture catalogue model.

Therefore, this subsystem requires as input the custom gestures sketched by the users on a touch-based surface. As output, the subsystem produces the gesture catalogue model.

The subsystem contains the M1 activity described in the following paragraphs.

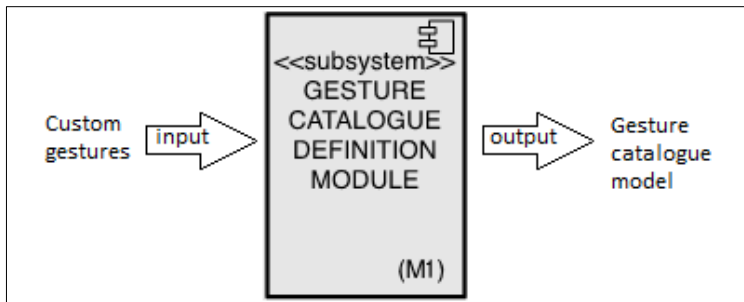


Figure 36. An excerpt of Figure 35 showing the subsystem "Gesture Catalogue Definition Module"

- i. **M1 Activity:** Firstly, the subject draws custom gestures using a finger (or a pen/stylus) on a touch-based screen (Table 21, Intention "Define a gesture" and Strategy "By sketching a gesture"). Each gesture is stored in a repository (Table 21, Intention "Include a gesture in a repository" and Strategy "By storing the gesture"). Then, in order to define the platform-independent gesture catalogue (Table 21, Intention "Define gesture catalogue (PIM)"), the subject chooses one or more gestures from the repository (Table 21, Strategy "By selecting gestures") and then they are inserted in the gesture catalogue model. This gesture catalogue model (conforms to the metamodel described in Chapter 4) is the input for the "Model Transformation Module" and the "Gesture-Action Correspondence Definition Module" subsystems.

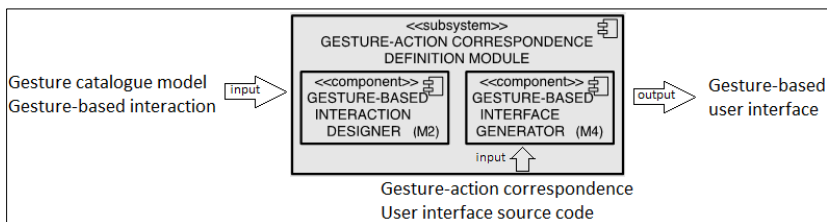
This subsystem gives as result the gesture catalogue model. This model is used in the other subsystem as input.

### 5.2.2 Subsystem "Gesture-Action Correspondence Definition Module"

The "Gesture-Action Correspondence Definition Module" subsystem provides functionalities for defining the gesture-action correspondence that consists in the relation between a custom gesture of the gesture catalogue and an action contained in a user interface.

We apply a parsing process (Table 21, Intention “*Obtain actions (commands)*”) to obtain the actions included in the source code of a user interface. The parsing process (Table 21, Strategy “*By parsing source code*”) has as input the source code of a user interface. This process is based on the search of keywords according to the syntax of the programming language in which is written the source code and the primitives (e.g. button, panel) that are included in the user interface.

This subsystem contains two components (Figure 37): Gesture-based interaction designer (M2) and gesture-based interface generator (M4). Each one of these components are described here:



**Figure 37. An excerpt of Figure 35 showing the subsystem "Gesture-action Correspondence Definition Module"**

**Component: “Gesture-based Interaction Designer”.** This component provides the functionalities for defining the gesture-action correspondence in order to include gesture-based interaction in a user interface. The inputs for this component are: the gesture catalogue model (from M1) and the user interface to include gesture-based interaction. This subsystem contains the M2 activity (Figure 37):

- ii. M2 Activity: This defines the gesture-action correspondence through the following process: it begins selecting a user interface source code (Table 21, Intention “*Select user interface source code*” and Strategy “*By selecting source code*”) with the aim of analysing it and finding the actions included in it by applying a parsing process. The parsing process permits the discovery of a set of actions by means of checking the source code to search strings (or substrings) containing keywords (e.g. in the Java programming language: JButton, JPanel) [131].



The process of defining gesture-action correspondence (Table 21, Intention “*Define gesture-action correspondence*”) takes as input two arguments: (i) the previously defined gesture catalogue model (Table 21, Intention “*Obtain specific gesture catalogue (PSM)*”) with the aim of assigning each gesture with an action; (ii) the source code of a user interface (Table 21, Intention “*Select user interface source code*”) to search keywords related with actions (Table 21, Intention “*Obtain actions (commands)*”) contained in the structure of source code that is based on a programming language such as Java (e.g. JButton to define a button, JPanel to define a panel).

As a result of this process we obtain a set of actions included in the user interface. Therefore, if any action is found, a one-to-one relationship is defined between this action and a gesture.

**Component: “Gesture-Based Interface Generator”.** This component has the functionalities to apply a model-to-text transformation with the aim of generating the source code of the user interface with gesture-based interaction included. The inputs for this component are: gesture-action correspondence and the user interface source code. The output of this component is the new version of the user interface source code. It contains the M4 activity (Figure 37):

- iii. **M4 Activity:** This executes a model-to-text transformation in order to apply a code generation process (Table 21, Intention “*Obtain gesture-based interface source code*”) to obtain the new version of the user interface source code containing gesture-based interaction.

Considering the source code of the user interface, and by using an automatic process, we insert each gesture-action correspondence in the corresponding component of the user interface. This process is iterative while any action is found in the source code of the user interface. Finally, we apply a code generation process obtaining the user interface with gesture-

based interaction included (Table 21, Strategy “*By automatic generation*”).

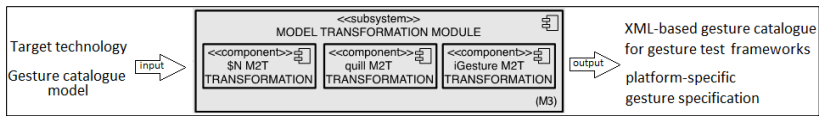
### 5.2.3 Subsystem “Model Transformation Module”

The “Model Transformation Module” subsystem provides the functionalities required to apply the model-to-model transformation and the model-to-text transformations included in the process for obtaining gesture-based interaction. The inputs for this subsystem are: gesture catalogue model and the target platform to perform each one of the model transformations.

This subsystem contains the M3 activity (Figure 38):

- iv. **M3 Activity:** This includes the transformation rules and the scripts written in ATL and Aceleo to apply M2M and M2T transformations, respectively. This activity requires two inputs: the gesture catalogue definition model and the target technology.

Firstly, a M2M transformation (Table 21, Strategy “*By transforming*”) is performed to obtain the gesture catalogue model (Table 21, Intention “*Obtain specific gesture catalogue*”) according to the specification of the gestures to be used in the gesture recogniser algorithm. In this case, we consider as target platform (Table 21, Intention “*Define target platform*” and Strategy “*By specifying platform*”) the \$N gesture recogniser and we obtain the platform-specific gesture catalogue specification. In a second place, an M2T transformation is performed to obtain a gesture catalogue to be included in two frameworks to test gestures (Table 21, Strategy “*By selecting platform*”): (i) quill [89] using GDT 2.0 to describe the gesture catalogue and (ii) iGesture [48] using XML to describe the gesture catalogue. Finally, another M2T transformation (Table 21, Strategy “*By automatic generation*”) is performed to obtain the user interface source code including gesture-based interaction (Table 21, Intention “*Obtain gesture-based interface source code*”).



**Figure 38. Excerpt of Figure 35 showing the subsystem "Model Transformations Module"**

### 5.3 Development methodology of the tool support

We followed a standard software development process and applied various techniques encompassing the specification and validation of software requirements, the modelling of the system architecture, the design of the software and user interface, the use of standard programming practices, and the validation of the resulting software application [132].

In this thesis, we use Design Science methodology which supports a pragmatic research paradigm promoting the creation of artifacts to solve real-life problems [33]. As suggested by the design science approach, we conducted an ongoing evaluation of the tool based on its application to a concrete case to ensure its usefulness in a concrete setting.

In order to give the reader a more concrete understanding of the various artefacts used by our tool, we will illustrate their concrete application to two cases: (i) in a framework to test gestures and (ii) in a software system to manage information. Yet, this example is not only intended to facilitate the understanding of the tool by showing its application to a concrete case, but also to evaluate its applicability in a real context to define (and to test) custom gestures and to include gesture-based interaction in user interfaces of a based-form software system.

### 5.4 Implementation of the tool support

According to our proposal described in Chapter 4, regarding the components included in gestUI, the implemented tool support requires three options (Figure 39):

- (i) *“New Catalogue”* to define gesture catalogue model (Table 21, Intention *“Define gesture catalogue (PIM)”*).
- (ii) *“Specific Catalogue”* associated with platform-specific gesture specification (Table 21, Intention *“Obtain specific-gesture catalogue (PSM)”*).
- (iii) *“Gesture-Action”* to define gesture-action correspondence and source code generation (Table 21, Intention *“Obtain gesture-based interface source code”*).

Figure 39 shows a screenshot of the main interface of gestUI tool support.



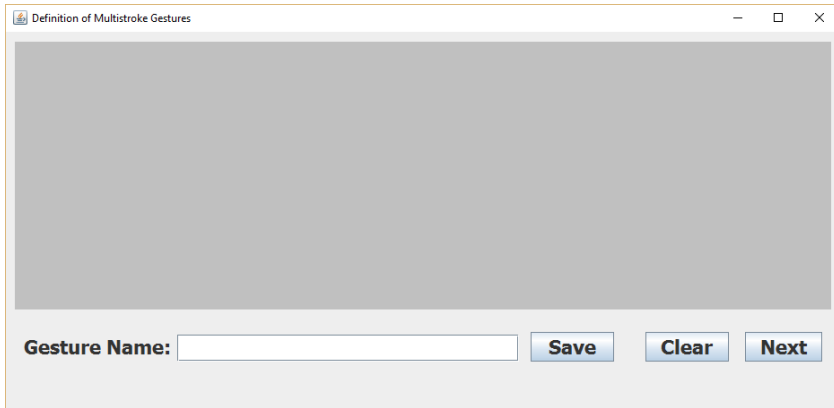
**Figure 39. Main interface of the tool support**

The options (i) and (ii) correspond to the implementation of the *“Gesture Catalogue Definition Module”* subsystem described in Section 5.2.1 and *“Model Transformation Module”* subsystem described in Section 5.2.3.

The option (iii) corresponds to the implementation of the *“Gesture-Action Correspondence Definition Module”* subsystem described in Section 5.2.2.

#### **5.4.1 Option 1: “Gesture catalogue definition”**

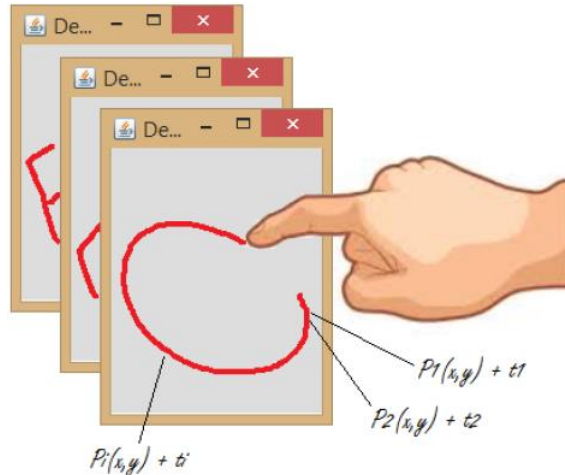
This module supports the definition of new multi-stroke gestures by means of an interface implemented in Java containing a canvas on which the user sketches the gestures. Figure 40 shows a screenshot of the interface implemented to sketch of multi-stroke gestures.



**Figure 40.** Screenshot of the interface of gestUI to sketch gestures

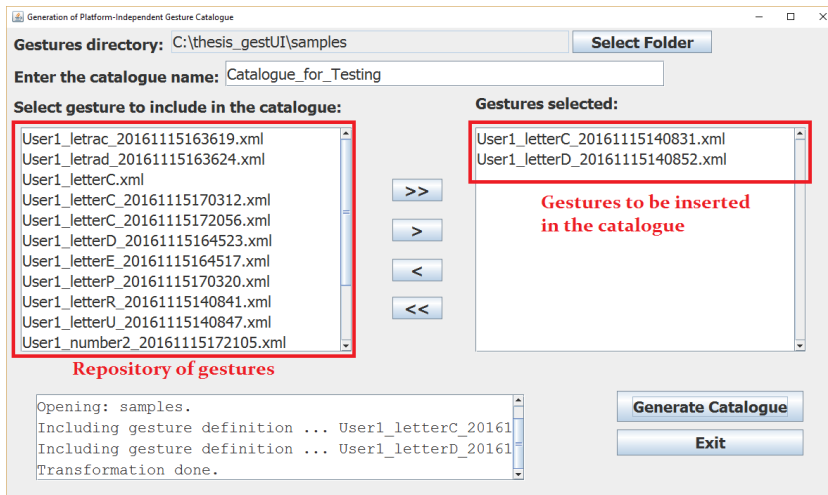
We adopt  $\$N$  as the gesture recognizer in this tool support. Then, when the gesture is sketched on a canvas, the following data are required: number of strokes specified during the sketching of the gesture, the information of each stroke, number of points contained in each stroke and the value of each point (X, Y) together with the timestamp (t) of each point (Figure 41).

Therefore, each gesture sketched by the user (Table 21, Intention "*Define a gesture*") consists of one or more strokes, each stroke is defined by a set of points described by coordinates (X, Y) and a timestamp (t).



**Figure 41. User sketching a gesture and storing it in a repository**

After capturing the data required by  $\$N$  to analyse each gesture, the data of each gesture are stored in a repository (Table 21, Intention “*Include a gesture in a repository*”) containing the gestures of the users registered in the software system (Figure 42, left), as is described in Section 5.2.1.



**Figure 42. Screenshot of the user interface to obtain the platform-independent gesture catalogue**

Then, by selecting gestures of the repository (Figure 42, left), the user defines the gestures to be inserted in the gesture catalogue model. In Figure 42, right, is shown the gestures selected for the

“Catalogue\_for\_Testing” gesture catalogue model. Finally, with the “Generate Catalogue” button, the platform-independent gesture catalogue is generated (Table 21, Intention “*Define gesture catalogue PIM*”).

#### 5.4.2 Option 2: “Specific catalogue”

This second option makes it possible to obtain the platform-specific gesture catalogue by means of an M2M transformation. The transformation rules are written in ATL. Figure 43 shows an excerpt of the rule in the model-to-model transformation.

```

-- rule to create Gesture in multiStrokeGesture
rule Gesture {
  from
    s1: MMTG!Gesture(s1.employs->size()=1)
  to
    t1: MMSG!Gesture (
      gestureName <- s1.gestureName, gestureType <- s1.gestureType,
      gestureDate <- s1.gestureDate, gestureTime <- s1.gestureTime,
      realizes <- thisModule.Action(s1.realizes),
      strokes <- s1.employs->collect(e|e.strokes->collect(d|thisModule.Stroke(d)))
    )
}

```

**Figure 43. An excerpt of a rule of the M2M transformation**

With the aim of applying a model-to-model transformation required in the process, we develop a module using Java programming language to implement the user interface. Figure 44 shows a screenshot of this interface.

**Figure 44. M2M transformation parameters**

The user must specify the following parameters in this interface: gesture catalogue model, gesture catalogue metamodel and the platform-specific gesture specification. As a result we obtain the

platform-specific gesture catalogue (Table 21, Intention “*Obtain specific gesture catalogue (PSM)*”).

### 5.4.3 Option 3: “Gesture-action correspondence definition”

This module allows the developer to specify the action to be executed when the gesture recogniser tool validates a gesture sketched by the user on the user interface. We currently provide automated support to code-centric developments made in Java, i.e. this module parses the source code of the user interface to obtain a list of actions.

This module requires two inputs (Figure 45): the previously created ‘*Gesture catalogue model*’ that is specified in the “Gesture Source Folder” text field in the interface and the user interface (e.g. a Java source code).

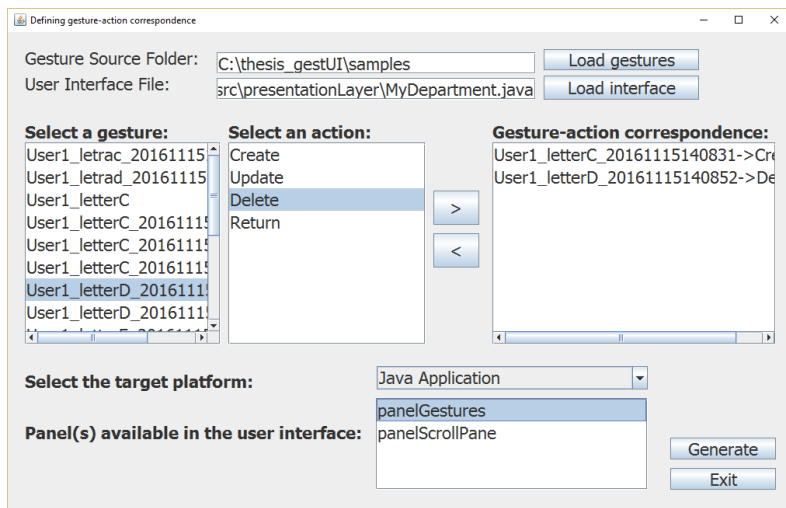


Figure 45. Interface for defining gesture-action correspondence and to generate source code

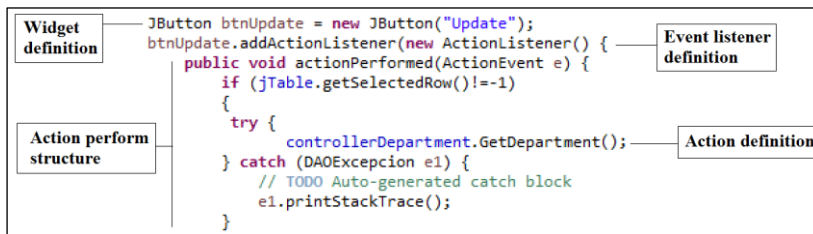
The output of this module is the source code of the previously specified user interface, but now it includes source code to support the gesture-based interaction.

In order to apply the parsing process in the user interface source code (Table 21, Intention “*Select user interface source code*”) we included some methods in the implementation of the tool support to analyse



two types of Java applications: (i) a Java desktop application using SWT, and (ii) Java desktop RCP application using JFace and SWT.

In the former type, SWT provides widgets (controls and composites) to be included in the user interface with the aim of assigning actions [22] (Table 21, Intention “*Obtain actions (commands)*”). The user interface source code also includes other sections containing event listeners and action-perform structures in order to specify the actions to be executed when the user clicks on a widget (canvas, button, text field, etc.) on the user interface (Figure 46). The parsing process then searches for these actions in order to complete the gesture-action correspondence definition (Table 21, Intention “*Define gesture-action correspondence*”).



**Figure 46. SWT components to define actions**

In the second type, in conjunction with SWT, JFace provides actions to allow users to define their own behaviours and to assign them to specific components, such as menu items, toolbar items, buttons, etc. [22]. In this case, the user interface source code includes structures to specify the actions to be executed when the user clicks on a widget in the user interface. These actions are taken during the parsing process in order to determine the gesture-action correspondence (Figure 47).

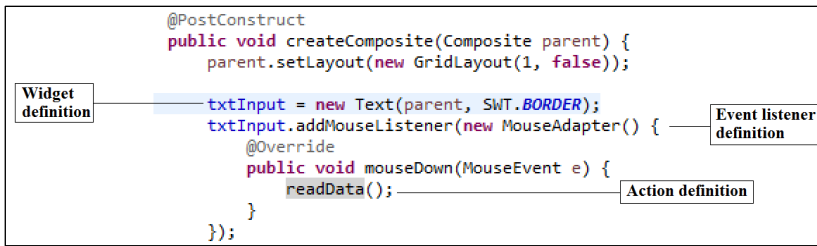


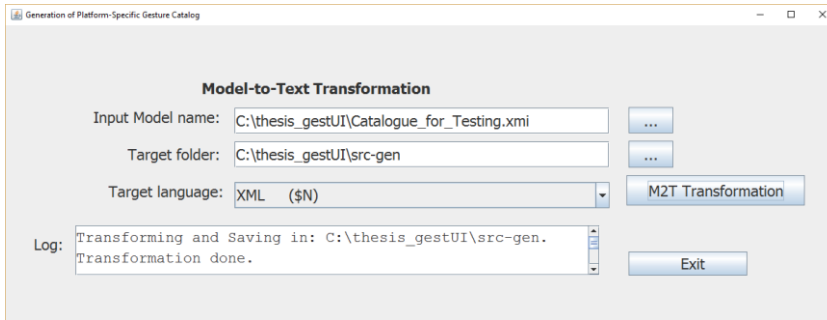
Figure 47. JFace and SWT components used to define an action in a user interface

The parsing process analyses the user interface source code searching for keywords corresponding to widgets available in Java language to include elements of a user interface (text, buttons, image, etc.). Each widget found in the process is stored in the table containing the gestures selected to define the gesture-action correspondence.

When generating the user interface Java source code, many references are included (e.g., to gestures management libraries, to gesture-recognition technology libraries (e.g. \$N)), and some methods are added (e.g. to execute the gesture-action correspondence and to capture gestures). Also, the classes' definition is changed to include some event listeners. Finally, the source code obtained from the complete process should be inserted in the complete source code of the user interface and, of course, be compiled again (Table 21, Intention “*Obtain gesture-based interface source code*”).

Additionally, we implemented a second model-to-text transformation to generate the gesture catalogue with the aim of testing the gestures using a gesture recognition tool, as is explained in Section 4.5.3. In this case, the following information is required: (i) the gesture catalogue name, (ii) the target platform, (iii) a folder name to store the source code generated.

Figure 48 shows a screenshot with the interface to apply the model-to-text transformation described in this section.



**Figure 48. Interface to execute a model-to-text transformation**

#### **5.4.4 Module to redefine gesture**

This module is not a component of gestUI tool support, however we implement it with the aim of demonstrating the gesture redefinition feature included in our proposal. This module is required in the software system containing user interface supporting gesture-based interaction.

We implement this module to be included in the software system with the aim of redefining custom gesture in the runtime stage (execution stage).

The interface contains two canvas to manage custom gestures (Figure 49): (i) it permits to show the current definition and (ii) it permits to sketch the new definition of the custom gesture.

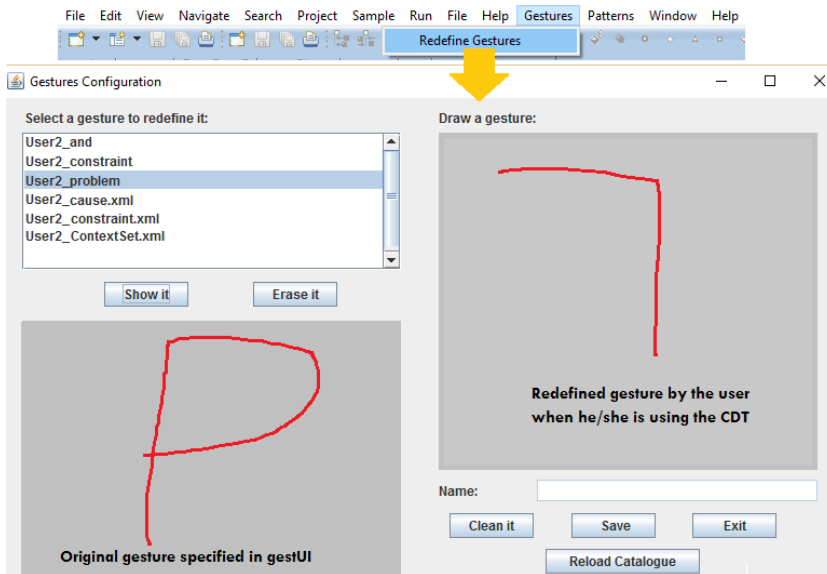


Figure 49. An example of the module to redefine custom gestures

## 5.5 Demonstration of the tool support

We applied gestUI and the tool support in two scenarios: (i) we use gestUI and the tool support to obtain a gesture catalogue to be used in the \$N, quill and iGesture frameworks; (ii) we used gestUI and the tool support to integrate gestUI into a code-centric user interface development method.

### 5.5.1 Applying the method and tool to testing a gesture catalogue

Using the tool support, we define a gesture catalogue containing three gestures to test them in the above frameworks: a triangle, a line and the letter "S" (Figure 50).

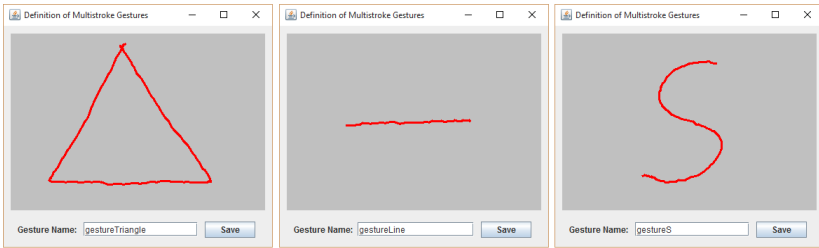


Figure 50. Gesture catalogue defined by gestUI

The gesture representation in each framework is contained in two sections: (i) a header specifying general information on the gesture, and (ii) the points specified by coordinates (X, Y) and a timestamp (t). \$N and iGesture employ XML for gesture definition and quill employs GDT 2.0 for this purpose (Figure 51).

To test the gestures we use the second M2T transformation described in Section 5.3.3, considering successively \$N, quill and iGesture as the target platform (Table 21, Intention “*Define target platform*”), with the aim of obtaining the gesture catalogue in the structure specified for each framework (Figure 51). In this case we specified the transformation rules with Aceleo and then we ran the M2T transformation for each framework.

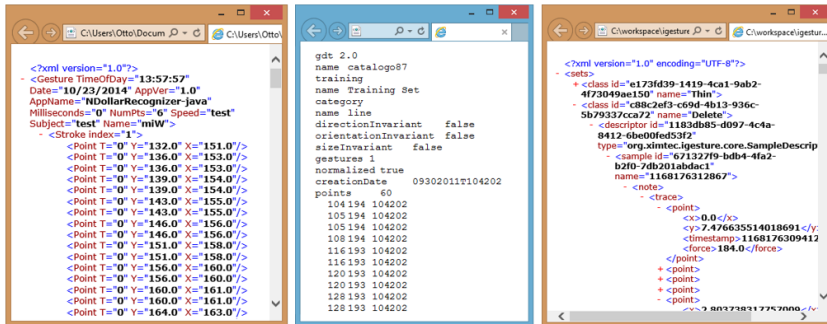


Figure 51. Gesture description files: \$N (left), quill (centre), iGesture (right)

In the next step, we use each framework to test the gestures. For instance, we include some quill interfaces. The quill interface used to import the gesture catalogue obtained in the model transformation is shown on left side of Figure 52. On the right, the gesture catalogue included in the framework can be seen.

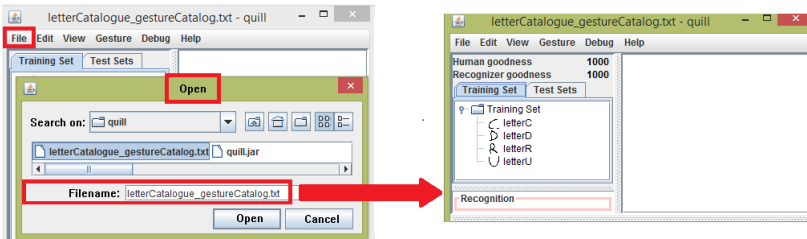


Figure 52. Importing the gesture catalogue to the quill framework

In the last step the user sketches the gestures contained in the gesture catalogue using the sketch area defined in the interface of each framework. All the frameworks include the algorithm (not described here) used to recognize the gestures sketched by the users. Figure 53 shows how the gesture catalogues are effectively recognised when imported to SN, quill and iGesture frameworks.

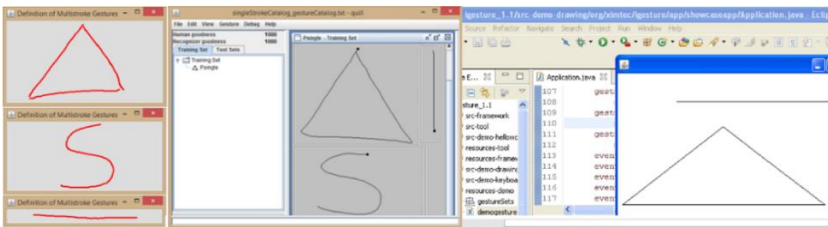
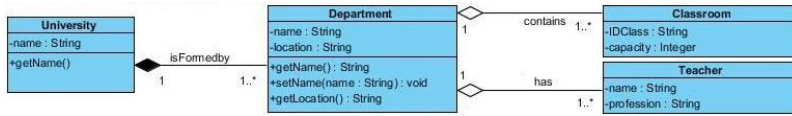


Figure 53. Examples of multi-stroke gestures: \$N\$ (left) and quill (centre) and iGesture (right)

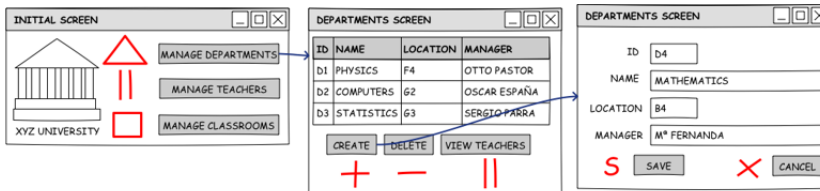
### 5.5.2 Applying the method and the tool to integrate gestUI into user interface development

For illustration purposes, we use a form-based information system, in this case a simple fictional university management case and we narrate the project as if it actually happened. Figure 54 shows the classroom management diagram of a university. In this section, we consider an IS with WIMP interfaces and for the sake of brevity, we only consider two interfaces for the demonstration: the main interface and department management interface. The form-based information system is developed in Java on Microsoft Windows.



**Figure 54. UML class diagram of the demonstration case**

In the first iteration, the university tells the developers that it would like the gestures to resemble parts of the university logo. They thus use the Gesture catalogue definition module to create the first version of the ‘Gesture catalogue model’ containing these three gestures:  $\triangle$  for departments,  $||$  for teachers and  $\square$  for classrooms. However, when the first user interface design is available (see Figure 55), they soon realise that other gestures are needed.










**Figure 55. Screen mockups (gestures are shown in red, next to action buttons)**

After defining and testing new gestures, they decide that navigation will be by means of the above-mentioned gestures, but that similar actions that appear on different screens will have the same gestures (e.g. the gesture  $+$  will be used to create both new departments and teachers).

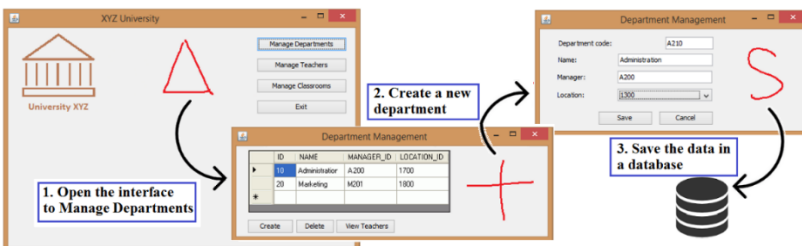
The developer assigns the gesture-action correspondence with the user, supported by the Gesture-action correspondence definition module. The correspondences are informally shown in Figure 55, next to each action button and are described in Table 23.

The user can employ the model transformation option to apply an M2T transformation and to obtain a platform-specific gesture catalogue. We consider that if the Java source code of the user interface using traditional keyboard and mouse interactions is available, then the components that support the gesture-based interaction can be generated. In this case, the underlying gesture-recognition technology chosen is \$N.

**Table 23. Platform-independent gesture catalogue definition**

Action	Gesture	Observations
Manage departments		This gesture opens a department management interface.
Manage teachers		This gesture opens a teacher management interface. The same gesture permits teacher information to be viewed.
Manage classrooms		This gesture opens a classrooms management user interface.
Create a new department		This gesture creates a new department, a new teacher or a new classroom.
Delete a department		This gesture deletes a department, teacher, or classroom.
Save the information		This gesture saves the information on a new department, teacher or classroom.
Cancel the action		This gesture cancels the process of creating a department, teacher or classroom.

As the users felt more comfortable with multi-stroke gestures (especially when tracing certain letters and symbols) quill was discarded. The final information system interface consists of several screens for managing university information. The users can still interact with the information system in the traditional way (i.e. by the keyboard and mouse) but now they can also draw the gestures with one finger on the touch-based screen to execute the actions.



**Figure 56. Using gestures to execute actions on the interfaces**

Figure 56 represents three interfaces from the information system:



- (i) Left: The task starts with the main interface where the users can select one of the options of the menu. For simplicity, the menu is showed as an array of buttons.
- (ii) Centre: According to the aforementioned requirements, if a user sketches the gesture “△” in the main interface of the information system then he/she obtains a second user interface containing the information on the existing departments.
- (iii) Right: In order to create a new department, he/she draws a “+” on this second user interface obtaining a third user interface with the fields for entering information on a new department. When the user finishes entering the information, sketching “S” on this third interface saves the information to a database.

## 5.6 Summary and Conclusions

In MDD is very important to provide tool support in order to promote the application of methods and tools.

This chapter describes the tool support implementation for the gestUI method. We applied Eclipse technologies since they have been applied successfully for supporting MDD methods and techniques. As programming language to implement the components of the tool support we used Java.

After the implementation, we assessed the method and tool support by applying them to a gesture testing case, generating the platform-specific gesture specification for three existing gesture-recognition technologies (quill, iGesture and \$N) in order to verify the tool's multiplatform capability. All the gestures were successfully recognised by the corresponding tools. When the proposed method was applied to a form-based IS, the final gesture-based interface components were automatically generated and successfully integrated into the IS interface. This process was applied in both Microsoft Windows and Ubuntu (Linux) systems to demonstrate its multiplatform capability.

The advantages of the proposed method are: platform independence enabled by the MDD paradigm, the convenience of including user-defined symbols and its iterative and user-driven approach.

Further developments should be performed around this prototype to make it more stable and usable.



---

EMPIRICAL  
EVALUATION

6

The topics covered in this chapter are:

- 6.1 Introduction
- 6.2 Experimental planning
- 6.3 Results
- 6.4 Discussion
- 6.5 Conclusions



## Chapter 6. Empirical Evaluation

### 6.1 Introduction

The next step in our engineering cycle to develop gestUI method is the design of the validation. The main objective is to validate gestUI in certain context to analyze the effects on its application.

Moody [133] considers that the objective of the validation should not be to demonstrate that the method is “correct” but that the method could be adopted based on its pragmatic success which is defined as “the efficiency and effectiveness with which a method achieves its objectives”. According to ISO 9241-210 [134] and ISO 25062-2006 [135], usability is defined as “the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use”.

Additionally, ISO 25062-2006 establishes that usability evaluation involves using (1) subjects who are representative of the target population of users of the software, (2) representative tasks, and (3) measures of efficiency, effectiveness and subjective satisfaction. The ISO also defines that at least one indicator in each of these aspects should be measured to determine the level of usability achieved [136]. In order to evaluate satisfaction, we consider the Method Evaluation Model (MEM) [137] [133] which contemplates three primary constructs: perceived ease of use – PEOU (“the degree to which a person believes that using a particular system would be free of effort”), perceived usefulness – PU (“the degree to which a person believes that using a particular system would enhance his or her job performance”) and intention of use – ITU (“the extent to which a person intends to use a particular system”).

With the aim of validating gestUI, we have designed a comparative empirical evaluation in which we consider two methods to define custom gestures and to include the gesture-based interaction. We evaluate efficiency, effectiveness and satisfaction (by means of PEOU,

PU and ITU) when the subjects apply gestUI (first method) in comparison with a code-centric method (second method) to include gesture-based interaction in existing user interfaces.

gestUI is described in Chapter 4. In Appendix A, we describe a generic code-centric method to include gesture-based interaction in existing user interfaces.

Results of this evaluation help to know to which extent the use of Model-driven development (MDD) helps in the process to define custom gestures and to include gesture-based interaction in user interfaces.

The remainder of this chapter is organized as follows: Section 6.2 describes the experimental planning. Section 6.3 includes the results obtained in the experiment. Section 6.4 includes the discussion about the results obtained in the experiment considering effectiveness, efficiency and satisfaction. Finally, the conclusions of the experiment are included in Section 6.5.

## 6.2 Experimental planning

This section describes the design of the experiment according to the guidelines of Wohlin et.al. [138].

### 6.2.1 Goal

According to the Goal/Question/Metric template suggested by Moody [133], the research goal is:

*Analyse the outcome of a code-centric and a model-driven method for including gesture-based interaction into user interfaces,*

*For the purpose of carrying out a comparative evaluation*

*With respect to their usability*

*From the viewpoint of researchers*

*In the context of researchers and practitioners interested in gesture-based interaction*

### 6.2.2 Research Questions and Hypothesis Formulation

The goal of our study is to compare the usability of a method to deal with gesture-based interfaces through code-centric versus model-driven. Since usability is an abstract concept, we need to operationalize it through more measurable concepts. According to ISO 25062-2006 [135], usability can be measured through effectiveness, efficiency and satisfaction. Following the works of Moody [133], satisfaction can be measured using perceived usefulness, perceived ease of use and intention to use.

We consider two scenarios in the experiment, the first one is related to the inclusion of gesture-based interaction (the subject follows a set of tasks specified in the experiment to include gesture-based interaction in a user interface) and the second scenario is related to the definition of custom touch gesture (the subject employs a finger or a pen/stylus to sketch a gesture on a touch-based surface).

Therefore, in the evaluation of efficiency and effectiveness we consider research questions (RQ1, RQ2, RQ3 and RQ4) to measure usability within each scenario, since we are interested in evaluating the subjects when they are including gesture-based interaction in the user interface and when they are defining gestures. However, for the evaluation of satisfaction (PEOU, PU and ITU) we consider research questions (RQ5, RQ6 and RQ7) without differentiating between scenarios, since we are interested in the global value of the method (code-centric and gestUI) for usability.

Considering this perspective, the research questions and the hypothesis proposed for the experiment are:

**RQ1:** *Regarding the inclusion of gesture-based interaction in user interfaces, is there any difference between the effectiveness of the code-centric method and gestUI?* The null hypothesis tested to address this research questions is:  $H_{01}$ : *There is no difference between the effectiveness of gestUI and*



*the code-centric method in the inclusion of gesture-based interaction in user interfaces.*

**RQ2:** *Concerning the definition of custom touch gestures, is there any difference between the effectiveness of the code-centric method and gestUI? The null hypothesis tested to address this research questions is: H<sub>02</sub>: There is no difference between the effectiveness of gestUI and the code-centric method to specify custom gestures.*

**RQ3:** *Regarding the inclusion of gesture-based interaction in user interfaces, is there any significant difference between the efficiency of the code-centric method and gestUI? The null hypothesis tested to address this research question is: H<sub>03</sub>: There is no difference between the efficiency of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces.*

**RQ4:** *Concerning the definition of custom touch gestures, is there any difference between the efficiency of the code-centric method and gestUI? The null hypothesis tested to address this research question is: H<sub>04</sub>: When the subjects define gestures, efficiency is the same independently of the method used.*

**RQ5:** *How do subjects perceive the usefulness of gestUI in relation to the code-centric method? The null hypothesis tested to address this research question is: H<sub>05</sub>: gestUI is perceived as easier to use than the code-centric method.*

**RQ6:** *How do subjects perceive the ease of use of gestUI in relation to the code-centric method? The null hypothesis tested to address this research question is: H<sub>06</sub>: gestUI is perceived as more useful than the code-centric method.*

**RQ7:** *What is the intention to use of gestUI related to the code-centric method? The null hypothesis tested to address this research question is: H<sub>07</sub>: gestUI has the same intention to use as the code-centric method.*

### 6.2.3 Factor and Treatments

Each software development characteristic to be studied that affects the response variable is called a factor [139] (a.k.a. “independent variable”). In this case, the factor detected in the experiment is the method to use and it has two treatments: the code-centric method and the model-driven method. Table 24 includes the description of the factor and its two treatments.

Eclipse Framework is used as a tool to operationalize the code-centric method. This tool is used to implement the source code in Java that represents a user interface. gestUI operationalizes the model-driven method. gestUI is used to include gesture-based interaction in a user interface through conceptual models (without writing any lines of code) [125].

**Table 24 Factor and treatments of the experiment**

Factor	Treatment		Description
	ID	Name	
Method to use	I	Code-centric method	Subjects manually write the source code to define custom gestures and to include gesture-based interaction in a user interface.
	II	gestUI	Subjects employ gestUI with the aim of defining custom gestures and including gesture-based interaction in a user interface.

### 6.2.4 Response variables and metrics

Response variables are the effects studied in the experiment caused by the manipulation of factors. In this experiment, we evaluate gestUI with regard to: effectiveness, efficiency and satisfaction.

#### 6.2.4.1 Response variables for effectiveness and efficiency

In this experiment, we are interested in the evaluation of the subjects when they define custom gestures using a finger (or a pen/stylus) on a touch-based surface, and we also are interested in the evaluation of the subjects using gestUI to include gesture-based interaction. Therefore, we need metrics to evaluate efficiency and effectiveness for each scenario.

In this experiment, in order to answer the research questions (RQ1, RQ2, RQ3 and RQ4), we define a metric per research question with the aim of evaluating the effectiveness and efficiency of gestUI when the subjects work in two scenarios: (i) they include gesture-based interaction in a user interface and (ii) they define custom gestures during the experiment. Table 25 shows the response variables classified per scenario and research question. The columns of Table 25 describe the response variables, their metrics, definition and the research question that they aim to answer.

#### *6.2.4.2 Response variables for satisfaction*

**In this experiment, in order to answer research questions RQ5, RQ6 and RQ7, we define a metric for each one with the aim of measuring satisfaction through PEOU, PU and ITU. We use a 5-point Likert scale in order to measure ITU, PEOU and PU. In this case we are not distinguishing between defining custom gestures and including gesture-based interaction in a user interface during the experiment, rather we are measuring satisfaction of the whole process.** Table 26 describes response variables, their metrics, definition and the research questions that we aim to answer.

**Table 25 Response variables to evaluate effectiveness and efficiency of gestUI**

Response variables	Metrics	Definition	Research question
<b>INCLUSION OF GESTURE-BASED INTERACTION</b>			
Effectiveness in the inclusion of gesture-based interaction	Percentage of correct tasks carried out in the inclusion of gesture-based interaction (PTCCI).	This is the relationship between: the number of tasks correctly completed and the total number of tasks during the inclusion of gesture-based interaction in the user interface	RQ1
Efficiency in the inclusion of gesture-based interaction	Time to finish the task during the inclusion of gesture-based interaction in the user interface (TFTI).	This is the number of minutes spent on each task. This is reported by the subjects during the inclusion of gesture-based interaction in the user interface.	RQ3
<b>CUSTOM GESTURE DEFINITION</b>			
Effectiveness in the custom gesture definition	Percentage of correct tasks carried out in the custom gesture definition (PTCCG).	This is the relationship between: the number of tasks carried out correctly and the total number of tasks during the definition of custom gestures	RQ2
Efficiency in the custom gesture definition	Time to finish the task during the custom gesture definition (TFTG).	This is the number of minutes spent on the experimental task. This is reported by the subjects during the definition of custom gestures.	RQ4

**Table 26 Responses variables to measure satisfaction of use gestUI<sup>12</sup>**

Response Variable	Metrics	Definition	Research question
Satisfaction	Perceived ease of use (PEOU)	This is the arithmetic mean of the Likert scale values of MEM questionnaire items related with perceived ease of use	RQ5
	Perceived usefulness (PU)	This is the arithmetic mean of the Likert scale values of MEM questionnaire items related with perceived usefulness	RQ6
	Intention to use (ITU)	This is the arithmetic mean of the Likert scale values of MEM questionnaire items related with intention to use.	RQ7

<sup>12</sup> We are aware that Likert scales are qualitative data but some studies propose converting them to quantitative to work with statistical tests [164].

Table 27 shows a summary of the research questions, hypotheses, response variables and metrics used to test these hypotheses.

**Table 27 Summary of RQ's, hypotheses, response variables and metrics**

Response Variables	Metric	RQ	Hypotheses
Effectiveness in the inclusion of gesture-based interaction	PTCCI	RQ1	H <sub>01</sub>
Effectiveness in the custom gesture definition	PTCCG	RQ2	H <sub>02</sub>
Efficiency in the inclusion of gesture-based interaction	TFTI	RQ3	H <sub>03</sub>
Efficiency in the custom gesture definition	TFTG	RQ4	H <sub>04</sub>
Perceived ease of use	PEOU	RQ5	H <sub>05</sub>
Perceived usefulness	PU	RQ6	H <sub>06</sub>
Intention to use	ITU	RQ7	H <sub>07</sub>

### 6.2.5 Experimental Subjects

The experiment was conducted in the context of the Universitat Politècnica de València (Spain). We had 21 subjects (15 males and 6 females) who are master (M. Sc.) and doctoral (Ph.D.) students in Computer Science. The experiment is not part of a course and the students are encouraged to participate on a voluntary basis.

The background and experience of the subjects are found through a demographic questionnaire handed out at the first session of the experiment. This instrument consists of 15 questions on a 5-point Likert scale. According to the questions included in the demographic questionnaire, the results are:

- Most of the subjects are between 25-29 (33%) and 30-34 years (24%).
- Regarding the computing platform, two of the most used are: Microsoft Windows (52% of the subjects) and MacOS (33%).
- All subjects indicated that they had taken a Java programming course. 62% of the participants had taken a model-driven

development (MDD) course and 52% of the subjects had taken a human-computer interaction (HCI) course.

- Regarding the software development experience using Eclipse IDE and Java, 43% of the subjects reported that they have “Average” self-rated programming expertise on a 5-point Likert scale, where 3 was “Intermediate” and 5 was “Expert”.
- Furthermore, the subjects reported their experience in model-driven development. The “Average” self-rated model-driven development expertise was 33% on a 5-point Likert scale where 3 was “Intermediate” and 5 was “Expert”. Also, in this field, 29% have a “Poor” level and 14% have a “Very Poor” level.
- Regarding experience using gestures on a device/computer, 71% of the subjects occasionally use gestures in their daily activities. Additionally, 43% of the subjects would like to define custom gestures to use them in their daily activities.

Table 28 summarizes the information about the subjects extracted from the demographic questionnaire. We conclude that subjects have some experience in the context of software development related with this experiment, but they do not have experience in the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces.

### **6.2.6 Experiment design**

In this experiment, we use a *crossover design* [138] (a.k.a. a paired comparison design). This is a type of design where each subject applies both methods, that is, the subjects use one method (the code-centric method) and then they use a second method (gestUI, a model-driven method) or vice versa. The order of use of each method depends on which group the subject was assigned to at the beginning of the experiment in such a way that each treatment is balanced among all the subjects. This design has the advantages that we are using the largest sample size to analyse the data, hence we avoid the learning effect and the problem is not confounded with the treatments.

**Table 28 Summary of demographic questionnaire**

	No. of subjects	%
<b>Average age</b>		
25-29 years	7	33.0
30-34 years	5	24.0
35-39 years	4	19.0
>39 years	5	24.0
<b>Gender</b>		
Male	15	71.4
Female	6	28.6
<b>Computing platform</b>		
Microsoft Windows	11	52.0
MacOS	7	33.0
Other	3	15.0
<b>Courses taken</b>		
Java	21	100.0
HCI	11	52.0
MDD	13	62.0
<b>Software development experience</b>		
Average experience	9	43.0
<b>Experience using gestures</b>		
	15	71.0
<b>Model-driven development experience</b>		
Average experience	7	33.0
Poor experience	6	29.0
Very Poor experience	3	14.0

With the aim of comparing both methods against each other, each subject uses both methods (treatments) on the same object; to minimise the effect of the order in which subjects apply the methods, we balanced the treatment applied in the first term. As Table 29 shows, the experiment is carried out with the subjects separated into two groups (G1 and G2). Each group is composed of subjects that are assigned according to a random value obtained by means of a random numbers calculator available on the Internet (<https://www.random.org/>). Therefore, the 21 subjects were randomly split into two groups following a process known as counterbalancing: (a) 11 subjects first apply gestUI and then the code-



centric method, whilst (b) the other 10 subjects start with the code-centric method and then apply gestUI.

**Table 29 Crossover design**

ID	Treatment	Subjects	
I	Code-centric method	G1	G2
II	Model-driven method (gestUI)	G2	G1

Even though there was no time limit to perform the experiment, the expected time to fulfill the tasks was around two hours. This value was estimated based on two factors: (i) a previous pilot test and (ii) using the KLM method (Keystroke Level Method) [140] [141]. KLM is a model for predicting the time that an expert user needs to perform a given task on a given computer system. KLM is based on counting keystrokes and other low-level operations, including the user's mental preparations and the system's responses [141]. Using this model, we estimate the time required to input the lines of code required in the code-centric method considering the operators and their average time proposed in [142] and shown in Table 30.

**Table 30 Operators and average time on KLM**

Operator	Description	Average Time	Observations
M	Mental Operation	1.2 sec.	Mentally prepare
H	Home	0.4 sec.	Home in on keyboard or mouse (change of device).
P	Point	1.1 sec.	Point with mouse
K	Keystroke	0.28 sec.	Keystroke or mouse button press
R(t)	System responsive	t sec.	Waiting for the system to become responsive (t)

The values of K operator is defined according to type of user: expert typist, average skilled typist, average non-secretarial typist, worst typist [140]. In this experiment, we consider the average of a non-secretarial typist. R(t) operator (t indicates the time in seconds that the user has to wait) defines the time when the computer is busy doing some processing, and the user must wait before they can interact with

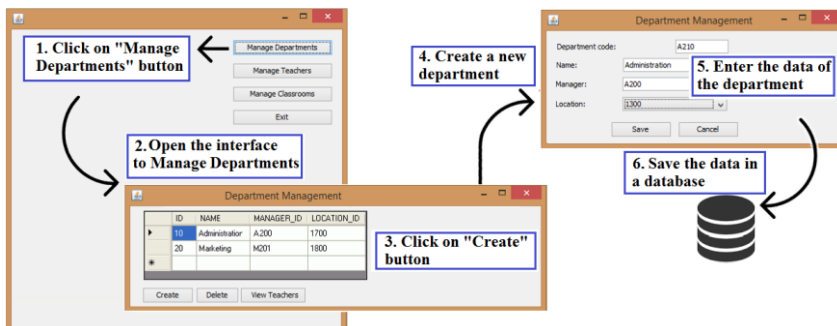
the system. The estimated values of time to perform the experiment are shown in Table 31.

**Table 31 Estimating time for the experiment**

Treatment	Previous pilot test	By using KLM
<b>Code centric method</b>	1h 08 min.	0h 57 min.
<b>Model-driven method</b>	0h 24 min.	0h 21 min.
<b>Total time</b>	<b>1h 32 min.</b>	<b>1h 18 min.</b>

### 6.2.7 Experimental objects

The object used in the experimental investigation is a requirements specification created for this purpose. It contains the description of a problem related with the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces of a software system supporting traditional interaction using a mouse and keyboard. Figure 57 shows this software system containing a main user interface to manage information of departments, teachers and classrooms in a university by means of CRUD (Create, Read, Update, and Delete) operations. Each option opens a new interface to specify information required by the university.



**Figure 57 Software system supporting traditional interaction**

Using traditional interaction, when the subjects click on the 'Manage Departments' button a new interface is opened, which contains the information of each previously defined department in a grid included

in the user interface. Next, clicking on the 'Create' button, a new interface is opened to enter information concerning a new department. Finally, when the information is complete, the 'Save' button saves the information in a database.

The user must perform the same CRUD operations but using custom gestures, that is, by means of gesture-based interaction. If gesture-based interaction is included in user interfaces, the subjects can sketch gestures on the touch-based display of the computer in order to execute some actions (the CRUD operations). One gesture can contain the definition of one or more actions, but the gesture-action correspondence must be unique per interface. Gestures are defined during the specification of the gesture-based interaction in each user interface. In this case, the 'D' gesture contains two actions (each one in a different interface): (i) it can be used to open the user interface to manage departments, and (ii) it can be used to delete one previously selected record in the database.

Even though the problem is small, it contains the necessary elements to validate the method: (i) a gesture catalogue definition containing the aforementioned six gestures, and (ii) the process to include the gesture-based interaction in the existing user interface source code. The inclusion of a greater number of user interfaces or gestures in the catalogue during the experiment would mean repetitive work for the subjects.

### **6.2.8 Instrumentation**

All the material required to support the experiment was developed beforehand, including the preparation of the experimental object, instruments and task description documents for data collection used during the execution of the experiment. The instruments used in the experiment are described in Table 32.

**Table 32 Instruments defined for the experiment**

Instrument	Description
Demographic Questionnaire	Questionnaire to assess the subjects' knowledge and experience of the technologies and concepts used in the experiment. This document includes questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree)
Task Document for the code-centric method	Document that describes the tasks to be performed in the experiment using the code-centric method and containing empty spaces to be filled in by the subjects with the start and end times of each step of the experiment. This document contains guidelines to guide the subject throughout the experiment and the source code to be included in the user interface.
Task Document for the model-driven method (gestUI)	Document that describes the tasks to be performed in the experiment using the model-driven method and containing empty spaces to be filled in by the subjects with start and end times of each step of the experiment. This document contains guidelines to guide the subject throughout the experiment.
Post-test Questionnaire for the code-centric method	Questionnaire with 16 questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree) to evaluate satisfaction of the whole process when the subjects use the code-centric method to define custom gestures and to include gesture-based interaction.
Post-test Questionnaire for the model-driven method (gestUI)	Questionnaire with 16 questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree) to evaluate satisfaction of the whole process when the subjects use the model-driven method (gestUI) to define custom gestures and to include gesture-based interaction.







### 6.2.9 Experiment procedure

This section describes the procedure used to conduct the experiment. Prior to the experiment session, a pilot test was run with one subject who finished the Master's degree in Software Engineering in the Universitat Politècnica de València. This pilot study helped us to improve the understandability of some instruments.

In this experiment, we consider a user interface of the existing software system mentioned in Section 6.2.7. In this user interface, users perform CRUD operations to manage information by means of a

traditional interaction with a mouse and a keyboard. We are interested in including gesture-based interaction in the user interfaces of a software system. So, the experiment addresses a real problem, i.e. the definition of custom gestures and the inclusion of gesture-based interaction in an existing user interface to perform the aforementioned operations.

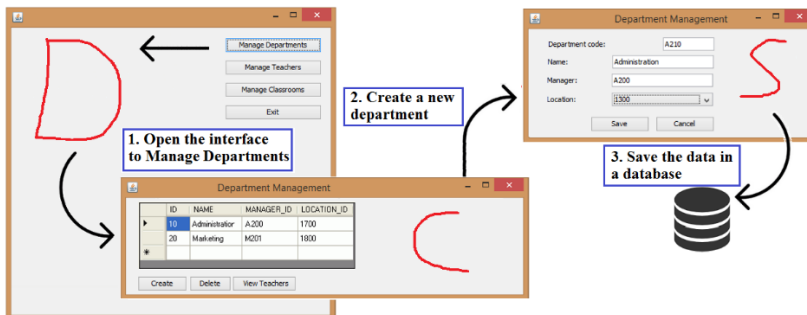
**Table 33 Gesture catalogue defined in the experiment**

Action	Gesture	Description	User interface
Open the "Managing Department" user interface		The user sketches this gesture to open the user interface to manage departments in the university.	Main user interface
Create a new department		The user sketches this gesture to open the user interface to create a new department.	Managing departments
Read a department record		The user sketches this gesture to open the user interface to read the previously selected record of a department.	Managing departments
Update the information of the existing department		The user sketches this gesture to open the user interface to update the previously selected record of a department.	Managing departments
Delete a record of a department		The user sketches this gesture to open the user interface to delete the previously selected record of a department.	Managing departments
Save the information of a department		The user sketches this gesture to save the information of a department in the database.	Department Information

Prior to the experiment, we define the gesture catalogue (see Table 33) that the subjects require to apply both treatments in the experiment.

The gesture catalogue consists of four gestures to execute each CRUD operations action and one additional gesture to save the information in the database ('S' gesture). Observe that the gesture 'D' has an overloaded meaning; that is, it triggers two distinct actions. However, note that the gesture is interpreted differently in the context of two distinct application windows (see the right-most column). Also, potential usability issues regarding this gestures are not relevant to the purposes of this experiment. This gesture catalogue is included in the Task Description Document of each treatment.

Hence, the user interface must contain the definition of gestures to perform CRUD operations. For instance, Figure 58 shows three gestures defined in the user interface: (i) 'D', to open the user interface to manage departments; (ii) 'C', to create a new department, by opening the user interface to enter the information of a new department; (iii) 'S', to save the information in the database.



**Figure 58 Software system supporting gesture-based interaction**

We consider two versions of the “Task Description Document”, as explained in Table 32. We use a sub-index ‘c’ when naming the task ID to express the treatment “Code-centric method” and we use a sub-index ‘g’ to express this treatment gestUI when naming the task ID. The subjects apply both treatments designed in the experiment with the aim of managing the input of gestures sketched by the users to execute actions in the software system.

Task Description Documents were delivered to the subjects before starting the experiment.

The steps in the procedure of the experiment are:

**Step 1:** The goal of the experiment was introduced to the subjects and guidelines on how to conduct the process were given to them.

**Step 2:** Each subject filled in a Demographic Questionnaire before starting the experiment where the subjects were asked about age, gender, courses taken, experience in software development, experience in model-driven development, and experience using gestures (Table 28). Results of this questionnaire are described in Section 6.2.5.

**Step 3:** The subjects did the experiment divided into two groups (G1 and G2) following the instructions given in the Task Description Document of each method. In this experiment, for each method, we separately evaluate two processes: (i) custom gesture definition and (ii) inclusion of gesture-based interaction, since we are interested in evaluating effectiveness and efficiency of the subjects when they specify gestures on a touch-based device and when they include gesture-based interaction. The evaluation of effectiveness and efficiency, taking in account PTCCG, PTCCI, TFTI, and TFTG (see Section 6.2.4) is performed based on the information registered in the Task Description Document. Next, we evaluate each method (code-centric and gestUI) in a global way with regard to PEOU, PU and ITU. The sequence of steps for each group is the following.

- G1 group. G1 subjects applied the code-centric method to complete Treatment I.

Treatment 1 (code-centric method). In this case, the subjects received the Task Description Document containing instructions to apply the code-centric method with the aim of adding new source code to **define custom gestures**. Following the instructions included in the Task Description Document, the subjects perform

a sequence of steps (see Table 34 that contains an excerpt of the Task Description Document) to define the catalogue of gestures described in Table 33. The definition of a gesture using the code-centric method consists of the creation of an XML file whose structure, in this case, is based on the gesture specification according to \$N gesture recogniser [126].

**Table 34 An excerpt of the Task Description Document containing the sequence of steps for custom gesture definition using the code-centric method**

No.	Task ID	Task Description	Observations
1	TG1 <sub>c</sub>	Definition of gesture "C"	The subject sketches the "C" gesture using a finger or a pen/stylus
2	TG2 <sub>c</sub>	Definition of gesture "R"	The subject sketches the "R" gesture using a finger or a pen/stylus
3	TG3 <sub>c</sub>	Definition of gesture "U"	The subject sketches the "U" gesture using a finger or a pen/stylus
4	TG4 <sub>c</sub>	Definition of gesture "D"	The subject sketches the "D" gesture using a finger or a pen/stylus
5	TG5 <sub>c</sub>	Definition of gesture "S"	The subject sketches the "S" gesture using a finger or a pen/stylus
6	TG6 <sub>c</sub>	Save gesture catalogue	The subject saves the gesture catalogue

We provided the subjects with an Eclipse project containing existing source code of the user interface. The subjects had to include additional lines of code in order to add functionalities related with gesture-based interaction. In a real industrial setting, in the worst case scenario, the developers would have to write such lines from scratch using the editor of the Eclipse IDE; in the best case scenario, they would copy them from another project or from a repository of software patterns and paste them in the current project. We opted for providing the subjects with the actual code they had to copy; we included the code and clear instructions in the Task Description Document.

The rationale for providing them with the source code is the following. On the one hand, it is true that this decision benefits the code-centric method because it reduces the time needed to



complete the task. On the other hand, we indeed needed to reduce the duration of the experiment. If the subjects had been forced to write the source code to define gestures from scratch and then include gesture-based interaction in the existing user interface, they would have probably required a greater number of hours (or maybe days!). We could not run such a long-term experiment without running into serious threats to the validity of the results (demotivation and exhaustion of the subjects potentially leading to an unacceptable mortality rate, loss of control over their activities outside of the laboratory leading to unreliable outcomes, etc.). We therefore consider that providing the source code was a good trade-off between relevance and rigour. Also, based on the pilot experiments we had evidence-based expectations that, nonetheless, the code-centric method would be less efficient than the model-driven method. If the difference between the efficiency of both methods is significant and in favour of gestUI, then we can still claim with confidence that, in a real setting where developers would even take longer to write the code, adopting gestUI would still benefit them in terms of gained efficiency.

An excerpt of the sequence of steps to perform in the experiment to **include gesture-based interaction** using the code-centric method is included in Table 35.

Tasks T11<sub>c</sub>, T12<sub>c</sub> and T13<sub>c</sub> allow the adaptation of the source code of \$N gesture recogniser in the source code of the user interface with the aim of adding a gesture recogniser in the software system to recognise the gestures sketched by the users. T14<sub>c</sub> includes a panel in the user interface where the gestures are sketched by using a finger or pen/stylus. T15<sub>c</sub> and T16<sub>c</sub> permit the inclusion of listeners to sense the finger that is sketching a gesture. These listeners capture the information produced on the user interface when a gesture is sketched. T17<sub>c</sub> and T18<sub>c</sub> manage the process to draw the gesture on the user interface. T19<sub>c</sub> implements a method

to define the gesture-action correspondence. In this case, the subject needs to execute a process to search actions included in the source code. We use a user interface where the actions are related with buttons definition (e.g. 'Manage Departments', 'Create', 'Save'). Subjects define the action–gesture relationship using the specification of gestures described in Table 33.

**Table 35 An excerpt of the Task Description Document containing the sequence of steps for gesture-based interaction inclusion using the code-centric method**

Task ID	Task Description
T11 <sub>c</sub>	To include \$N as gesture recogniser in the software system
T12 <sub>c</sub>	To implement methods and attributes required to use \$N as gesture recognition
T13 <sub>c</sub>	To implement the method to read gestures sketched by the user.
T14 <sub>c</sub>	To add a new panel in the user interface to draw gestures.
T15 <sub>c</sub>	To write a method to implement a listener sensing the finger (or pen/stylus) that is drawing a gesture.
T16 <sub>c</sub>	To write a method to implement a listener sensing that the gesture definition is complete.
T17 <sub>c</sub>	To implement a method to manage graphics in Java.
T18 <sub>c</sub>	To implement a method to paint a gesture on the user interface.
T19 <sub>c</sub>	To implement a method containing the gesture-action correspondence
T110 <sub>c</sub>	To compile the new version of the source code and to run the software system

As a final result, the subjects obtain a new version of source code containing gesture-based interaction in the user interface in order to execute actions indicated in the requirements specification using gestures. Then, in T110<sub>c</sub>, the subjects must compile the source code of the software system in Eclipse IDE, and then they can execute the software system in order to test the gestures defined in the process to execute the previously specified actions in the experiment.

- G2 group. G2 subjects employed gestUI to complete Treatment II.

Treatment II (gestUI). In this procedure, we consider the same user interfaces of the software system shown in Figure 57. G2 subjects received the Task Description Document containing instructions to apply gestUI to define custom gestures and to include gesture-based interaction in the user interface. This treatment consists of the definition of the gesture catalogue, and the specification of data to apply model transformations in order to generate the source code of the user interface containing the gesture-based interaction.

Firstly, the subjects define the gesture catalogue by means of a pen/stylus or a finger on a touch-based surface. These gestures are stored in a repository, as described in Section 4.5 of the Chapter 4, and then the **platform-independent gesture catalogue** (gesture-catalogue model) is obtained. The tasks to perform this step are included in Table 36, which shows an excerpt of the Task Description Document for this treatment.

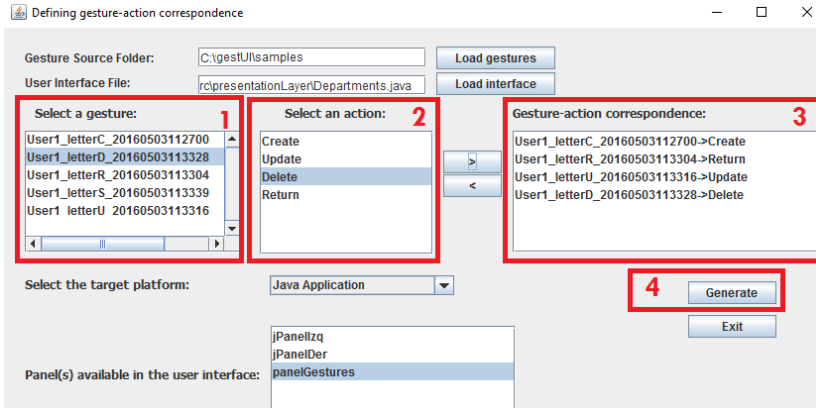
**Table 36 An excerpt of the Task Description Document for custom gesture definition using gestUI**

Task ID	Task Description
TG1 <sub>G</sub>	Definition of gesture "C"
TG2 <sub>G</sub>	Definition of gesture "R"
TG3 <sub>G</sub>	Definition of gesture "U"
TG4 <sub>G</sub>	Definition of gesture "D"
TG5 <sub>G</sub>	Definition of gesture "S"
TG6 <sub>G</sub>	Executing model-transformation to obtain a platform-independent gesture catalogue

Secondly, with the aim of obtaining the **platform-specific gesture specification**, subjects apply a model-to-model transformation that requires as input the gesture catalogue model.

Thirdly, the subject selects the user interface and the platform-specific gesture specification to design the gesture-based interaction by defining the gesture-action correspondence. This correspondence is defined with the aim of assigning each gesture

to an action. Figure 59 shows the interface of the tool that contains the process to define this correspondence consisting of steps 1 to 4 shown in red.



**Figure 59** Gesture-action correspondence definition using tool support

Table 37 contains the description of the steps shown in Figure 59.

Finally, gestUI generates the code with a new version of the user interfaces including gesture-based interaction. Then, the subjects use Eclipse IDE to compile the source code of the software system and afterwards they test the gestures defined in the process.

**Table 37** Gesture-action correspondence step-by-step definition

No.	Description	Explanation
1	It selects a gesture from the gesture catalogue	This contains the gesture selected by the subject.
2	It selects an action from the list of actions included in the user interface	This contains the actions selected by the subject.
3	It contains the gesture-action correspondence definition	The subject confirms the gesture-action correspondence.
4	It generates the new version of the source code of the user interface	This contains the process to generate the source code of the user interface containing gesture-based interaction.

At the end of this process, the result is the generated source code of the user interface of the software system supporting gesture-based interaction to execute actions, according to the definition of gesture-action correspondence. Figure 58 shows the same software system described in Figure 57 but supporting gesture-based interaction.

**Step 4.** Subjects filled in the corresponding Post-Test Questionnaire according to the treatment employed in the experiment.

According to Table 29, in Section 6.2.6, after the G1 subjects employed the code-centric method they must employ gestUI to complete Treatment II, repeating steps 1 to 3 again. In similar way, after the G2 subjects employed the gestUI method they must employ the code-centric method to complete Treatment I.

The data to evaluate PEOU, PU and ITU in this experiment were obtained from the post-task and post-test questionnaires. After the data were gathered, they were checked for correctness and the subjects were consulted when necessary. The data obtained of the aforementioned questionnaires filled in by the subjects are used to measure the response variables defined in Section 6.2.4.

#### **6.2.10 Threats of validity**

In this section we discuss the most important threats to the validity of this evaluation. We have classified the threats according to Wohlin et.al. [138], each of which is discussed below.

**Internal validity:** The main threats to the internal validity of the experiment are:

- (i) *Subject's experience in defining gesture-based interaction:* this threat was resolved since none of the subjects had any experience in tasks related to the topic of custom gesture definition included in the experiment, according to the pre-test questionnaire. So, the subjects' experience in both treatments is the same.

- (ii) *Subject's experience in software development*: there are some factors that can influence the experiment:
  - a. Some of the subjects could have more experience than others in the development of software. Although we used the pre-test questionnaire in order to find out their experience in this field, this threat could not be resolved since we designed the groups in a random way. This threat could affect the evaluation of the effectiveness and the efficiency because the time required to perform the experiment depends on the experience level of the subjects.
  - b. In some cases, subjects without an adequate level of experience in managing source code could produce syntax errors in the source code when inserting the additional source code. This threat could be resolved, since the subjects received adequate information and printed source code without errors included in the Task Description Document with the aim of obtaining a new version of the existing source code of the user interface.
- (iii) *Information exchange among subjects*: this threat was resolved since the experiment was developed in one session, and it was difficult for the subjects to exchange information with each other;
- (iv) *Learning effect*: this threat could not be resolved in both treatments (described in Section 6.2.9) since the process to define custom gestures is identical to the five gestures included in the experiment. Therefore, the definition of the first gesture required more time and effort compared to the following gestures. This threat could affect the evaluation of efficiency and effectiveness because the time needed to perform the experiment depends on the experience level of the subjects.

**External validity:** The main threats to the external validity of the experiment are:

- i. *Duration of the experiment:* there are some factors that can influence the duration of the experiment.
  - a. Since the duration of the experiment was limited to 2 hours, only one interface, six actions (CRUD operations + save the information + open the interface to manage departments) and five gestures were selected. However, repetitive tasks could permit a reduction of time since the subject already knows the process to perform. This threat could not be resolved since these tasks, even though repetitive, were necessary to build the system.
  - b. Since the subjects receive source code that has not been written by them or known before the experiment, then they require time to analyse the structure and the logic of the existing source code before the inclusion of the additional source code. This threat could be resolved by including adequate instructions in the Task Description Document in order to perform the experiment.
  - c. If any subject requires the maximum amount of time to perform the experiment, which is 2 hours (according to what is specified in Section 6.2.6), the information is considered not valid to process because this situation can represent some of the following situations: (i) the subject writes source code slowly using the keyboard and mouse, (ii) a subject does not have the same experience in the use of software tools for software development in relation to other subjects and he/she requires more time to complete the experiment probably performing additional tasks (e.g.

checking if the source code was completely transcribed from the Task Description Document to the Eclipse project, checking for syntax errors in the source code).

- d. Total time required to perform the experiment depends of the typing speed and the experience of the subject in managing source code. This threat could not be resolved in Treatment I (it contains more lines of code to write than Treatment II) since we do not check each subject's typing ability on the computer.
  - e. Time required to check whether the inclusion of the gesture-based interaction was successful varies depending on the experience of the subjects. This threat could be resolved since the subjects answered a question in the pre-test questionnaire about experience in the use of an IDE to develop software in a positive way (43% have an "average" self-rated expertise and 38% have an "experienced" self-rated experience).
- ii. *Representativeness of the results:* despite the fact that the experiment was performed in an academic context, the results could be representative with regard to novice evaluators with no experience in evaluations related with the gesture interaction definition and inclusion. With respect to the use of students as experimental subjects, several authors suggest that the results can be generalised to industrial practitioners [143] [144].

**Construct validity:** The main threat to the construct validity of the experiment is:

- (i) *Type of measurements to consider in the experiment:* measurement that are commonly employed in this type of experiment were used in the quantitative analysis. The



reliability of the questionnaire was tested by applying the Cronbach test, the obtained value is higher than the acceptable minimum (0.70).

**Conclusion validity:** The main threats to the conclusion validity of the experiment are:

- (i) *Validity of the statistical tests applied:* this was resolved by applying Wilcoxon Signed-rank test, one of the most common tests used in the empirical software engineering field. According to Wohlin et al. [138] if we have a sample whose size is less than 30 and we have a factor with two treatments, we can use non-parametric statistical tests such as the Wilcoxon Signed-rank test. In Section 6.2.11 the non-parametric tests used in this experiment are detailed.
- (ii) *Low statistical power:* this happens when the sample size is not large enough. The power of any statistical test is defined as the probability of rejecting a false null hypothesis. According to G\*Power [145] the sample size needed for an effect size of 0.8 is 20 subjects, which is the number of subjects we have. So, this threat has been minimized.

### 6.2.11 Data analysis

The calculated values are checked to see the p-value (significance level). An important issue is the choice of significance level which specifies the probability of the result being representative. Generally speaking, the practice dictates rejecting the null hypothesis when the significance level is less than or equal to 0.05 [139].

The first step is to analyse the reliability of the data obtained in the experiment: we start by calculating the *Cronbach coefficient* (alpha). In this case, the result obtained is 0.736. According to Maxwell [146] if the Cronbach coefficient is greater or equal to 0.7 then the reliability of the data is assumed.

Boone et al. [147] recommend some data analysis procedures for Likert scale data: (a) for central tendency: *mean*, (b) for variability: *standard deviation*, (c) for associations: *Pearson's r*, and (d) other statistics using: *ANOVA*, *t-test*, *regression*. According to Juristo et al. [139], if we have a sample whose size is less than 30 and it follows a normal distribution, then we employ t-distribution (Student's), but if the sample does not follow a normal distribution then we can apply the Wilcoxon Signed-rank test in order to analyse the data obtained in the experiment. A normality test using the Shapiro-Wilk test is required in order to verify if the data is normally distributed. We use this test as our numerical means of assessing normality because it is more appropriate for small sample sizes (< 50 samples). Then, using Shapiro-Wilk we obtained the result that the data is not normally distributed. In this case, we cannot apply the t-distribution test because this test requires normally distributed data. So, we apply the Wilcoxon Signed-rank test.

The next step is verifying whether the data satisfy the sphericity condition and whether they are homogeneous:

- In order to check the *sphericity condition*, *Mauchly's test* can be used. However, in this work, there are only two levels of repeated measures (with the *gestUI* method and with a code-centric method), which precludes a sphericity violation and the test is unnecessary.
- *Non-parametric Levene's test* is used to test if the samples have homogeneity in their variances. In the result of this test we can observe in column "Sig." in Table 38, that the non-parametric Levene's test for homogeneity of variances provides a  $p\_value > 0.05$ , allowing us to assume that the data have homogeneity in their variances.

**Table 38 Non-parametric Levene's test for the variables in the experiment**

Variable	F	df1	df2	Sig.
PEOUg	0.353	1	19	0.560
PEOUc	0.004	1	19	0.948
PUg	0.042	1	19	0.840
PUc	0.754	1	19	0.396
ITUg	0.147	1	19	0.706
ITUc	0.416	1	19	0.527

In Section 6.3, we report the quantitative results of the experiment based on the statistical analysis of the data using (i) descriptive statistics (mainly arithmetic mean), (ii) box-and-whisker plot, (iii) Spearman's Rho correlation coefficient to study the correlation between both treatments, and (iv) the Wilcoxon Signed-rank test with the aim of addressing the research questions. The results of applying Wilcoxon Signed-rank test are described grouped by variables (PTCCI, PTCCG, TFTI, TFTG, PU, PEOU and ITU).

Additionally, at the end of Section 6.3, we include the results of the effect size calculation in order to check the meaningfulness of the results and allow comparison between studies.

A significance level of 0.05 was established to statistically test the obtained results with subjects in the experiment. The analysis has been performed using the SPSS v.23 statistical tool.

## 6.3 Results

In this section, the subscript 'g' located at the end of each variable means "using the gestUI method", and the subscript 'c' means "using the code-centric method". Next, we analyse the results for each research question.

### 6.3.1 RQ1: Effectiveness in the inclusion of gesture-based interaction

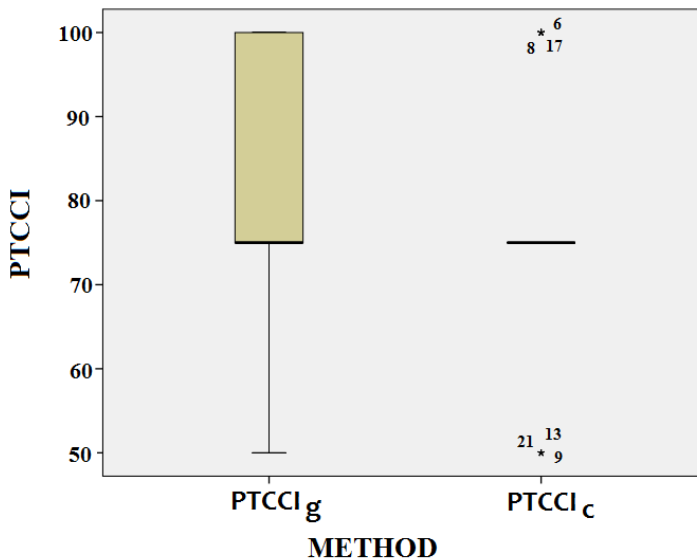
According to Section 6.2.4, in the inclusion of gesture-based interaction, effectiveness (represented by PTCCI) was defined as

the percentage of correctly carried out tasks during the process of inclusion of gesture-based interaction in the user interface. We consider two treatments to analyse PTCCI in the inclusion of gesture-based interaction: PTCCI<sub>g</sub> and PTCCI<sub>c</sub>.

**Table 39 Descriptive statistics for PTCCI**

	N	Min.	Max.	Mean	Std. Dev.
<b>PTCCI<sub>g</sub></b>	21	50	100	82.1429	17.9284
<b>PTCCI<sub>c</sub></b>	21	50	100	77.3810	15.6220
<b>Valid N</b>	21				

According to Table 39, the mean of PTCCI<sub>g</sub> (82.14%) is greater than the mean of PTCCI<sub>c</sub> (77.38%), that is, the subjects achieved a greater percentage of correctly carried out tasks using gestUI than when they employed the code-centric method.



**Figure 60 Box-and-whisker plot of PTCCI**

Figure 60 presents the box-and-whisker plot containing the distribution of the PTCCI variable per method. The medians of PTCCI<sub>g</sub> and PTCCI<sub>c</sub> are similar, but the third quartile is better for PTCCI<sub>g</sub>, since the percentage of correctly carried out tasks achieved

by the subjects using gestUI is greater than the percentage achieved when the subjects use the code-centric method. This means that gestUI is slightly more effective than the code-centric method when the subjects include gesture-based interaction in user interfaces.

Using Spearman's Rho correlation coefficient, we obtained the result shown in Table 40. The samples of PTCClg and PTCClc have a positive correlation (0.638). So, we can conclude that PTCClg and PTCClc are strongly correlated, that is, when the percentage of correctly carried out tasks using gestUI increases, the percentage using the code-centric method also increases.

**Table 40 Spearman's Rho correlation coefficient of PTCCI**  
**Correlations**

		PTCClg	PTCClc
Spearman's rho	Correlation Coefficient	1,000	,638**
	PTCClg Sig. (2-tailed)	.	,002
	N	21	21
	Correlation Coefficient	,638**	1,000
	PTCClc Sig. (2-tailed)	,002	.
	N	21	21

\*\* . Correlation is significant at the 0.01 level (2-tailed)

In order to check whether the observed differences were significant we ran the Wilcoxon Signed-rank test. We obtained the results shown in Table 41 and Table 42.

**Table 41 Wilcoxon Signed-rank test for PTCCI**  
**Ranks**

		N	Mean Rank	Sum of Ranks
PTCClc	Negative Ranks	6 <sup>a</sup>	4,50	27,00
	Positive Ranks	2 <sup>b</sup>	4,50	9,00
	Ties	13 <sup>c</sup>		
PTCClg	Total	21		

a. PTCClc < PTCClg    b. PTCClc > PTCClg    c. PTCClc = PTCClg

**Table 42 Wilcoxon Signed-rank test statistics for PTCCI**  
**Test statistics <sup>a</sup>**

	PTCClc - PTCClg
Z	-1,414 <sup>b</sup>
Asymp. Sig. (2-tailed)	,157

a. Wilcoxon Signed Ranks Test b. Based on positive ranks

They show that two subjects (2/21) have obtained a greater number of correctly carried out tasks using the code-centric method compared to gestUI to include gesture-based interaction in the experiment. Six subjects (6/21) have obtained a greater number of correctly carried out tasks using gestUI compared to the code-centric method. However, thirteen subjects (13/21) have obtained the same number of correctly carried out tasks for both methods.

The 2-tailed p-value obtained with this test was  $p=0.157>0.05$ , therefore, according to this result, we cannot reject the null hypothesis and can conclude that *“There is no difference between the effectiveness of the gestUI and the code-centric methods in the inclusion of gesture-based interaction in user interfaces”*.

### 6.3.2 RQ2: Effectiveness in the definition of custom gestures

According to Section 6.2.4, in the definition of custom gestures, effectiveness (represented by PTCCG) was defined as the percentage of correctly carried out tasks in the custom gesture definition. We consider two treatments to analyse PTCCG in the custom gesture definition: PTCCGg and PTCCGc.

**Table 43 Descriptive statistics for PTCCG**

	N	Min.	Max.	Mean	Std. Dev.
PTCCGg	21	75	100	91.6667	12.0762
PTCCGc	21	25	100	71.4286	19.8206
<b>Valid N</b>	21				

According to Table 43, the mean of PTCCGc (71.43%) is less than the mean of PTCCGg (91.67%), that is, the subjects achieved a relatively

greater percentage of correctly carried out tasks using gestUI than when they employed the code-centric method.

Figure 61 presents the box-and-whisker plot containing the distribution of the PTCCG variable per method. The median, the first quartile and the third quartile are better for PTCCGg, since it achieved a greater percentage of correctly carried out tasks. This means that gestUI was more effective than the code-centric method when the subjects define custom gestures.

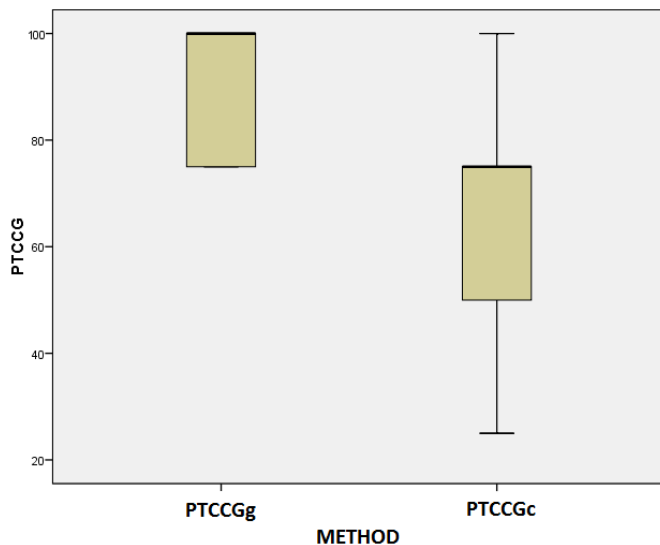


Figure 61 Box-plot-whisker of PTCCG

Using Spearman's Rho correlation coefficient we obtained the results shown in Table 44. The samples of PTCCG have a positive correlation (0.456). Then, we can conclude that PTCCGg and PTCCGc have a moderate correlation, that is, when the percentage of correctly carried out tasks with PTCCGg increases, there is a moderate increment in the percentage of PTCCGc.

**Table 44 Spearman's Rho correlation coefficient of PTCCG Correlations**

		PTCCGg	PTCCGc	
Spearman's rho	PTCCGg	Correlation Coefficient	1,000	,456*
		Sig. (2-tailed)	.	,038
		N	21	21
	PTCCGc	Correlation Coefficient	,456*	1,000
		Sig. (2-tailed)	,038	.
		N	21	21

\*. Correlation is significant at the 0.05 level (2-tailed)

In order to check whether the observed differences were significant we ran the Wilcoxon Signed-rank test. We obtained the results shown in Table 45 and Table 46.

**Table 45 Wilcoxon Signed-rank test for PTCCG Ranks**

		N	Mean Rank	Sum of Ranks
PTCCGc	Negative Ranks	14 <sup>a</sup>	7,50	105,00
	Positive Ranks	0 <sup>b</sup>	0,00	,00
-	Ties	7 <sup>c</sup>		
PTCCGg	Total	21		

a. PTCCGc < PTCCGg    b. PTCCGc > PTCCGg    c. PTCCGc = PTCCGg

**Table 46 Wilcoxon Signed-rank test statistics for PTCCG Test Statistics <sup>a</sup>**

PTCCGc - PTCCGg	
Z	-3,556 <sup>b</sup>
Asymp. Sig. (2-tailed)	,000

a. Wilcoxon Signed Ranks Test  
b. Based on positive ranks

It shows that fourteen subjects (14/21) have obtained more correctly carried out tasks using gestUI compared to using the code-centric method, zero (0/21) subjects have obtained more correctly carried out tasks using the code-centric method than using gestUI, and there are



seven (7/21) subjects that have obtained the same percentage using both methods.

The 2-tailed p-value obtained with this test was  $p=0.000<0.05$ , therefore, according to this result, we reject the null hypothesis and can conclude that *“gestUI is more effective than the code-centric method in the definition of custom gestures”*.

### 6.3.3 RQ3: Efficiency in the inclusion of gesture-based interaction

According to Section 6.2.4, efficiency (represented by TFTI) was defined as the time to finish the task during the inclusion of gesture-based interaction in the user interface. We consider two treatments to analyse TFTI in the inclusion of gesture-based interaction: TFTI<sub>g</sub> and TFTI<sub>c</sub>.

According to Table 47, the mean of TFTI<sub>c</sub> (28.38) is greater than that of TFTI<sub>g</sub> (19.71), that is, the time required to include gesture-based interaction in the experiment using the code-centric method is greater than the time needed to perform this task using gestUI.

**Table 47 Descriptive statistics for TFTI**

	N	Min.	Max.	Mean	Std. Dev.
<b>TFTI<sub>g</sub></b>	21	9.00	33.00	19.7143	7.0224
<b>TFTI<sub>c</sub></b>	21	18.00	49.00	28.3810	7.8834
<b>Valid N</b>	21				

Figure 62 presents the box-and-whisker plot containing the distribution of the TFTI variable per method. The medians, first quartile and third quartile are better for TFTI<sub>g</sub>, since the time needed to conduct the experiment is less when the subjects use gestUI rather than when the subjects use the code-centric method. This means that the time to finish the task with gestUI is better than with code-centric.

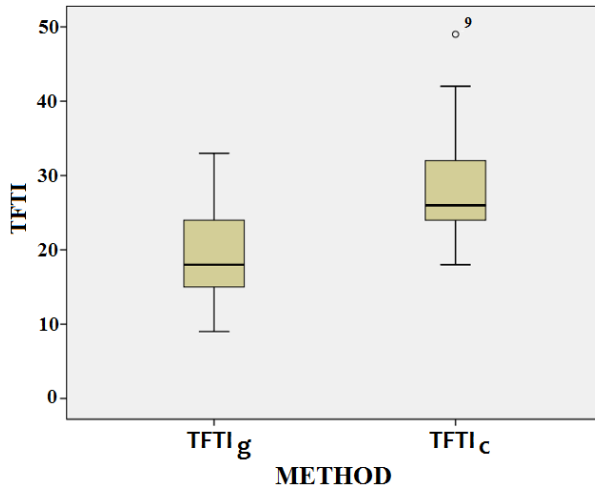


Figure 62 Box-plot for TFTI

Using Spearman’s Rho correlation coefficient we obtained the results shown in Table 48. The samples of TFTI have a positive correlation (0.210). Then, we can conclude that TFTI<sub>g</sub> and TFTI<sub>c</sub> have a weak correlation, that is, between TFTI<sub>g</sub> and TFTI<sub>c</sub> there is not a significant relationship (Sig. (2-tailed)>0.05) in the process of including gesture-based interaction.

Table 48 Spearman's Rho correlation coefficient of TFTI Correlations

		TFTI <sub>g</sub>	TFTI <sub>c</sub>
Spearman's rho	TFTI <sub>g</sub>	Correlation Coefficient	1,000
		Sig. (2-tailed)	,210
		N	,361
	TFTI <sub>c</sub>	Correlation Coefficient	21
		Sig. (2-tailed)	21
		N	,210
		,361	1,000
		,210	,361
		21	21

In order to check whether the observed differences were significant we ran the Wilcoxon Signed-rank test. We obtained the results shown in Table 49 and Table 50. They show that eighteen subjects (18/21) have employed more time using the code-centric method compared to gestUI to include gesture-based interaction in the experiment. Three subjects (3/21) have employed less time using the code-centric

method than gestUI to include gesture-based interaction in the experiment.

**Table 49 Wilcoxon Signed-rank test for TFTI Ranks**

		N	Mean Rank	Sum of Ranks
TFTIc -	Negative Ranks	3 <sup>a</sup>	7,17	21,50
TFTIg	Positive Ranks	18 <sup>b</sup>	11,64	209,50
	Ties	0 <sup>c</sup>		
	Total	21		

a. TFTIc < TFTIg    b. TFTIc > TFTIg    c. TFTIc = TFTIg

**Table 50 Wilcoxon Signed-rank test statistics for TFTI Test Statistics <sup>a</sup>**

	TFTIc - TFTIg
Z	-3,269 <sup>b</sup>
Asymp. Sig. (2-tailed)	,001

a. Wilcoxon Signed Rank Test

b. Based on negative ranks

The 2-tailed p-value obtained with this test was  $p=0.001 < 0.05$ , therefore, according to this result, we reject the null hypothesis and we can conclude that *“gestUI is more efficient than the code-centric method in the inclusion of gesture-based interaction in user interfaces”*.

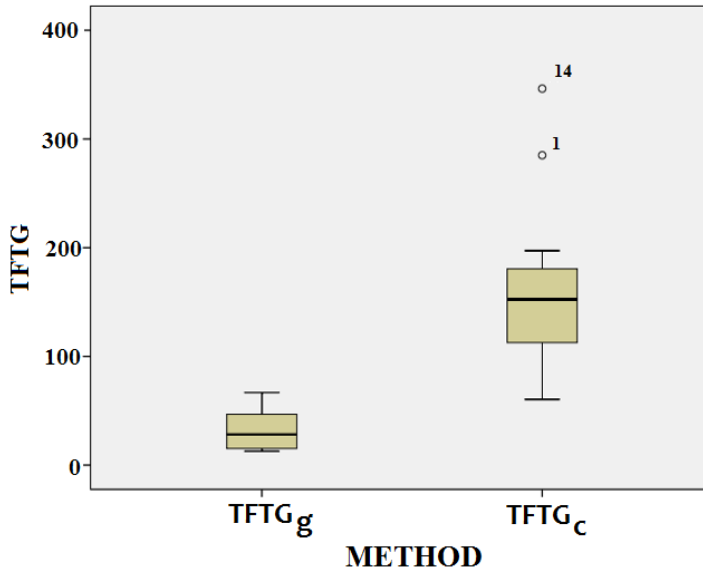
#### **6.3.4 RQ4: Efficiency in the definition of custom gestures**

According to Section 6.2.4, efficiency (represented by TFTG) was defined as the time to finish the task during the custom gesture definition. We consider two treatments to analyse TFTG in the definition of custom gestures: TFTG<sub>g</sub> and TFTG<sub>c</sub>.

According to Table 51, the mean of TFTG<sub>c</sub> (154.67) is greater than the mean of TFTG<sub>g</sub> (31.89), which means that the time required to define custom gestures in the experiment using the code-centric method is greater than the time to do this task using gestUI.

**Table 51 Descriptive statistics for TFTG**

	N	Min.	Max.	Mean	Std. Dev.
<b>TFTGg</b>	21	12.75	66.75	31.8929	16.8301
<b>TFTGc</b>	21	60.50	346.25	154.6786	66.5967
<b>Valid N</b>	21				



**Figure 63 Box-plot of TFTG**

Figure 63 presents the box-and-whisker plot containing the distribution of the TFTG variable per method. The median, first quartile and third quartile are better for TFTGg, since TFTGg needs less time to complete the task. This means that gestUI was more efficient than code-centric method regarding the time required by the subject to define custom gestures during the experiment.

Using Spearman's Rho correlation coefficient, we obtained the result shown in Table 52. The samples of TFTG have a positive correlation (0.216). Then, we can conclude that TFTGg and TFTGc have a weak correlation, that is, when the time required to define custom gestures using code-centric method increases, the time using gestUI method also has a weak increment.

In order to check whether the observed differences were significant, we run Wilcoxon Signed-rank test. We obtain the results shown in Table 53 and Table 54. It shows that twenty-one subjects (21/21) have employed more time using the code-centric method than gestUI to define custom gestures in the experiment.

**Table 52 Spearman's Rho correlation coefficient of TFTG Correlations**

			TFTGg	TFTGc
Spearman's rho	TFTGg	Correlation	1,000	,216
		Coefficient	21	,346
		Sig. (2-tailed)		21
		N		
	TFTGc	Correlation	,216	1,000
		Coefficient	,346	.
		Sig. (2-tailed)	21	21
		N		

The 2-tailed p-value obtained with this test was  $p=0.000 < 0.05$ , therefore, according to this result, we reject the null hypothesis and we can conclude that *“When the subjects define gestures, gestUI is more efficient than the code-centric method”*.

**Table 53 Wilcoxon Signed-rank test for TFTG Ranks**

		N	Mean Rank	Sum of Ranks
TFTGc - TFTGg	Negative Ranks	0 <sup>a</sup>	,00	,00
	Positive Ranks	21 <sup>b</sup>	11,00	231,00
	Ties	0 <sup>c</sup>		
	Total	21		

a. TFTGc < TFTGg    b. TFTGc > TFTGg    c. TFTGc = TFTGg

**Table 54 Wilcoxon Signed-rank test statistics for TFTG Test Statistics<sup>a</sup>**

	TFTG <sub>c</sub> – TFTG <sub>g</sub>
Z	-4,015 <sup>b</sup>
Asymp. Sig. (2-tailed)	,000

- a. Wilcoxon Signed Ranks Test
- b. Based on negative ranks

### 6.3.5 RQ5: Perceived Ease of Use

According to Section 6.2.4, the variable PEOU is defined as perceived ease of use of the method. We consider two treatments to analyse PEOU: PEOU<sub>g</sub> and PEOU<sub>c</sub>.

Table 55 presents the results obtained through questions related to PEOU within Post-task and Post-test questionnaires. In this case, the mean is above 3.0 in both cases. There is a difference of 0.042 between the mean of PEOU<sub>c</sub> and the mean of PEOU<sub>g</sub>, that is, the PEOU of gestUI is relatively greater than the PEOU of the code-centric method.

**Table 55 Descriptive statistics for PEOU**

	N	Min.	Max.	Mean	Std. Dev.
<b>PEOU<sub>g</sub></b>	21	1	5	3.2857	0.2154
<b>PEOU<sub>c</sub></b>	21	1	5	3.3280	0.5073
<b>Valid N</b>	21				

**Table 56 Spearman's Rho correlation coefficient of PEOU Correlations**

			PEOU <sub>g</sub>	PEOU <sub>c</sub>
Spearman's rho	PEOU <sub>g</sub>	Correlation coefficient	1,000	,408
		Sig. (2-tailed)	.	,066
		N	21	21
	PEOU <sub>c</sub>	Correlation coefficient	,408	1,000
		Sig. (2-tailed)	,066	.
		N	21	21

Using Spearman's Rho correlation coefficient we obtain the next result (Table 56). The samples of PEOU have a positive correlation

(0.408). So, we can conclude PEOU<sub>g</sub> and PEOU<sub>c</sub> have a moderate correlation, that is, when the perceived ease of use with gestUI increases, PEOU using the code-centric method also increases.

Figure 64 shows the box-and-whisker plot containing the distribution of the PEOU variable per method. The medians of both treatments are the same. The first quartile is slightly better for gestUI and the third quartile is slightly better for the code-centric method. This means that there are no differences between both treatments.

In order to check whether the observed differences were significant, we ran the Wilcoxon Signed-rank obtaining the results shown in Table 57 and Table 58. They show that eight subjects (8/21) perceive that gestUI is easier to use than the code-centric method, eight subjects (8/21) perceive than the code-centric method is easier to use than gestUI and, five (5/21) perceive that both methods are easy to use.

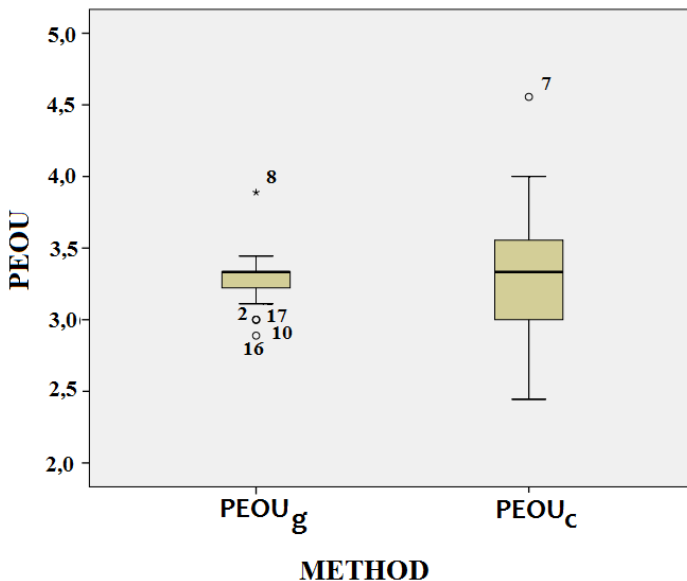


Figure 64 Box-plot for PEOU

**Table 57 Wilcoxon Signed-rank test for PEOU Ranks**

		N	Mean Rank	Sum of Ranks
PEOUc - PEOUg	Negative Ranks	8 <sup>a</sup>	8,25	66,00
	Positive Ranks	8 <sup>b</sup>	8,75	70,00
	Ties	5 <sup>c</sup>		
	Total	21		

---

a. PEOUc < PEOUg    b. PEOUc > PEOUg    c. PEOUc = PEOUg

**Table 58 Wilcoxon Signed-rank test statistics for PEOU Test Statistics<sup>a</sup>**

	PEOUc – PEOUg
Z	-,104b
Asymp. Sig. (2-tailed)	,917

- a. Wilcoxon Signed Ranks Test  
b. Based on negative ranks

The 2-tailed p-value obtained with this test was  $p=0.917 > 0.05$ , therefore, according to this result, we cannot reject the null hypothesis and we can conclude that *“gestUI is perceived as easier to use than the code-centric method”*.

### 6.3.6 RQ6: Perceived Usefulness

According to Section 6.2.4, the variable PU is defined as perceived usefulness of the method. We consider two treatments to analyse perceived usefulness: PUg and PUC.

Table 59 presents the results obtained through questions related to PU in Post-task and Post-test questionnaires. In this case, the mean of PUC is less than PUg, that is, perceived usefulness of gestUI (mean=3.82) is greater than the perceived usefulness of the code-centric method (mean=3.28).

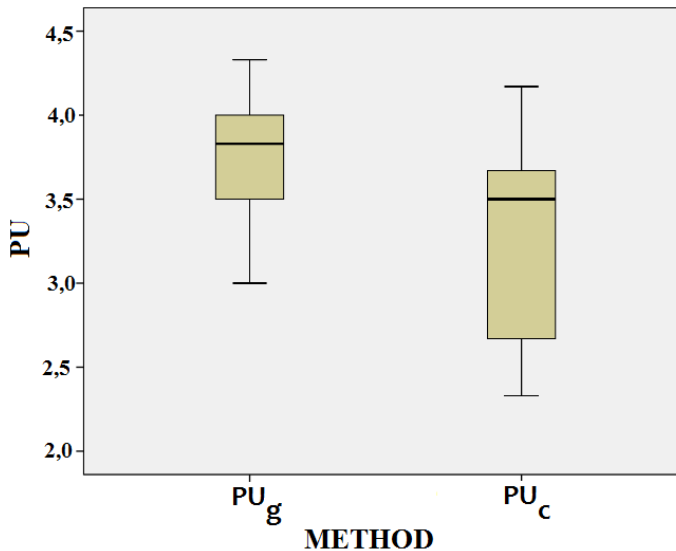


**Table 59 Descriptive statistics for PU**

	N	Min.	Max.	Mean	Std. Dev.
<b>PU<sub>g</sub></b>	21	1	5	3.8176	0.3451
<b>PU<sub>c</sub></b>	21	1	5	3.2786	0.5762
<b>Valid N</b>	21				

Figure 65 presents the box-and-whisker plot containing the distribution of the PU variable per method. The median, first quartile and third quartile of PU<sub>g</sub> is better than PU<sub>c</sub>. This means that the subjects perceived gestUI to be more useful than the code-centric method.

Using Spearman's Rho correlation coefficient, we obtain the next result (Table 60). The samples of PU have a positive correlation (0.310). So, we can conclude that PU<sub>g</sub> and PU<sub>c</sub> have a weak correlation, that is, when the perceived usefulness of the code-centric method increases, the perceived usefulness using the gestUI method also increases.



**Figure 65 Box-plot of PU**

**Table 60 Spearman's Rho correlation coefficient of PU Correlations**

		PUg	PUc
Spearman's rho	PUg	Correlation Coefficient	,310
		Sig. (2-tailed)	,172
		N	21
	PUc	Correlation Coefficient	,310
		Sig. (2-tailed)	,172
		N	21

In order to check whether the observed differences were significant, we ran the Wilcoxon Signed-rank obtaining the results shown in Table 61 and Table 62. This test shows that fifteen subjects (15/21) perceive gestUI to be more useful than the code-centric method in the experiment. Three subjects (3/21) perceive the code-centric method to be more useful than gestUI, and three (3/21) consider that both methods have the same level of perceived usefulness in the experiment.

**Table 61 Wilcoxon Signed-rank test for PU Ranks**

		N	Mean Rank	Sum of Ranks
PUc - PUg	Negative Ranks	15 <sup>a</sup>	10,63	159,50
	Positive Ranks	3 <sup>b</sup>	3,83	11,50
	Ties	3 <sup>c</sup>		
	Total	21		

a. PUc < PUg    b. PUc > PUg    c. PUc = PUg

The 2-tailed p-value obtained with this test was  $p=0.001 < 0.05$ , therefore, according to this result, we reject the null hypothesis and we can conclude that *“gestUI is perceived as more useful than the code-centric method”*.

**Table 62 Wilcoxon Signed-rank test statistics for PU  
Test Statistics<sup>a</sup>**

PUC – PUG	
Z	-3,239 <sup>b</sup>
Asymp. Sig. (2-tailed)	,001

a. Wilcoxon Signed Ranks Test

b. Based on positive ranks

### 6.3.7 RQ7: Intention to Use

According to Section 4.4, the variable ITU is defined as the intention to use of the method. We consider two treatments to analyse ITU: ITUg and ITUc.

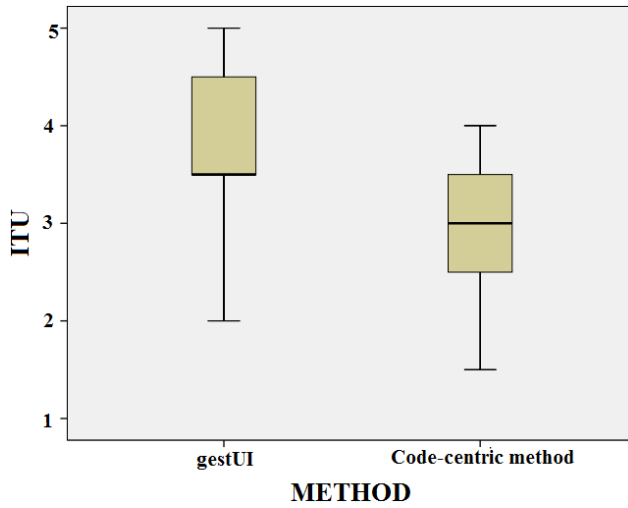
Table 63 presents the results obtained through questions related to ITU in Post-test and Post-task questionnaires. In this case, the mean of ITUg (3.74) is above 3.0 while the mean of ITUc (2.93) is below to 3.0.

**Table 63 Descriptive statistics for ITU**

	N	Min.	Max.	Mean	Std. Dev.
<b>ITUg</b>	21	2	5	3.7381	0.7179
<b>ITUc</b>	21	1	4	2.9286	0.6761
<b>Valid N</b>	21				

Figure 66 presents the box-and-whisker plot containing the distribution of the ITU variable per method. The median, the first and third quartile are better for ITUg. This means that gestUI has a greater intention to use than the code-centric method when the subjects use it to define custom gestures and to include gesture-based interaction.

Using Spearman's Rho correlation coefficient we obtain the next result (Table 64). The samples of ITU have a positive correlation (0.080). So, we can conclude that ITUg and ITUc have a very weak correlation, that is, when the intention to use of gestUI (ITUg) increases, the intention to use of the code-centric method (ITUc) increases very little compared with ITUg.



**Figure 66** Box-plot of ITU

**Table 64** Spearman's Rho correlation coefficient of ITU Correlations

			ITUg	ITUc
Spearman's rho	ITUg	Correlation Coefficient	1,000	,080
		Sig. (2-tailed)	.	,731
		N	21	21
	ITUc	Correlation Coefficient	,080	1,000
		Sig. (2-tailed)	,731	.
		N	21	21

**Table 65** Wilcoxon Signed-rank test for ITU Ranks

		N	Mean Rank	Sum of Ranks
ITUc – ITUg	Negative Ranks	13 <sup>a</sup>	8,65	112,50
	Positive Ranks	2 <sup>b</sup>	3,75	7,50
	Ties	6 <sup>c</sup>		
	Total	21		

a. ITUc < ITUg

b. ITUc > ITUg

c. ITUc = ITUg

In order to check whether the observed differences were significant, we ran the Wilcoxon Signed-rank obtaining the results included in

Table 65 and Table 66. They show that gestUI has greater intention to use than the code-centric method (13/21 subjects), the code-centric method has two (2/21) subjects with intention to use, and six (6/21) subjects have an intention to use for both methods.

**Table 66 Wilcoxon Signed-rank test statistics for ITU Test Statistics<sup>a</sup>**

	ITUc – ITUg
Z	-3,005 <sup>b</sup>
Asymp. Sig. (2-tailed)	,003

a. Wilcoxon Signed Ranks Test  
b. Based on positive ranks

The 2-tailed p-value obtained with this test was  $p=0.003 < 0.05$ , therefore, according to this result, we reject the null hypothesis, and we can conclude that “*gestUI has an intention to use greater than the code-centric method*”.

In summary, the result of each hypothesis is shown in Table 67.

### 6.3.8 Effect-size calculation

According to Kotrlik [148], effect size measures focus on the meaningfulness of the results and allow comparison between studies, furthering the ability of researchers to judge the practical significance of results presented. We use means and standard deviations of the metrics defined in this experiment to calculate Cohen’s d and effect-size correlation r. The calculation was performed using the effect size calculator provided by the University of Colorado (Colorado Springs), available at <http://www.uccs.edu/~lbecker/>.

**Table 67 Summary of the results obtained in the experiment**

<b>Variable</b>	<b>Null hypothesis status</b>	<b>Conclusion</b>
PTCCI	Not rejected	There is no significant difference between the effectiveness of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces.
PTCCG	Rejected	There is a significant difference between the effectiveness of gestUI and the code-centric method in the specification of custom gestures. Results obtained are better when the subjects use gestUI rather than the code-centric method, that is, PTCCGg is greater than PTCCGc.
TFTI	Rejected	There is a significant difference between the efficiency of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces. The results obtained are less when the subjects use gestUI rather than when they use the code-centric method.
TFTG	Rejected	When the subjects define gestures, gestUI is more efficient than the code-centric method.
PEOU	Not rejected	gestUI is perceived as easier to use than the code-centric method.
PU	Rejected	gestUI is perceived as more useful than the code-centric method.
ITU	Rejected	gestUI has an intention to use greater than the code-centric method

Based on the work of Lakens [149], we can see that the effect size is “Large” if  $d > 0.8$ , “Medium” if  $d \leq 0.5$  and  $d > 0.2$ , and “Small” if  $d < 0.2$ . In Table 68, we present the results of the effect size calculation of the metrics included in this experiment and this shows the equivalences applied to the results obtained.

**Table 68 Effect size of the metrics**

Response variable	Metric	Mean	St. Dev.	Cohen’s d	Equivalence
Effectiveness in the inclusion of gesture-based interaction.	PTCCI				
	PTCClg	82.1429	17.9284	0.2832	Medium
	PTCClc	77.3810	15.6220		
Effectiveness in the custom gesture definition.	PTCCG				
	PTCCGg	91.667	12.076	1.233	Large
	PTCCGc	71.428	19.821		
Efficiency in the inclusion of gesture-based interaction.	TFTI				
	TFTlg	19.714	7.022	1.161	Large
	TFTlc	28.381	7.883		
Efficiency in the custom gesture definition.	TFTG				
	TFTGg	31.893	154.678	2.5279	Large
	TFTGc	16.8301	66.5967		
Satisfaction	PU				
	PUg	3.8176	0.3451	1.1349	Large
	PUc	3.2786	0.5762		
PEOU					
	PEOUg	3.2857	0.2154	0.1085	Small
	PEOUc	3.3280	0.5073		
Satisfaction	ITU				
	ITUg	3.7381	0.7179	1.1609	Large
	ITUc	2.9286	0.6761		

According to this classification, the results obtained for effect size show that:

- (i) In the case of PTCCG, TFTI, TFTG, PU and ITU, the effect size calculated through Cohen’s d is greater than 0.8, which means that it is classified as “Large”. So, there is a significant

difference in the application of each method in this experiment related to: effectiveness in the definition of custom gestures (PTCCG), efficiency in the inclusion of gesture-based interaction (TFTI), efficiency in the definition of custom gesture (TFTG), perceived usefulness (PU) and intention to use (ITU).

- (ii) In the case of PTCCI, the effect size calculated through Cohen's  $d$  is equals to 0.2832 ( $d > 0.2$ ), which is classified as "Medium". So, the difference in the application of each method to include gesture-based interaction in a user interface considering the effectiveness in the inclusion of gesture-based interaction, is not important.
- (iii) In the case of PEOU, the effect size calculated through Cohen's  $d$  is less than 0.2 ( $d = 0.1085$ ), which is classified as "Small". So, there is a minimum difference in the application of each method in this experiment related to the perceived ease of use (PEOU).

In the next section, we analyse the results obtained in this experiment.

## 6.4 Discussion

In this section, we discuss the results of the experiment described in Section 6.3 in order to draw some conclusions regarding the comparison of gestUI (a model-driven method) and the code-centric method (traditional software development). In order to validate gestUI, three aspects are considered in this experiment: effectiveness (using PTCCI, PTCCG), efficiency (using TFTI and TFTG) and satisfaction (using PU, PEOU and ITU). The discussion about the results obtained in the experiment is performed according to the aforementioned research questions.

### 6.4.1 Effectiveness

RQ1: Effectiveness in the inclusion of gesture-based interaction

RQ1 is related to the PTCCI metric that is defined as the **percentage of task correctly carried out in the inclusion of gesture-based**



**interaction** in a user interface. Regarding PTCCI, the results obtained by applying statistical tests show that:

- Through the Wilcoxon Signed-rank test, there is no significant difference between the results obtained when the subjects applied gestUI and when the subjects applied the code-centric method to include gesture-based interaction in an existing user interface. We consider that the small difference obtained (approximately 4%) by applying both methods to calculate PTCCI is because (i) the subjects used existing source code (included in the Task Description Document) instead of writing the source code from scratch as is done in a typical development process [119]. This context helped to obtain better results with the code-centric method and the difference was less than expected; (ii) the subjects were not familiar with the process defined in gestUI to apply a model-driven method (i.e. by using model transformations to include gesture-based interaction); (iii) the subjects did not have experience in the inclusion of gesture-based interaction and when they applied gestUI, the process was not very intuitive to follow.

#### RQ2: Effectiveness in the definition of custom gestures

RQ2 is related with the PTCCG metric that is defined as the **percentage of task correctly carried out in the custom gesture definition** (PTCCG). Values obtained applying statistic tests for PTCCG show that:

- Through the Wilcoxon Signed-rank test we found that gestUI is significantly more effective than the code-centric method in the definition of custom gestures. The percentage obtained with gestUI is greater than the percentage obtained with the code-centric method. In this case, the difference between the percentage of task correctly carried out in the custom gestures definition using gestUI or using the code-centric method is almost 20%. This difference is due to subjects using gestUI having a more intuitive process to follow to define gestures and to obtain a XML file containing the description of the gesture. Using the code-

centric method, the process of defining gestures is more complex because it includes additional tasks (e.g. analyse the shape of the gesture, draw it and define it using XML, among others) requiring more effort.

#### 6.4.2 Efficiency

##### RQ3: Efficiency in the inclusion of gesture-based interaction

RQ3 is related with the TFTI metric that is defined as the **time to finish the task during the inclusion of gesture-based interaction** in the user interface. Values obtained for TFTI show that:

- Through the Wilcoxon Signed-rank test we found that gestUI is significantly more efficient than the code-centric method in the inclusion of gesture-based interaction in user interfaces. When the subjects did the experiment using gestUI, they required less time than when they used the code-centric method. The difference of time between both methods is moderate (8.67 min.), this could be related to the ability to type the source code in a correct way, probably because the subjects had experience developing software (according to the demographic questionnaire, the average self-rated programming expertise was 43%). Also, they required less time to type source code since they had experience using the integrated development environment used in the experiment (according to the demographic questionnaire 38% had an “experienced” level and 43% had a “medium experienced” level with Eclipse Framework).

##### RQ4: Efficiency in the definition of custom gestures

RQ4 is related with TFTG that is defined as the **time to finish the task during the custom gesture definition** (TFTG). Obtained results show that:

- Through the Wilcoxon Signed-rank test we found that the time required to define custom gestures using gestUI is less than the time required using the code-centric method. The difference is high (122.7857) since some subjects had some problems with the

definition of gestures using XML language as they were not familiar with the syntax of XML. Another aspect that could have increased the time required with the code-centric method is related to syntax errors generated during the process of gesture definition. If the subjects run the experiment first with gestUI and then with the code-centric method, they require a longer time than those subjects that run the experiment first with the code-centric method and then with gestUI. In this case, there were some problems when the subjects employed \$N to recognise some gestures sketched by them. This could have had some influence in the duration of the process of custom gesture definition.

In summary, regarding effectiveness and efficiency, we can say that:

- The result obtained in the experiment permit one to say, in general, that the effectiveness and efficiency of gestUI are greater than those of the code-centric method.
- Considering the metrics PTCCG, TFTG and TFTI, the results obtained with Cohen's d value ( $d > 0.8$ , i.e. "Large") suggest a high practical significance for the results obtained. Also, Cohen's d value ( $d = 0.2832$  for PTCCI) suggested a moderate practical significance for the results obtained.
- Concerning the values of TFTG and TFTI obtained in the experiment, we think that if the subjects had written the source code from scratch, the difference in time would have been greater. In general, the overall results lead us to interpret that gestUI has achieved better effectiveness and efficiency for the subjects in almost all the analysed statistics in comparison with the code-centric method.
- Finally, considering effect size, we can conclude that in comparison, effectiveness and efficiency of gestUI are better than those obtained with the code-centric method in the custom gesture definition.

### 6.4.3 Satisfaction

#### RQ5: Perceived ease of use

RQ5 is related with PEOU that is defined as **perceived ease of use** (PEOU). Obtained results show that:

- Through the Wilcoxon Signed-rank test we found that the difference between PEOUg (3.286) and PEOUc (3.328) is minimal (0.0423). So, we can say that the subjects perceive that both methods are easy to use. However, in the case of the code-centric method, this result could be influenced by the inclusion of source code in the Task Description Document as was explained in Section 6.2.8. This decision was taken with the aim of reducing the complexity of the code-centric method and the time required to do the experiment.

#### RQ6: Perceived Usefulness

RQ6 is related with PU that is defined as **perceived usefulness** (PU). Obtained results show that:

- Through the Wilcoxon Signed-rank test we found that there is difference (0.539) between the values of PUg (3.8176) and PUc (3.2786). So, we can say that the subjects perceive gestUI to be more useful than the code-centric method. The subjects perceive the usefulness of gestUI by noting that if gestUI is easy to use they may find gestUI more useful, and hence, have some motivation to use it. Specifically, the subjects perceive the usefulness of gestUI when they use it to automatically obtain source code to include gesture-based interaction in a user interface based on a specification of gestures and actions to define the gesture-based interaction.

#### RQ7: Intention to use

RQ7 is related with ITU that is defined as **intention of use**. Obtained results show that:

- Through the Wilcoxon Signed-rank test we found that there is a difference (0.8095) between the values of ITUg (3.7381) and ITUc (2.9286). So, we can say that the subjects have an intention to use gestUI greater than the code-centric method. This conclusion is based on the fact that the subjects considered gestUI as easy to use and useful compared to the code-centric method.

In general, the results of our work indicate that gestUI is accepted by the subjects since the results obtained for effectiveness, efficiency and satisfaction with gestUI are better than the results obtained with the code-centric method. With these results we could say that gestUI is a hopeful approach and justifies further investigation.

## 6.5 Conclusions

This validation process compares a model-driven method (gestUI) versus a traditional software development method (the code-centric method) in terms of (i) effectiveness in the custom gesture definition, (ii) effectiveness in the inclusion of gesture-based interaction, (iii) efficiency in the custom gesture definition, (iv) effectiveness in the inclusion of gesture-based interaction,) and satisfaction (PEOU, PU and ITU) through an experimental investigation. Results show that, in general, gestUI has a greater effectiveness, efficiency and satisfaction level than the code-centric method, and gestUI was also perceived by the subjects as easier to use than the code-centric method.

Some aspects that must be contextualised according to the type of experiment are:

- (i) The sample size is small, twenty-one (21) subjects.
- (ii) The subjects were M.Sc. and Ph.D. students and they do not have enough experience in the topics included in the experiment: tasks related with the custom gesture definition and the inclusion of gesture-based interaction.

- (iii) The subjects have experience in software development using the Java programming language, which could have influenced the results obtained with the code-centric method.
- (iv) We consider that the decision to include source code in the Task Description Document to reduce the time for the code-centric method has reduced the differences in terms of efficiency between treatments, since subjects only had to transcribe the source code specified in the document.

Gesture definition is interesting for the subjects since they can specify their own gestures with the aim of executing actions in a user interface. In this context, each subject defined four gestures in order to use them in the user interface doing CRUD operations in a database. The subjects could define their own gestures according to their preferences.

Even though the experimental results are good for the usefulness of gestUI, we are aware that more experimentation is needed to confirm these results. Existing results must be interpreted within the context of this experiment. In general, the subjects considered gestUI a good solution since they defined custom gestures and they included the gestures in the user interface in a short time compared to the time required when they used the code-centric method



---

TECHNICAL  
ACTION  
RESEARCH

7

The topics covered in this chapter are:

- 7.1 Introduction
- 7.2 Capability Design Tool
- 7.3 Validation using Technical Action Research
- 7.4 Action Research Procedure
- 7.4 Analysis and Interpretation of results
- 7.5 Threats to validity
- 7.6 Conclusions





## Chapter 7. Technical Action Research

### 7.1 Introduction

In this chapter, we describe the validation of gestUI through Technical Action Research (TAR). We conduct this evaluation with the purpose of knowing the experience of subjects in the industry when they apply gestUI in a tool they use to carry out their activities. This evaluation is complementary to that described in Chapter 6.

TAR can be seen as a research method that starts from the opposite side of traditional research methods. TAR starts with an artefact, and then tests it under practical conditions by using it to solve concrete problems [30].

According to Wieringa [150], TAR is related with the use of an experimental artefact to help a client and to learn about its effects in practice. The artefact is experimental, which means that it is still under development and has not yet been transferred to the original problem context. In a validation process with TAR, the researcher uses an artefact (e.g. method and a tool) in a real-world project to help a client, or gives the artefact to others so they can use it assisted by the researcher [150].

In this chapter, we report the validation of gestUI in real-world conditions through TAR. During the validation process, we aimed to discover just how gestUI can help stakeholders (e.g. software engineers, end-users) to define custom gestures and to include gesture-based interaction in existing user interfaces. We also aimed to obtain practical interpretations of the system from industry practitioners. We use TAR in the context of the CaaS Project (FP7 ICT Programme Collaborative Project no. 611351). The main outcomes of CaaS are: (i) the Capability-Driven Development (CDD) methodology [151] and (ii) the CDD environment. The Capability Design Tool is a CASE tool in the CDD environment that supports capability modelling according to the CDD meta-model [152]. Everis, a multinational firm

offering business consulting, as well as development, maintenance and improved information technology, collaborated in the evaluation. Everis's CaaS-project team is developing an e-government platform by applying the whole CDD methodology and environment.

We report on a user evaluation that involves business consultants using gestUI to include gesture-based interaction in a user interface and then carrying out a modelling task by means of gestures. We base the empirical validation on well-known frameworks and techniques, such as:

- (i) The Method Evaluation Model (MEM) [133] to relate subjects' performance, perceptions and intentions. MEM is described in Chapter 6.
- (ii) The User Experience Questionnaire (UEQ) [153] to measure user experience with gestUI.
- (iii) Microsoft Reaction Cards (MRC) to obtain desirability level and user experience [154] with gestUI.

The main goal of the User Experience Questionnaire (UEQ) is to obtain a fast and immediate measurement of user experience of interactive products [155]. The questionnaire format supports the user response to immediately express feelings, impressions, and attitudes that arise when they use a product [156]. The questionnaire consists of bipolar contrasting attributes on a seven-scale ranking. Figure 67 shows an excerpt of the user experience questionnaire.

The subjects express their agreement with the attributes by ticking the circle that most closely reflects their impression. The seven-scale ranking is converted into a positive and a negative scale, where +3 represents the most positive and the -3 represents the most negative value [157]. The user experience questionnaire contains six scales with 26 items in total: attractiveness, efficiency, perspicuity, dependability, stimulation, novelty [158].

	1	2	3	4	5	6	7		
annoying	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	enjoyable	1
not understandable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	understandable	2
creative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	dull	3
easy to learn	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	difficult to learn	4
valuable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	inferior	5
boring	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	exciting	6
not interesting	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	interesting	7
unpredictable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	predictable	8
fast	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	slow	9
inventive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	conventional	10
obstructive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	supportive	11
good	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	bad	12
complicated	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	easy	13
unlikable	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	pleasing	14
usual	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	leading edge	15
unpleasant	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	pleasant	16
secure	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	not secure	17
motivating	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	demotivating	18
meets expectations	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	does not meet expectations	19
inefficient	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	efficient	20
clear	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	confusing	21
impractical	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	practical	22
organized	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	cluttered	23
attractive	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unattractive	24
friendly	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	unfriendly	25
conservative	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	innovative	26

**Figure 67. An excerpt of User Experience Questionnaire (taken of [www.ueq-online.org](http://www.ueq-online.org))**

Product reaction cards (PRC) are called Microsoft Reaction Cards (MRC) since they were developed by Microsoft [154] as part of a “desirability toolkit” created to get the quality of desirability, a key component in user satisfaction [159]. MRC consist of a pack of 118 cards with 60% positive and 40% negative or neutral adjectives, from which subjects choose the words that reflect their feelings toward their interactive experience with a product [154]. Assessment based on PRC has been recognized as one of the preferred methods for measuring the perceived desirability of visual designs [159]. Figure 68 shows an excerpt of the 118 positive and negative phrases of Microsoft Reaction Cards.

The remainder of this chapter is organized as follows: Section 7.2 describes the Capability Design Tool. Section 7.3 (Validation using Technical Action Research) describes the experimental planning.

Section 7.4 includes the action research procedure. Analysis and Interpretation of the TAR results are detailed in Section 7.5. Section 7.6 includes the discussion about the threats to validity of the results obtained in the experiment. Finally, the conclusions of the experiment are included in Section 7.7.

Accessible	Creative	Meaningful	Slow
Advanced	Customizable	Motivating	Sophisticated
Annoying	Cutting edge	Not Secure	Stable
Appealing	Dated	Not Valuable	Sterile
Approachable	Desirable	Novel	Stimulating
Attractive	Difficult	Old	Straight Forward
Boring	Disconnected	Optimistic	Stressful
Boxy	Disruptive	Ordinary	Time-Consuming
Business-like	Distracting	Organized	Time-Saving
Busy	Dull	Overbearing	Too Technical
Confident	Enthusiastic	Reliable	Unrefined
Confusing	Essential	Responsive	Usable
Connected	Exceptional	Rigid	Useful
Consistent	Exciting	Satisfying	Valuable
Controllable	Expected	Secure	
Convenient	Familiar	Simplistic	<small>*Developed by and © 2002 Microsoft Corporation. All rights reserved.*</small>
Complex	Engaging	Professional	Undesirable
Comprehensive	Entertaining	Relevant	Unpredictable

**Figure 68. An excerpt of the 118 positive and negative phrases of Microsoft Reaction Cards**

## 7.2 Background: Capability Design Tool

CDD is a novel paradigm in which services are customised on the basis of the essential business capabilities and delivery is adjusted according to the current context [160]. The CDD methodology for capability-driven design and development consists of various components addressing different modelling aspects, such as context modelling, business services modelling, pattern modelling or capability modelling.

The CaaS project developed three components to support CDD [152]:

- (i) CDD methodology, is based on agile and model driven information systems development principles and consists of the CDD development process, a language for

representing capabilities according to the CDD meta-model, as well as modelling tools.

- (ii) Capability delivery patterns, representing reusable solutions for reaching business goals under different situational contexts. The context defined for the capability should match the context in which the pattern is applicable in. Patterns will represent reusable solutions in terms of business process, resources, roles and supporting IT components (e.g. code fragments, web service definitions) for delivering a specific type of capability in a given context.
- (iii) CDD environment providing a modelling tool called Capability Design Tool (CDT). The CDT (Figure 69) is designed as an integrated development environment built using Eclipse Modelling Framework (EMF) technologies<sup>13</sup> and Graphiti<sup>14</sup> on top of Eclipse's Graphical Editing Framework-GEF<sup>15</sup>. CDT supports capability modelling according to the CDD metamodel, including context modelling and goal, process and concept models.

We modified the source code of CDT to include gesture-based interaction such a way it supports two modes of operation:

- (i) The traditional interaction mode already existing in the tool, in which the user can manipulate the primitives and connectors contained in a diagram using mouse and keyboard.
- (ii) The gesture-based interaction mode is added in the CDT by using gestUI, in which the user can draw diagrams by means of gestures sketched by a finger or pen to obtain a primitive.

---

<sup>13</sup> <http://www.eclipse.org/emf>

<sup>14</sup> <http://www.eclipse.org/graphiti/>

<sup>15</sup> <http://www.eclipse.org/gef>

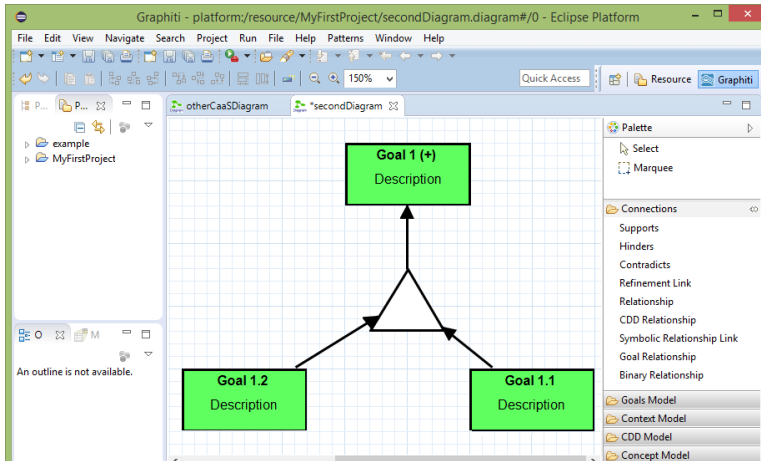


Figure 69. CDT with traditional interaction using keyboard and mouse

Figure 70 shows the CDT interface with gesture-based interaction. In this case, we have included two new elements: (i) the palette (right) allows changing the operation mode between traditional and gesture-based interaction; (ii) the main menu (up) has a new item (“Gesture”) to redefine custom gestures based on our approach.

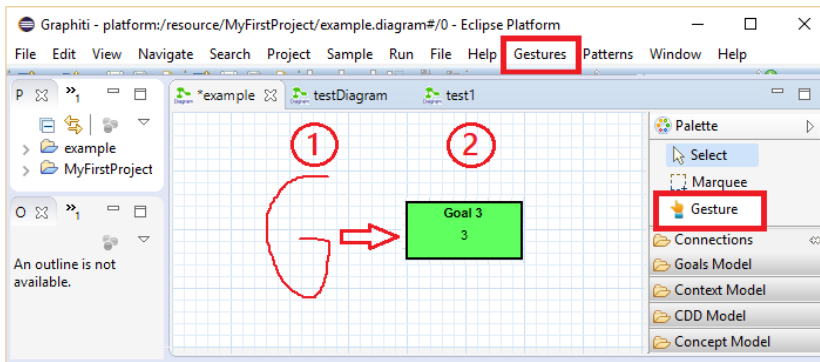


Figure 70. CDT with gesture-based interaction

### 7.3 Validation using Technical Action Research

The foundations of this TAR [150] are supported by means of setting up a theoretical framework, which allows the definition of research questions, response variables and their measures.

### 7.3.1 Goal of the TAR

The goal is to validate gestUI in real-world conditions in relation to two parameters:

- i. its *acceptance* by means of:
  - Perceived Ease of Use (PEOU), its definition is included in Section 6.1. According to Davis [137], when a software system is perceived as easier to use than another, it is more likely to be accepted by users;
  - Perceived Usefulness (PU) by the subjects, its definition is included in Section 6.1. According to Davis [137] if a user perceives the system as an effective way of performing the tasks, then there is a positive user-performance relationship.
- ii. The *user experience* by using the User Experience Questionnaire (UEQ) and the Microsoft Reaction Cards (MRC).

In this validation, we wanted to know if gestUI can help software engineers in defining custom gestures and including gesture-based interaction in existing user interfaces of a CASE tool used in an industrial context. Then, with the purpose of knowing how gestUI is perceived in this context we measure PEOU and PU with subjects who use the CDT tool in their daily work. That is, in this second evaluation, we are interested in knowing how is perceived gestUI when it is used to include gesture-based interaction in the aforementioned CASE tool.

In the first evaluation described in the previous chapter, PEOU and PU were measured within a different context and applying gestUI to include gesture-based interaction in a form-based software.

### 7.3.2 Experimental subjects

The TAR was conducted in collaboration with two technical analysts from Everis, a partner in the CaaS Project.



The technical analysts were women computer engineers with at least 5 years of experience in software development. They also had experience in using CDT with the traditional interaction. They are currently working on a CaaS project using the CDT tool with traditional interaction (keyboard and mouse) and had never seen gestUI before the TAR session. The background and experience of the subjects were found through a demographic questionnaire handed out at the first session of the experiment. This instrument consists of 15 questions on a 5-point Likert scale.

### **7.3.3 Research questions**

We focused on four research questions:

**RQ1:** *Do the subjects consider that gestUI is easy to use and useful in defining custom gestures?*

**RQ2:** *Do the subjects consider that gestUI is easy to use and useful for gesture-based interactions on user interfaces?*

**RQ3:** *What is the subject's experience when performing the process of obtaining gesture-based interfaces with gestUI?*

**RQ4:** *What is the desirability level of subjects when they use gestUI to generate gesture-based interfaces?*

### **7.3.4 Factor and Treatment**

In this case, the factor detected in the experiment is the CDT interaction method. This factor has only one treatment: the use of gesture-based interaction. We chose only this treatment since it was the goal of the experiment and the subjects already had knowledge of the process using the traditional interaction (mouse and keyboard).

**Table 69. Instruments defined for the validation**

Instrument	Description
Gesture Definition Form	Form in which the subjects defined the gestures to be used in the TAR to draw diagrams using gestures
Demographic Questionnaire	Questionnaire to assess the subjects' knowledge and experience of the technologies and concepts used in the experiment
Task Document	Document that describes the task to be performed in the action research using the gestUI method and containing empty spaces to be filled in by the subjects with start time and end time of the experiment
Post-test Questionnaires	Questionnaire with 16 questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree) to evaluate gestUI performance in the definition of custom gestures and in the inclusion of gesture-based interaction
Product reaction method	Questionnaire containing 118 positive and negative adjectives, employed by the subjects to describe their experience [161] with gestUI.
User Experience Questionnaire (UEQ)	Questionnaire to obtain fast and immediate measurement of user experience of interactive products. This questionnaire contains 6 scales with a total of 26 items: Attractiveness (Annoying/enjoyable, unlikable/pleasing, unpleasant/pleasant, good/bad, attractive/unattractive, and friendly/unfriendly); Efficiency (Fast/slow, inefficient/efficient, impractical/practical, and organized/cluttered); Perspicuity (Not understandable/understandable, easy to learn/difficult to learn, complicated/easy, clear/confusing); Dependability (Unpredictable/predictable, obstructive/supportive, secure/not secure, meets expectations/does not meet expectations); Stimulation (Valuable/inferior, boring/exciting, not interesting/interesting, and motivating/demotivating); Novelty (Creative/dull, inventive/conventional, usual/leading edge, conservative/innovative)

### 7.3.5 Response variables

Response variables are the effects studied in the experiment caused by the manipulation of factors. In this experiment, we have four response variables (PEOU, PU, UEQ and MRC) to analyse the acceptance of the Everis technical analysts.

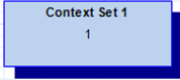

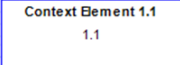

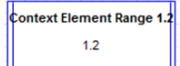

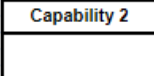

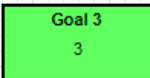

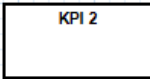

### 7.3.6 Instruments for the TAR

All the material required to support the experiment was developed beforehand, including the preparation of the experimental object, instruments and task description documents for data collection used during the execution of the experiment. The instruments prepared to perform the TAR are described in Table 69.

### 7.3.7 Experimental Object

With the aim of performing the TAR, we considered CDT as an experimental object in this validation. Using this experimental object, the subjects must sketch an excerpt of a diagram defined in Everis (see Figure 71), with the primitives included in Table 70. This diagram is an example of work related to a project on the development of an e-government platform.

**Table 70. Gesture catalogue defined by the subjects**

Primitive	Symbol	Gesture
Context Set		
Context Element		
Context Element Range		
Capability		
Goal		
KPI		

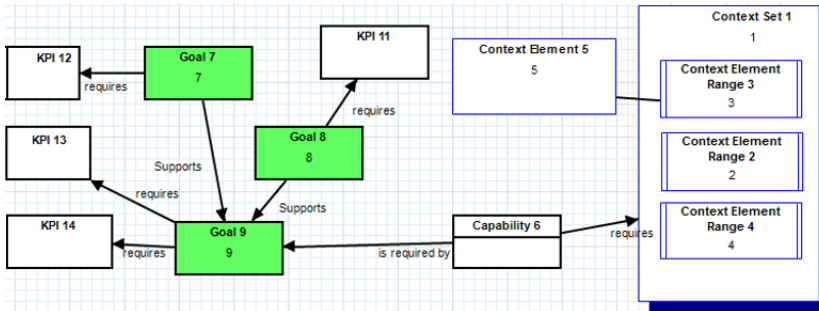


Figure 71. Excerpt of a model defined in Everis

## 7.4 Action Research Procedure

This section describes the TAR procedure used to conduct the experiment performed in a meeting room in Everis offices. Previous to the TAR session, a pilot test was run with a researcher from the PROS Research Centre in the Universitat Politècnica de Valencia. This pilot test helped us improve the understandability of the instruments.

The steps of the experiment procedure are:

Step 0: The first step is related to the gesture catalogue definition, which was completed for the subjects before the TAR session. In a previous session, the subjects filled in the Gesture Catalogue Definition Form with the gestures to be used in CDT to draw the aforementioned diagram. The subjects defined custom gestures for each primitive of the aforementioned diagram according to their preferences (Table 70).

Step 1: Before the experiment each subject filled in a Demographic Questionnaire in which they were asked about their experience in tasks related with CDT, experience with gesture-based interaction, experience in software development, and experience in model-driven development.

Step 2: The planned action research procedure was described to the subjects with a verbal explanation.

Step 3: By means of a live demo, the subjects were instructed to use gestUI in gesture definition and inclusion of gesture-based interaction on the user interface of CDT.

Step 4: Subjects used gestUI to define the gestures previously specified in the Gesture Catalogue Definition Form (Table 70) following the process defined in Chapter 4 to define a gesture. Subjects used the Task Description Document to follow the required instructions in order to obtain the gesture catalogue, and to include gesture-based interaction in the interface of CDT.

Step 5: Subjects filled in the Post-Task Questionnaire on their opinion of gestUI regarding custom gesture definition and inclusion of gesture-based interaction in CDT.

Step 6: Subjects employed CDT to draw the diagram shown in Figure 71. They used the Gesture Catalogue Definition Form to help them with the previously defined gestures.

Step 7: Subjects redefined three gestures using the module to redefinition included in CDT.

Step 8: Subjects filled in the Post-Task Questionnaire to assess gestUI capacity to define custom gestures and to include gesture-based interaction.

Step 9: Subjects filled in the User Experience Questionnaire on their experience with custom gesture definition and the inclusion of gesture-based interaction.

Step 10: Subjects filled in the Microsoft Reaction Cards on the desirability level of using gestUI to define custom gestures and include gesture-based interaction.

Table 71 contains a summary of the steps performed in the experiment, the instruments used in each step and the time estimated to perform each step.

**Table 71. A summary of the experiment procedure**

ID	Description	Instrument used		Time
1	Subjects filled in the <i>Demographic Questionnaire</i> on their experience in related topics.	Demographic questionnaire		8 min.
2	The planned action research procedure was described to the subjects.	Verbal explanation		10 min.
3	Subjects were instructed to use gestUI in gesture definition and inclusion of gesture-based interaction on the user interface of the CDT.	Live demo		10 min.
4	Subjects employed gestUI to define the gestures previously specified in the gesture catalogue. They employed the <i>Task Description</i> document to follow the required instructions.	Task Document	Description	15 min.
5	Subjects filled in the Post-Task Questionnaire on their opinion of gestUI.	Post-Task Questionnaire		5 min.
6	Subjects employed the CDT to draw the diagram shown in Figure 71. They used the <i>Gesture Catalogue Definition Form</i> to help them with the previously defined gestures.	Task Document	Description and Gesture Catalogue Definition Form	20 min.
7	Subjects redefined three gestures using the module to redefinition included in the CDT	Task Document	Description	10 min.
8	Subjects filled in the PEOU and PU Post-Task Questionnaire to assess gestUI capacity to define custom gestures and include gesture-based interaction.	Post-Task Questionnaire		5 min.
9	Subjects filled in the User Experience Questionnaire on their experience with custom gesture definition and the inclusion of gesture-based interaction.	User Experience Questionnaire		5 min.
10	Subjects filled in the reaction cards on the desirability level of using gestUI to define custom gestures and include gesture-based interaction.	Microsoft Cards	Reaction	8 min.
<b>Total time</b>				<b>96 min.</b>

## 7.5 Analysis and Interpretation of results

Since there were only 2 subjects involved in the TAR we did not apply any statistical test to analyse and interpret the information. We analysed the responses of each subject regarding each research question obtained from the aforementioned instruments containing the questionnaires filled in by the subjects:

*Regarding RQ1*, the results obtained through the questionnaires show that both subjects think that the feature to define custom gestures implemented in gestUI is perceived as both easy to use and useful.

*Regarding RQ2*, the results obtained from the questionnaires show that both subjects think that the feature to include gesture-based interaction implemented in gestUI is perceived as both easy to use and useful.

*Regarding RQ3*, after completing the tasks, the subjects filled out the UEQ, obtaining the results shown in Figure 72 and Figure 73. The values vary from -3 to +3. The six scales, their description [156], the values obtained and their corresponding percentages in the TAR are shown in Table 72.

**Table 72. Results obtained from the UEQ**

Scale	Description	Value obtained	
		Custom gesture definition	Gesture-based interaction
Attractiveness	Overall impression of the product	2.42 (81%)	2.25 (75%)
Perspiciuity	Is it easy to get familiar with the product?	2.75 (92%)	2.75 (92%)
Efficiency	Can users solve their tasks without unnecessary effort?	1.38 (46%)	1.63 (54%)
Dependability	Does the user feel in control of the interaction?	1.63 (54%)	2.00 (67%)
Stimulation	Is it exciting and motivating to use the product?	2.25 (75%)	2.50 (83%)
Novelty	Is the product innovative and creative?	2.38 (79%)	2.63 (88%)

The results obtained show that efficiency and dependability scales in custom gesture definition had values lower than 67%. The efficiency scale in gesture-based interaction also had a value lower than 67%. In both cases, efficiency is related to items such as: fast/slow, inefficient/efficient, impractical/practical, and organized/cluttered.

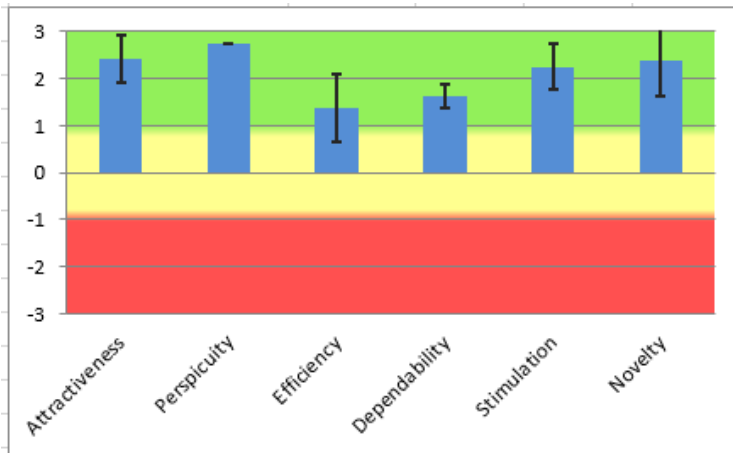


Figure 72. UEQ results: custom gesture definition interaction

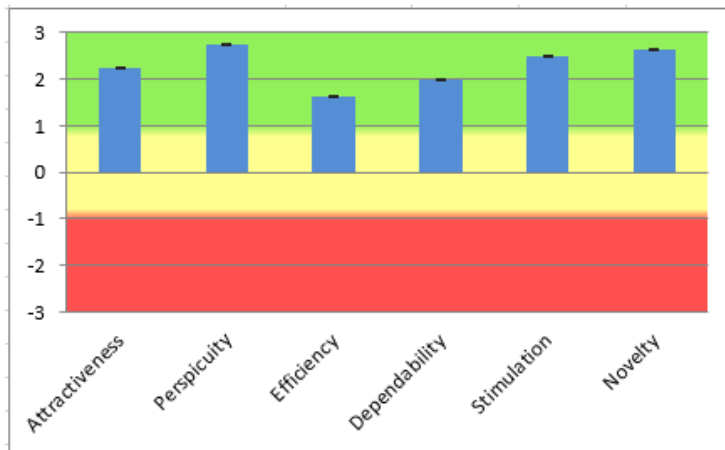


Figure 73. UEQ results: inclusion of gesture-based interaction

Regarding RQ4, we applied MRC to study positive and negative aspects related with the inclusion of gesture-based interaction (blue line) and custom gesture definition (orange line). With the aim of



reporting these results, we use two figures: (i) Figure 74 shows positive results and Figure 75 shows negative results. Values showed in Figure 74 represent the frequency of use of each positive adjective for the subjects in the experiment (e. g. “simplistic” was selected two times, one time per subject). These values correspond to the values included in the "Value" column in Table 73, which shows the most frequently used positive adjectives on the gestUI experience.

**Table 73. Reaction cards positive results**

Process	Positive Adjective	Value
Custom Gesture Definition	Simplistic	2
	Innovative, Customizable, Useful, Clear, Easy to use.	1
Inclusion of Gesture-based Interaction	Innovative, Useful	2
	Comfortable, Creative, Attractive, Time saving, Simplistic, Easy to use.	1

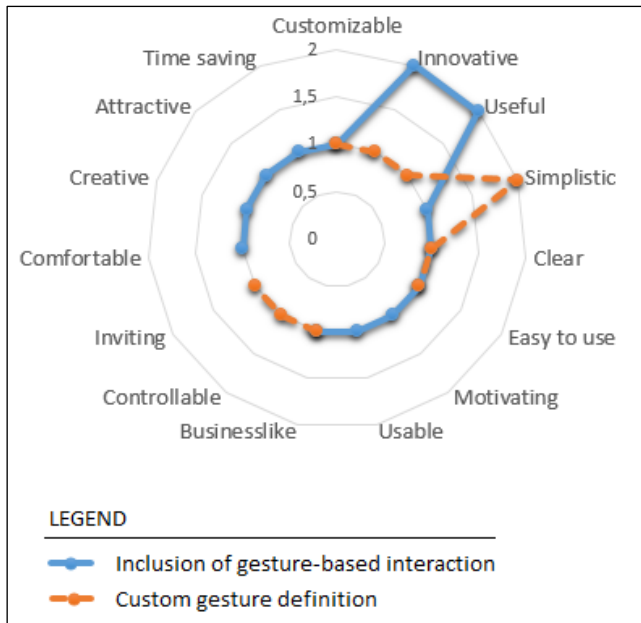
From Figure 75 we obtained the negative results related with the inclusion of gesture-based interaction and custom gesture definition shown in Table 74. The meaning of the values included in this figure is the same as in Figure 74.

**Table 74. Reaction cards negative results**

Process	Negative Adjective	Value
Custom Gesture Definition	Too technical	2
	Time consuming, Unattractive	1
Inclusion of Gesture-based Interaction	Slow	2
	Sensible, Annoying, Fragile	1

In the case of custom gesture definition, the subjects described the custom gesture definition as simplistic but also too technical and time consuming. This opinion could have been related with the null experience of the subjects in custom gesture definition in using CDT and also because the UI of gestUI to define gestures could have been better designed to obtain an attractive gesture definition process. In the case of the inclusion of gesture-based interaction, the subjects defined it as innovative and useful, but also that the inclusion of gesture-based interaction is slow, sensitive and annoying. This opinion

could have been due to the null experience of the subjects in the use of gestures to draw diagrams.



**Figure 74. Reaction cards positive results**

The subjects considered the new proposal to redefine gestures as useful and thought that it helped them to solve memorizing or sketching problems.

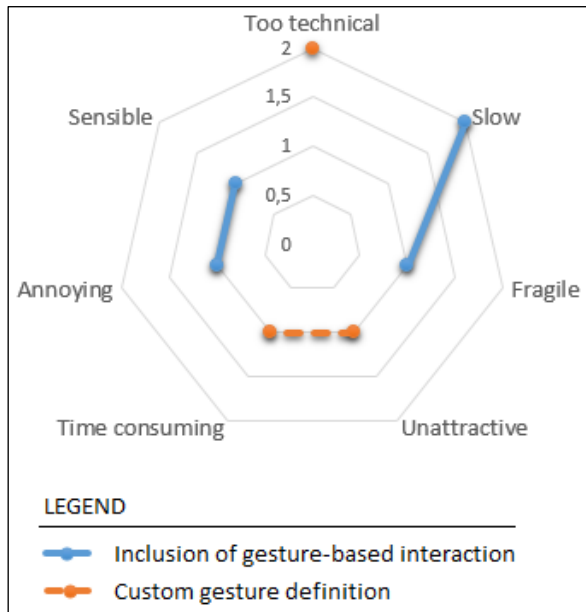


Figure 75. Reaction cards negative results

## 7.6 Threats to validity

This section deals with the most important threats to the validity of this evaluation, classified according to Wohlin et al. [138]:

(A) Internal Validity: the main threats to the internal validity of the experiment are: (1) *Subject experience in tasks performed in the experiment*: this threat was eliminated since none of the subjects had any experience in tasks related with custom gestures definition or the inclusion of gesture-based interaction in user interfaces. (2) *Subject experience in the use of CDT with gesture-based interaction*: this threat was eliminated since none of the subjects had any experience in the use of CDT with gesture-based interaction.

(B) External Validity: the main threat to the external validity of the experiment was: (1) *Duration of the experiment*: since the duration of the experiment was limited to 96 minutes, only one diagram was selected with six primitives and six gestures. However, experience in the use of CDT in traditional interaction and repetitive tasks could have

affected the duration of the experiment, since the subjects already knew the process to be performed. This threat could not be ruled out since they were familiar with the repetitive tasks required to build the diagram. (2) *Representativeness of the results*: the experiment was performed in an industrial context on subjects with no experience in the tasks related with the experiment. This means the results could only be representative for novice evaluators with no experience in custom gesture definition and in the inclusion of gesture-based interaction.

(C) Validity Conclusions: The main threat to the validity of the experiment was: (1) *Validity of the statistical test applied*: In this case, we did not apply any statistical tests to obtain answers to the research questions because the sample size was too small. However, we considered the results obtained with other methods, such as MRC and UEQ.

## 7.7 Conclusions

This chapter describes the validation of gestUI in industry by means of a TAR, studying (i) PEOU; (ii) PU; (iii) the desirability level with MRC; and, (iv) user experience with UEQ.

To validate the performance of gestUI in industrial settings, we included gesture-based interaction in the CDT tool from the CaaS Project. The subjects were two business analysts from a consultancy firm who defined custom gestures by either fingers or pen/stylus and also redefined some gestures from the gesture catalogue considered in the experiment.

The main findings of the study are: (1) gestUI helped the business analysts to define custom gestures and include gesture-based interaction in user interfaces. (2) The subjects considered gestUI easy to use and useful for defining custom gestures and including gesture-based interaction in CDT. (3) Although the subjects did not enjoy defining custom gestures and applying the automated transformations, they did feel motivated while using this version of the CDT.



---

# CONCLUSIONS, DISCUSSION AND FUTURE WORK

# 8

The topics covered in this chapter are:

- 8.1 Summary of the thesis
- 8.2 Contribution of this thesis
- 8.3 Future Work
- 8.4 Conclusion
- 8.5 Publications



## **Chapter 8. Conclusions, Contributions and Future Work**

This chapter summarizes the thesis, discusses its findings and contributions, points out limitations of the current work, and also outlines directions for future research.

The purpose of this thesis is to develop a methodological framework based on MDA for the development of user interfaces with gesture-based interaction of software systems.

The chapter is divided into five sections. Section 8.1 is a summary of the thesis. Section 8.2 presents a discussion of the contributions of the current work. Section 8.3 discusses the future work. Section 8.4 brings the thesis to a conclusion and finally, Section 8.5 describes the publications that emerged during the development of this thesis.

### **8.1 Summary of the thesis**

This thesis has introduced gestUI, a model-driven method to define custom gestures and to include gesture-based interaction in user interfaces of software systems.

Chapter 1 contains the introduction of the thesis. It describes the motivation, the problem statement, the research questions and the thesis objectives. Also, it describes the research methodology, the expected contributions and the thesis context.

In Chapter 2, a theoretical framework has been presented where a series of concepts related to the work developed in this thesis have been included. It is considered that the thesis is framed in two areas: model-driven development and human-computer interaction. In this sense, this theoretical framework has been divided into two parts including in each one concepts that help to explain and to understand the work done.

Chapter 3 includes the State of the Art in the two aforementioned areas. We describe the results of the search in the related literature



regarding gesture representation and the gesture recognition tools with the aim of knowing the different techniques used to describe the gestures and the tools developed for their recognition. Then, we describe the results of the search in the related literature with respect to the Model-driven Engineering in the Human-Computer Interaction, where we review works that consider models to create a user interface that includes user interaction. Finally, we include the results of the search in the related literature in relation to two evaluations techniques: empirical evaluation and technical action research with the aim of knowing how are used these techniques in the evaluation of methods.

In Chapter 4, we describe our proposal called gestUI. In this chapter we explain why we consider the model-driven paradigm in the design of gestUI. Also, we explain why we consider the Model-View-Controller design pattern to design the method. Then, we describe the needed resources to obtain gesture-based user interfaces. The description of gestUI comprises features, metamodel description, components of the method and model transformations used in the process to include gesture-based interaction in user interfaces. Also, we explain the process to personalize the gesture definition from scratch and from an existing definition of custom gestures. Finally, we include an overview of gestUI to include gesture-based interaction in user interfaces.

Chapter 5 describes the tool support that has been built to support the models and activities of gestUI. This chapter includes the description of its components and how have been implemented using Eclipse Modelling Framework and Java programming language. The chapter concludes with a demonstration of the applicability of the tool support in a form-based software system and in a Case Tool.

In Chapter 6 the empirical evaluation performed to evaluate the usability of gestUI is described. The usability was measured with effectiveness, efficiency and satisfaction. The chapter includes the design of the experiment and the analysis of the reliability of the data

obtained in the experiment. Then, it describes the results obtained applying Wilcoxon Signed-rank test and Shapiro-Wilk because the data is not normally distributed. Finally, we include the discussion of the results regarding effectiveness, efficiency and satisfaction.

Chapter 7 includes the description of the technical action research applied to evaluate gestUI in an industrial context. This chapter describes the Capability Design Tool that has been modified to include gesture-based interaction using gestUI. Also, the design of the experiment, the analysis and interpretation of the results are included. Finally, the threats to validity of the experiment are described.

## 8.2 Contribution of this thesis

As a result of the development of this thesis various contributions can be highlighted. These contributions are the evidence of achieving the research goals, as well as the answers of the established research questions. The main contributions are presented below:

- With regard to research question 1 (**What elements should be considered for the definition of a method to include gesture-based interaction in user interfaces?**), we contribute with a theoretical framework to establish a common knowledge about the model-driven paradigm and gestUI (Chapter 2). In this chapter, we define the most important concepts related with the elements required to define a method to include gesture-based interaction in user interfaces of software systems.
- Regarding research question 2 (**What model-driven methods exist to include gesture-based interaction in user interfaces with human-computer interaction based on gestures?**), our contribution is centred in Chapter 3 (State of Art) where we describe the results of a search of related literature regarding methods that permit to define user interfaces.
- With regard to research question 3 (**Is it possible to define a model-driven method for the inclusion of gesture-based interaction in software systems user interfaces?**), we can say that it is possible to define a method based on the model-driven

paradigm that permits the inclusion of gesture-based interaction in user interfaces of software systems. Our proposal (gestUI, described in Chapter 4) is a model-driven method designed to define custom gestures and to include gesture-based interaction.

By following the model-driven paradigm, gestUI is contained in three layers: platform-independent layer, platform-specific layer and source code layer where are defined the elements that permit to define custom gestures and to include gesture-based interaction. Therefore, our contribution is related with the definition of these elements: conceptual model definition (metamodel described in Section 4.5.2), model transformations (model-to-model and model-to-text transformations described in Section 4.5.4), transformation rules that permit to obtain the user interface with gesture-based interaction and the additional feature to redefine existing gestures of a user interface (described in Section 4.6).

Also, we define a tool that permits to represent a gesture based on the conceptual model specified with the previously obtained information (included in Section 4.5.2). We include a gesture recognition algorithm that permits to recognise custom gestures sketched by the users. In this case, we adopt an existing gesture recognition algorithm known as \$N.

We use Java programming language and Eclipse Modelling Framework to define components of the proposed method in a tool support (Chapter 5) to demonstrate its applicability.

- Regarding research question 4 (**What advantages and disadvantages has the model-driven method for the inclusion of gesture-based interaction in software system user interfaces?**), two demonstrations have been performed (described in Chapter 5) with gestUI to evaluate its feasibility before to apply it in empirical tasks. We apply gestUI to test custom gestures in three gesture recognition tools (quill, iGesture and \$N) obtaining good results. Also, we apply gestUI to include gesture-based interaction in a form-based software system. In this case, we define a gesture

catalogue and we include gesture-based interaction in a user interface of this software system.

We perform an empirical comparative evaluation to validate gestUI (described in Chapter 6). We measure the usability of gestUI based on efficiency, effectiveness and satisfaction obtaining positive results.

Additionally, we perform a technical action research of gestUI in everis with the aim of evaluating our method in an industrial context (described in Chapter 7). This evaluation was performed in the context of the “Capability as a Service” - CaaS Project (FP7 ICT Programme Collaborative Project no. 611351). We measure usability of gestUI and user satisfaction when they use gestUI to perform tasks associated with a project related with CaaS Project. We obtain positive results of this evaluation.

### 8.3 Future work

The research that is presented in this thesis is not a closed work; it can be improved and extended in several ways. The following paragraphs summarize the research directions that are planned for the near future. The main goal of this future work will be to overcome some of the limitations of the work that has been developed thus far.

- We consider that is necessary to extend the solutions of the tool to include mobile devices as target platform in the application of gestUI. In this way, users will be able to define custom gestures and to include gesture-based interaction via gestUI on mobile devices, overcoming the current difficulties when the developers use the traditional tools for these tasks. Therefore, we need to include in the tool support some model transformations to consider the mobile devices as an additional target when we apply gestUI.
- We plan to apply gestUI in the user interface development process of software systems for disabilities people. The main goal is to help to these people to improve the communication with other people and to improve the access to public services requiring technology.

This is an application of gestUI in mobile devices that will help us to evaluate the additions in gestUI to include a new target platform.

## 8.4 Conclusion

Interfaces with new techniques of interaction play an important role in the field of software engineering that mainly includes software systems supporting gesture-based interaction. At present, there are more and more devices that support gesture-based interaction, however, certain tasks make difficult the process of development of software system with this type of interaction. Developers of such software systems are faced with the following challenges:

- i. Manage high complexity: Developing software systems that support gesture-based interaction across multiple heterogeneous devices represent a complex process.
- ii. An increase of efficiency of multi-platform software development across heterogeneous computing platforms (Windows, iOS, Android, Windows Phone etc.).
- iii. An integration of user centered design into the development process, extending the existing methods to cover the necessary adaptation options.

The aim of this PhD thesis is to propose a model-driven method that helps to solve these challenges with the capability to be integrated into different model-driven processes to develop user interfaces.

Our proposal described in this thesis has the following benefits:

- It is based on MDA which has in its favour the advantages of the methodology.
- It is independent of target device platform. The target platform may be decided by the developer in the PSM definition stage.
- It does not require the developer to learn a set of SDK's or some programming languages.

- Time to develop is reduced when the source code of gesture-based user interface is automatically generated.
- Definition of the new gestures is performed through a specification of features, without the necessity of a developer or user.

Empirical evaluation was performed and it showed that gestUI is perceived more efficient than the code-centric method considered in the evaluation. Regarding effectiveness, this got similar results in the empirical evaluation.

Then, we apply a Technical Action Research in an industrial context in collaboration with the company “Everis” (Valencia, Spain) in order to know the usability level by means of UEQ and MRC. The results are described by means of positive and negative phrases as shown in Chapter 7.

Its main current limitations are related to the target interface technologies (currently, only Java) and the fact that multi-finger gestures are not supported. These limitations will be addressed in future work.

## 8.5 Publications

### Papers

- Parra, O.  
*“A model-driven method for gesture-based interface requirements specification”*, 20th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2014). Doctoral Symposium. Publication: On-line <http://ceur-ws.org/Vol-1138/>. ISSN: 1613-0073, Essen, Germany, 2014.
- Parra, O., España, S., Pastor, O.,  
*“Including multi-stroke gesture-based interaction in user interfaces using a model-driven method”*, XVI International Conference on Human Computer Interaction – Interaccion 2015, Publication: Online <http://dl.acm.org/citation.cfm?doid=2829875.2829931>,

- ISBN: 978-1-4503-3463-1, DOI: 10.1145/2829875.2829931. Vilanova I la Geltrú, Spain, 2015.
- Parra, O., España, S., Pastor, O.,  
“A Model-driven Method and a Tool for Developing Gesture-based Information System Interfaces”, CAiSE Forum at the 27<sup>th</sup> International Conference on Advanced Information System Engineering – CaiSE 2015, Publication: Online <http://ceur-ws.org/Vol-1612/>. ISSN: 1613-0073, Stockholm, Sweden, 2015.
  - Parra, O., Pastor, O.,  
“gestUI: Un método dirigido por modelos para incluir interacción gestual multi-trazo en interfaces de usuario”, XVIII Iberoamerican Conference on Software Engineering (CibSE 2015), Poster. Publication: Print ISBN: 978-1-5108-0387-9. Lima, Perú, 2015.
  - Parra, O., España, S., Pastor, O.  
“Including Gesture-based Interaction in Capability Design Tool”, 2<sup>nd</sup> International Workshop on Capability-oriented Business Informatics. Publication: Online <http://ceur-ws.org/Vol-1408/>. ISSN: 1613-0073. Lisbon, Portugal, 2015.
  - Parra, O., España, S., Panach, J., Pastor, O., Buriel, V.  
“gestUI tool: A tool to include gesture-based interaction in user interfaces through model-driven”, Tool Demonstration on 35<sup>th</sup> International Conference on Conceptual Modeling –ER 2016, Core Index: A. Publication: On-line <http://er2016.cs.titech.ac.jp/assets/papers/ER2016-tool-parra.pdf>, Gifu, Japan, 2017.
  - Parra, O., España, S., Pastor, O.,  
“Tailoring user interfaces to include gesture-based interaction with gestUI”, 35<sup>th</sup> International Conference on Conceptual Modeling – ER 2016, Core Index: A. Publication: On-line <https://link.springer.com/book/10.1007%2F978-3-319-46397-1?page=3#toc>, Proceedings part of the Lecture Notes in Computer Science (LNCS, volume 9974). DOI: 10.1007/978-3-319-46397-1, Gifu, Japan, 2017.
  - Parra, O., España, S., Panach, J., Pastor, O.

*“Extending and validating gestUI using Technical Action Research”*,  
IEEE 11<sup>th</sup> International Conference on Research Challenges in  
Information Science – RCIS 2017, Core Index: B, Brighton, UK,  
2017.

## **Journals**

- Parra, O., España, S., Pastor, O.,  
*“gestUI: A Model-driven Method and a Tool for Including  
Gesture-based Interaction in User Interfaces”*, Complex Systems  
Informatics and Modeling Quarterly – CSIMQ, Vol. 6, Publication:  
Online <https://csimq-journals.rtu.lv/article/view/csimq.2016-6.05>. ISSN: 2255-9922. DOI: 10.7250/csimq.2016-6.05. 2016.
- Parra, O., España, S., Panach, J., Pastor, O.  
*“An empirical comparative evaluation of gestUI to include gesture-  
based interaction in user interfaces”*, Under Review in Journal of  
Systems and Software, July 2017.





A CODE-CENTRIC  
METHOD FOR  
DEVELOP USER  
INTERFACES WITH  
GESTURE-BASED  
INTERACTION

A

The topics covered in this chapter are:

- A.1 Introduction
- A.2 The code-centric method



## Appendix A. A code-centric method for develop user interfaces with gesture-based interaction

### A.1 Introduction

This appendix presents the description of a code-centric method for develop user interfaces with gesture-based interaction.

Figure 76 shows the user interface development life cycle for this method. In this case, we start from existing activities and products (represented by means of colour grey) used to develop interfaces that must be enhanced to support gesture-based interaction and a set of new activities and products (represented by means of the colour white) that deal explicitly with the gesture-based interaction. In the following section we describe proposed activities and products of this method.

### A.2 The code-centric method

The **code-centric method** consists in a set of tasks [29] (e.g. conceptualization and requirements gathering, analysis and functional description, design, coding, testing and deployment) related with the implementation of a software system using a programming language and a tool where software engineers work entirely by editing source code (e.g. Microsoft Visual Studio, Eclipse Window Builder, NetBeans, etc.).

An example of this method is the process to develop a user interface by means of Eclipse SWT Designer (Window Builder) [162]. This toolkit does not include components to define custom gestures nor to include gesture-based interaction. SWT works under the assumption that the user interface is already implemented and the developer writes additional source code containing gesture-based interaction.



catalogue and the actions to be performed using said gesture catalogue. The product obtained in this process is a requirements document containing the specification of the interaction between gestures and actions included in a user interface.

2. Activity C2: this activity permits software engineers to select the user interface to include the gesture-based interaction according to the aforementioned requirements specification, then he/she analyses the source code of the selected user interface with the aim of determining the actions included in the user interface source code. The software engineer defines the gesture-action correspondence by specifying the gesture that allows the execution of an action included in the user interface.
3. Activity C3: this activity allows software engineers to specify, by means of XML language each gesture included in the requirements document of the gesture catalogue. This gestures specification is required in order to be supported by the gesture recognizer algorithm. In this work we use \$N [126] as the gesture recognizer. The product obtained in this step is the gesture catalogue specification written in XML.
4. Activity C4: in this activity the software engineer implements the methods needed to execute the actions specified with the previously defined gestures, that is, the software engineer combines two products (i) gesture-based interaction source code and (ii) gesture catalogue specification in order to obtain the gesture-based user interface. The product obtained in this last step is the user interface source code including gesture-based interaction.
5. Activity C5: this permits testing gestures using existing frameworks (e.g. quill, iGesture, \$N). The gesture catalogue is generated according to the gesture definition of each framework, hence the users sketch gestures in order to test them.

There are activities represented in Figure 76 (e.g. “Implement interface”, “Interface design”) whose functionality is included in the process of development of user interfaces using some tools available. These activities are not described in this Appendix because we consider that these activities belong to traditional development

methods for obtaining user interfaces by using typical development tools.

When a software engineer employs a code-centric method to include gesture-based interaction some of the following problems are involved [117] [118] [119]: (i) the software engineer has two options to obtain the source code: writing the methods required to implement the software from scratch or adapting existing source code; (ii) the gesture specification is not multi-platform; (iii) it is hard to reuse the source code to support gesture-based interaction in other platforms; (iv) software engineers require skills in the programming language of each platform employed in the implementation of IS user interfaces; (v) in some cases, the IDE is not available in all platforms required by users.

This thesis proposes a method that pretends to help to solve these problems.

## References

- [1] J. v. Biljon and P. Kotzé, "Modelling the Factors that Influence Mobile Phone Adoption," *Proceedings of the 2007 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pp. 152-161, 2007.
- [2] G. Dapper and P. Egbert, "A gestural interface to free-form deformation," *Proceedings of Graphics Interface*, pp. 113-220, 2003.
- [3] D. Wigdor and D. Wixon, *Brave NUI World: Designing Natural User Interfaces for Touch and Gesture*, UK: Morgan Kaufmann Publishers - Elsevier, 2011.
- [4] E. v. d. Hoven and A. Mazalek, "Grasping gestures: Gesturing with physical artifacts," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 25, no. 3, pp. 255-271, 2011.
- [5] T. Schlomer, B. Poppinga, N. Henze and S. Boll, "Gesture Recognition with a Wii Controller," in *Proceedings of the Second International Conference on Tangible and Embedded Interaction*, Bonn, Germany, 2008.
- [6] J. Yao, T. Fernando and H. Wang, "A multi-touch natural user interface framework," *International Conference on Systems and Informatics (ICSAI)*, pp. 499-504, 2012.
- [7] D. Rubine, "Specifying gesture by example," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 329-337, 1991.
- [8] D. Saffer, *Designing Gestural Interfaces*, USA: O'Reilly Media Inc., 2009.
- [9] P. Kortum, *HCI Beyond the GUI: Design for Haptic, Speech, Olfactory, and Other Nontraditional Interfaces*, USA: Morgan Kaufmann Publishers, 2008.
- [10] M. Bhuiyan and R. Picking, "A Gesture Controlled User Interface for Inclusive Design and Evaluative Study of Its Usability," in *Journal of Software Engineering and Applications*, 2011.
- [11] M. Karam and M. C. Schraefel, "A taxonomy of Gestures in Human Computer Interaction," *ACM Transactions on Computer-Human Interactions*, pp. 1-45, 2005.
- [12] J. Yang, Y. Xu and C. S. Chen, "Gesture interface: modeling and learning," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, vol. 2, no. 4, pp. 1747-1752, 1994.



- [13] Fujitsu Laboratories Ltd., "Fujitsu Laboratories Develops Ring-Type Wearable Device Capable of Text Input by Fingertip," 13 01 2015. [Online]. Available: <http://www.fujitsu.com/global/about/resources/news/press-releases/2015/0113-01.html>. [Accessed 22 10 2016].
- [14] Y. Song, D. Demirdjian and R. Davis, "Continuous body and hand gesture recognition for natural human-computer interaction," *Journal ACM Transactions on Interactive Intelligent Systems (TiiS) - Special Issue on Affective Interaction in Natural Environments*, vol. 2, no. 1, pp. 1-25, 2012.
- [15] Y. Song, D. Demirdjian and R. Davis, "Tracking body and hands for gesture recognition: NATOPS aircraft handling signals database," in *Face and Gesture 2011*, Santa Barbara, CA, 2011.
- [16] S. Kim, Y. Ban and S. Lee, "Tracking and Classification of In-Air Hand Gesture Based on Thermal Guided Joint Filter," *Sensors*, vol. 17, no. 1, pp. 1-20, 2017.
- [17] P. Cardoso, J. Rodrigues, S. L., A. Mazayev, E. Ey, T. Correa and M. Saleiro, "A Freehand System for the Management of Orders Picking and Loading of Vehicles," *Universal Access in Human-Computer Interaction. Access to the Human Environment and Culture. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9178, pp. 422-431, 2015.
- [18] A. Weiss, M. Bader, M. Vincze, G. Hasenhütl and S. Moritsch, "Designing a service robot for public space: an action and experiences-approach," in *Proceedings of the 2014 ACM/IEEE international conference on Human-robot interaction - HRI '14*, Bielefeld, Germany, 2014.
- [19] S. Tan and A. Nareyek, "Integrating facial, gesture, and posture emotion expression for a 3D virtual agent," in *Proceedings of the 14th International Conference on Computer Games: AI, Animation, Mobile, Interactive Multimedia, Educational & Serious Games*, 2009.
- [20] L. Corral, A. Sillitti and G. Succi, "Mobile multiplatform development: An experiment for performance analysis," *Procedia Computer Science - Elsevier*, vol. 10, pp. 736-743, 2012.
- [21] J. Folks, "Using Microsoft Visual Studio to Create a Graphical User interface," 03 04 2015. [Online]. Available: [http://www.egr.msu.edu/classes/ece480/capstone/spring15/group11/doc/AppNote/ECE480\\_AppNotes\\_JoshuaFolks.pdf](http://www.egr.msu.edu/classes/ece480/capstone/spring15/group11/doc/AppNote/ECE480_AppNotes_JoshuaFolks.pdf). [Accessed 17 01 2017].

- [22] D. Gallardo, E. Burnette and R. McGovern, *Eclipse in Action. A guide for Java developers*, Greenwich: Manning Publications Co., 2003.
- [23] G. Meixner, G. Calvary and J. Coutaz, "Introduction to Model-Based User Interface," December 2013. [Online]. Available: <http://www.w3.org/2011/mbui/drafts/mbui-intro/>. [Accessed 21 02 2014].
- [24] M. Hesenius, T. Griebe, S. Gries and V. Gruhn, "Automating UI Tests for Mobile Applications with Formal Gesture Descriptions," *Proc. of 16th Conf. on Human-computer interaction with mobile devices & services*, pp. 213-222, 2014.
- [25] S. H. Khandkar, S. M. Sohan, J. Sillito and F. Maurer, "Tool support for testing complex multi-touch gestures," in *ACM International Conference on Interactive Tabletops and Surfaces, ITS'10*, NY, USA, 2010.
- [26] N. Aquino, J. Vanderdonckt, J. I. Panach and O. Pastor, "Conceptual Modeling of Interaction," in *Handbook of Conceptual Modeling. Theory, Practice, and Research Challenges*, Springer , 2011, pp. 335-358.
- [27] P. E. Papotti, A. F. do Prado, W. Lopes de Souza, C. E. Cirilo and L. Ferreira Pires, "A Quantitative Analysis of Model-Driven Code Generation through Software Experimentation," *Proceedings of 25th International Conference CAISE 2013*, vol. LNCS 7908, pp. 321-337, 2013.
- [28] S. Beydeda and V. G. M. Book, *Model-Driven Software Development*, Springer, 2005.
- [29] A. Kleppe, J. Warmer and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2003.
- [30] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*, Enschede, The Netherlands: Springer-Verlag New York Inc., 2014.
- [31] R. Wieringa, "Design science methodology: principles and practice," in *Proceeding of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 2*, Cape Town, South Africa, 2010.
- [32] F. Paternó, "Model-based tools for pervasive usability," *Interacting with Computers*, vol. 17, no. 3, pp. 291-315, 2005.
- [33] A. Hevner, S. March, J. Park and S. Ram, "Design science in information systems research," *MIS Quarterly*, vol. 28, no. 1, pp. 75-105, 2004.

- [34] T. Lukman and M. Mernik, "Model-Driven Engineering and its Introduction with Metamodeling Tools," in *9th International PhD Workshop on Systems and Control*, Slovenia, 2008.
- [35] B. Selic, "The Pragmatics of Model-Driven Development," *IEEE Computer Society*, vol. 20, no. 5, pp. 19-25, 2003.
- [36] S. Sendall and W. Kozaczynski, "Model Transformation: The Heart and Soul of Model-Driven Software Development," *Software IEEE*, vol. 20, no. 5, pp. 42-45, 2003.
- [37] K. Tripathi, "A Study of Interactivity in Human Computer Interaction," *International Journal of Computer Applications*, vol. 16, no. 6, pp. 4-6, 2011.
- [38] F. Karray, M. Alemzadeh, J. Saleh and M. Arab, "Human Computer Interaction: Overview on State of the Art," *INTERNATIONAL JOURNAL ON SMART SENSING AND INTELLIGENT SYSTEMS*, pp. 137-159, 2008.
- [39] Oxford University, *Oxford Paperback Dictionary and Thesaurus*, London, UK: Oxford University Press, 2009.
- [40] B. Altakrouri and A. Schrader, "Describing movements for motion gestures," in *1st International Workshop on Engineering Gestures for Multimodal Interfaces - EGMI 2014*, Rome, Italy, 2014.
- [41] N. Gillian and J. Paradiso, "The Gesture Recognition Toolkit," *Journal of Machine Learning Research*, vol. 15, pp. 3483-3487, 2014.
- [42] M. Kaushik and R. Jain, "Gesture Based Interaction NUI: An Overview," *International Journal of Engineering Trends and Technology (IJETT)*, vol. 9, no. 12, pp. 633-636, 2014.
- [43] Apple, "Event Handling Guide for iOS," 28 01 2013. [Online]. Available: <https://developer.apple.com/library/ios/documentation/EventHandling/Conceptual/EventHandlingiPhoneOS/EventHandlingiPhoneOS.pdf>. [Accessed 26 04 2014].
- [44] Z. Fitz-Walter, S. Jones and D. Tjondronegoro, "Detecting Gesture Force Peaks for Intuitive Interaction," in *Proceedings of the 5th Australasian Conference on Interactive Entertainment - IE '08*, Brisbane, Australia, 2008.
- [45] M. Nacenta, Y. Kamber, Y. Qiang and P. Kristensson, "Memorability of Pre-designed & User-defined Gesture Sets," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI 2013*, Paris, France, 2013.
- [46] U. Oh and L. Findlater, "The challenges and potential of end-user gesture customization Factors in Computing Systems," in *Proceedings of the SIGCHI Conference on Human*, Paris, France, 2013.

- [47] S. Zhai, P. Kristensson, C. Appert, T. Andersen and X. Cao, "Foundational Issues in Touch-Screen Stroke Gesture Design - An Integrative Review," *Foundations and Trends in Human-Computer Interaction*, vol. 5, no. 2, pp. 97-205, 2012.
- [48] B. Signer, U. Kurmann and M. Norrie, "iGesture: A General Gesture Recognition Framework," in *9th Conf. on Document Analysis and Recognition*, Brazil, 2007.
- [49] D. Willems, R. Niels, M. van Gerven and L. Vuurpijl, "Iconic and multi-stroke gesture recognition," *Pattern Recognition*, vol. 42, no. 12, pp. 3303-3312, 2009.
- [50] L. Anthony and J. O. Wobbrock, "A Lightweight Multistroke Recognizer for User Interface Prototypes," *Proceedings of Graphics Interface (GI '10)*, pp. pp. 245-252, 2010.
- [51] J. Wobbrock, A. Wilson and Y. Li, "Gestures without Libraries, Toolkits or Training: A \$1 Recognizer for User Interface Prototypes," in *Proceedings of ACM Symposium on User Interface Software and Technology - UIST 2007*, Newport, Rhode Island, USA, 2007.
- [52] J. Wobbrock and L. Anthony, "\$N-protractor: A fast and accurate multistroke recognizer," in *Proceedings of the 38th Graphics Interface Conference, GI 2012*, Toronto, ON, Canada, 2012.
- [53] R. Vatavu, L. Anthony and J. Wobbrock, "Gestures as Point Clouds: A \$P Recognizer for User Interface Prototypes," in *ICMI'12*, Santa Monica, California, USA, 2012.
- [54] S. Swigart, "Easily Write Custom Gesture Recognizers for Your Tablet PC Applications," 11 2005. [Online]. Available: <https://msdn.microsoft.com/en-us/library/aa480673.aspx>. [Accessed 25 03 2016].
- [55] W. Liu, "Natural user interface- next mainstream product user interface," *IEEE 11th International Conference on Computer-Aided Industrial Design & Conceptual Design (CAIDCD)*, vol. 1, pp. 203-205, 2010.
- [56] P. Zheng and L. Ni, *Smartphone and Next Generation Mobile Computing*, San Francisco, USA: Elsevier, 2006.
- [57] J. Miller and J. Mukerji, *MDA Guide version 1.0.1*, OMG, 2003.
- [58] O. DRAFT, "The MDA Foundation Model," FRI Camb AB Edits, 2010.
- [59] A. Kleppe, J. Warmet and W. Bast, *MDA Explained: The Model Driven Architecture: Practice and Promise*, USA: Addison Wesley, 2003.
- [60] P. Mohagheghi, V. Dehlen and T. Neple, "Definitions and Approaches to Model Quality in Model-Based software development - A review of

- literature,” in *Information and Software Technology Journal*, Butterworth-Heinemann Newton, MA, USA, 2009.
- [61] S. Mellor, S. Kendall, A. Uhl and D. Weise, *MDA Distilled. Principles of Model-Driven Architecture*, Addison-Wesley Professional, 2004.
  - [62] R. France and B. Rumpe, “Model-driven Development of Complex Software: A Research Roadmap,” in *Future of Software Engineering (FOSE'07)*, Minneapolis, MN, 2007.
  - [63] N. Koch, “Transformation techniques in the model-driven development process of UWE,” in *Proceedings of the sixth international conference on Web engineering - ICWE '06*, Palo Alto, California, USA, 2006.
  - [64] J. Tolvanen and S. Kelly, “Model-Driven Development Challenges and Solutions - Experiences with Domain-Specific Modelling in Industry,” in *Proceedings of the 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD 2016)*, 2016.
  - [65] M. Asadi, M. Ravakhah and R. Ramsin, “An MDA-based System Development Lifecycle,” *IEEE Computer Society - Second Asia International Conference on Modelling & Simulation*, pp. 836-842, 2008.
  - [66] N. Chungoora, R. Young, G. Gunendran, C. Palmer, Z. Usman, N. Anjum, A. Cutting-Decelle, J. Harding and K. Case, “A model-driven ontology approach for manufacturing system interoperability and knowledge sharing,” *Computers in Industry Journal*, vol. 64, no. 4, pp. 392-401, 2013.
  - [67] A. Kriouile, T. Gadi and Y. Balouki, “CIM to PIM Transformation: A criteria Based Evaluation,” *International Journal on Computer Technology & Applications*, vol. 4, no. 4, pp. 616-625, 2013.
  - [68] I. Kurtev, “State of the Art of QVT: A Model Transformation Language Standard,” *Applications of Graph Transformations with Industrial Relevance*, vol. 5088, pp. 377-393, 2008.
  - [69] S. Deelstra, M. Sinnema, J. v. Gorp and J. Bosch, “Model Driven Architecture as Approach to Manage Variability in Software Product Families,” in *Proceedings of the Workshop on Model Driven Architectures: Foundations and Applications*, 2003.
  - [70] J. Gray, Y. Lin and J. Zhang, “Automating Change Evolution in Model-Driven Engineering,” *Computer*, vol. 39, no. 2, pp. 51-58, 2006.
  - [71] O. M. G. -. OMG, “MOF Model to Text Transformation Language, v1.0,” Object Management Group Inc., 2008.
  - [72] A. Manoli, J. Muñoz, V. Pelechano and O. Pastor, “Model to Text Transformation in Practice: Generating Code from Rich Associations

- Specifications,” *Lecture Notes in Computer Science - Springer - Advances in Conceptual Modeling - Theory and Practice*, vol. Volume 4231, pp. pp 63-72, 2006.
- [73] L. Rose, N. Matragkas, D. Kolovos and R. Paige, “A Feature Model for Model-to-Text Transformation Languages,” in *MISE 2012*, Zurich, 2012.
- [74] L. D. Spano, A. Cisternino and F. Paternó, “A Compositional Model for Gesture Definition,” *Proceedings of the 4th International Conference HCSE-2012*, pp. 34-52, 2012.
- [75] A. Lascarides and M. Stone, “Formal Semantics for Iconic Gesture,” in *Proceedings of brandial'06, the 10th International Workshop on the Semantics and Pragmatics of Dialogue (SemDial10)*, 2006.
- [76] G. Giorgolo, “A Formal Semantics for Iconic Spatial Gestures,” *Logic, Language and Meaning*, Vols. Lecture Notes in Computer Science, vol 6042 , pp. 305-314, 2010.
- [77] K. Kin, B. Hartmann, T. DeRose and M. Agrawala, “Proton: Multitouch Gestures as Regular Expressions,” *Proceedings of the SIGCHI Conference on Human Factors in Computing System - CHI'12*, pp. 2885-2894, 2012.
- [78] K. Kin, B. Hartmann, T. DeRose and M. Agrawala, “Proton++: A Customizable Declarative Multitouch Framework,” *Proceedings of UIST 2012*, pp. 477-486, 2012.
- [79] L. Spano, A. Cisternino, F. Paternó and G. Fenu, “GestIT: A Declarative and Compositional Framework for Multiplatform Gesture Definition,” in *Proceedings of the 5th ACM SIGCHI symposium on Engineering interactive computing systems - EICS '13*, London, United Kingdom, 2013.
- [80] B. Puype, *Extending the iGesture Framework with Multimodal Gesture Interaction Functionality*, Vrije Universiteit Brussel, 2010.
- [81] Ideum, “GestureML,” 13 12 2016. [Online]. Available: <http://www.gestureml.org/doku.php>. [Accessed 05 07 2017].
- [82] M. Görg, M. Cebulla and S. Rodriguez, “A framework for abstract representation and recognition of gestures in multi-touch applications,” in *3rd International Conference on Advances in Computer-Human Interactions, ACHI 2010*, 2010.
- [83] T. Hachaj and M. Ogiela, “Semantic Description and Recognition of Human Body Poses and Movement Sequences with Gesture Description Language,” in *Computer Applications for Bio-technology, Multimedia, and Ubiquitous City*, 2012.

- [84] N. Group, "wiki.nuigroup.com/Gesture\_Recognition," 27 10 2009. [Online]. Available: [wiki.nuigroup.com/Gesture\\_Recognition](http://wiki.nuigroup.com/Gesture_Recognition). [Accessed 10 07 2016].
- [85] D. Kammer, J. Wojdziak, M. Keck, R. Groh and S. Taranko, "Towards a formalization of multi-touch gesture," *ACM International Conference on Interactive Tabletops and Surfaces - ITS'10*, vol. 3/94, pp. 49-58, 2010.
- [86] H. Lü and Y. Li, "Gesture Coder : A Tool for Programming Multi-Touch Gestures by Demonstration," in *CHI 2012: ACM Conference on Human Factors in Computing Systems*, 2012.
- [87] H. Lü, M. Negulescu and Y. Li, "Gesturermote : Interacting with Remote Displays through Touch Gestures," in *Proceedings of the 2014 International Working Conference on Advanced Visual Interfaces - AVI '14*, Como, Italy, 2014.
- [88] S. Ruffieux, D. Lalanne, E. Mugellini and O. A. Khaled, "Gesture recognition corpora and tools: A scripted ground truthing method," *Computer Vision and Image Understanding*, vol. 131, pp. 72-87, 2015.
- [89] A. C. Long, J. A. Landay and L. A. Rowe, "quill: A Gesture Design Tool for Pen-based User Interfaces," 2009.
- [90] F. Beuvs and J. Vanderdonck, "Designing Graphical User Interfaces Integrating Gestures," *Proceedings of the SIGDOC'12*, pp. 313-322, 2012.
- [91] M. Guimaraes, V. Farinazzo and J. Ferreira, "A Software Development Process Model for Gesture-Based Interface," in *IEEE International Conference on Systems, Man, and Cybernetics*, Seoul, Korea, 2012.
- [92] M. Nielsen, M. Storrang, T. Moeslund and E. Granum, "A Procedure for Developing Intuitive and Ergonomic Gesture Interfaces for Man-Machine Interaction," Aalborg University, Aalborg, Denmark, 2003.
- [93] A. Bragdon, R. Zeleznik, B. Williamson, T. Miller and L. J., "GestureBar: Improving the Approachability of Gesture-based Interfaces," in *Proceedings of the 27th international conference on Human factors in computing systems - CHI 09*, Boston, MA, USA, 2009.
- [94] O. Pastor and J. C. Molina, *Model-Driven Architecture in Practice. A Software Production Environment Based on Conceptual Modeling*, Spain: Springer, 2007.
- [95] R. Deshayes, C. Jacquet, C. Hardebolle, F. Boulanger and T. Mens, "Heterogeneous modeling of gesture-based 3D applications," in *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, 2012.

- [96] J. Coutaz and G. Calvary, "HCI and Software Engineering for User Interface Plasticity," in *The Human-Computer Handbook - Fundamentals, Evolving Technologies, and Emerging Applications*, Julie, A. Jacko ed., CRC Press Taylor and Francis Group, 2012, pp. 1195-1220.
- [97] G. Calvary, D.-P. A., A. Ocelllo, R.-G. P. and M. Riveill, "At the Cross-Roads between Human-Computer Interaction and Model-Driven Engineering," *ARPN Journal of Systems and Software*, vol. 4, no. 3, pp. 64-76, 2014.
- [98] Q. V. J. M. B. B. L. F. M. T. D. Limbourg, "UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces," in *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"*, 2004.
- [99] G. Mori, F. Paternó and C. Santoro, "CTTE: Support for Developing and Analyzing Task Models for Interactive System Design," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 28, no. 8, pp. 797-813, 2002.
- [100] F. Valverde, J. I. Panach and O. Pastor, "An Abstract Interaction Model of a MDA Software Production Method," in *Twenty-Sixth International Conference on Conceptual Modeling - ER 2007 - Tutorials, Posters, Panels and Industrial Contributions*, Auckland, New Zeland, 2007.
- [101] J. Vanderdonckt, "A MDA-Compliant Environment for Developing User Interfaces of Information Systems," *Advanced Information Systems Engineering LNCS in Computer Science*, vol. 3520, pp. 16-31, 2005.
- [102] J. Vanderdonckt, "Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges," in *ROCHI'08*, Iasi, Romania, 2008.
- [103] 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, pp. 289-308, 2003.
- [104] T. Kapteijns, S. Jansen, S. Brinkkemper, H. Houët and R. Barendse, "A Comparative Case Study of Model Driven Development vs Traditional Development: The Tortoise or the Hare," in *4th European Workshop on "From code centric to model centric software engineering: Practices, Implications and ROI*, Enschede, The Netherlands, 2009.
- [105] C. Bunse, H. Gross and C. Peper, "Embedded System Construction - Evaluation of Model-Driven and Component-Based Development



- Approaches,” in *M. R. Chaudron (Ed.) Models in Software Engineering: Workshops and Symposia at MODELS 2008*, Heidelberg, Springer, 2009, pp. 66-77.
- [106] F. Ricca, M. Leotta, G. Reggio, A. Tiso, G. Guerrini and M. Torchiano, “Using UniMod for Maintenance Tasks: An Experimental Assessment in the Context of Model Driven Development,” in *Proceedings on 4th International Workshop on Modeling in Software Engineering (MiSE)*, Zurich, Switzerland, 2012.
- [107] N. Condori-Fernandez, J. I. Panach, A. I. Baars, T. Vos and O. Pastor, “An empirical approach for evaluating the usability of model-driven tools,” *Science of Computer Programming*, vol. 78, pp. 2245-2258, 2013.
- [108] Y. Martinez, C. Cachero and S. Meliá, “Evaluating the Impact of a Model-Driven Web Engineering Approach on the Productivity and the Satisfaction of Software Development Teams,” *Proceedings of 12th International Conference Web Engineering - ICWE 2012*, vol. LNCS 7387, pp. 223-237, 2012.
- [109] Y. Martinez, C. Cachero and S. Melia, “MDD vs Traditional Software Development: a practitioner subjective perspective,” *Information and Software Technology*, vol. 55, no. 2, pp. 189-200, 2013.
- [110] Y. Martinez, C. C. and S. Meliá, “Empirical study on the maintainability of Web applications: Model-driven Engineering vs Code-centric,” *Empirical Software Engineering*, vol. 19, pp. 1887-1920, 2014.
- [111] M. Cervera, M. Albert, V. Torres and V. Pelechano, “On the usefulness and ease of use of a model-driven Method Engineering approach,” *Information and Software Technology*, vol. 50, pp. 36-50, 2015.
- [112] J. I. Panach, S. España, O. Dieste, O. Pastor and N. Juristo, “In search of evidence for model-driven development claims: An experiment on quality, effort, productivity and satisfaction,” *Information and Software Engineering*, vol. 62, no. C, pp. 164-186, 2015.
- [113] M. Morales-Trujillo, H. Oktaba and M. Piattini, “Using Technical-Action-Research to Validate a Framework for Authoring Software Engineering Methods,” in *Proceedings of the 17th International Conference on Enterprise Information Systems*, 2015.
- [114] A. Morali and R. Wieringa, “Risk-based confidentiality requirements specification for outsourced IT systems,” in *Proc. 2010 18th IEEE Int. Requir. Eng. Conf. RE2010*, 2010.
- [115] U. Abelein, “User-Developer Communication in Large Scale IT Projects,” Heidelberg University, 2015.
- [116] V. Antinyan, M. Staron and A. Sandberg, “Validating Software Measures Using Action Research A Method and Industrial

- Experiences," in *Proceedings of the 17th International Conference on Enterprise Information Systems*, 2016.
- [117] A. Milicevic, D. Jackson, M. Gligoric and D. Marinov, "Model-based, Event-Driven Programming Paradigm for Interactive Web Applications," in *OnWard! 2013*, Indiana, USA, 2013.
- [118] S. Sim and R. Gallardo-Valencia, "Introduction: Remixing Snippets and Reusing Components," in *Finding Source Code on the Web for Remix and Reuse*, New York, Springer Science+Business, 2013, p. 348.
- [119] J. Farrell, *An Object-Oriented Approach to Programming Logic and Design*, Boston: Course Technology, 2013.
- [120] S. Mellor, A. Clark and T. Futagami, "Guest Editor'Introduction: Model-Driven Development," *IEEE Software*, vol. 20, no. 5, pp. 14-18, 2003.
- [121] C. Atkinson and T. Kuhne, "Model-Driven Development: A Metamodeling Foundation," *IEEE Software*, vol. 20, no. 5, pp. 36-41, 2003.
- [122] D. Plakalovic and D. Simic, "Applying MVC and PAC patterns in mobile Applications," *JOURNAL OF COMPUTING*, vol. 2, no. 1, pp. 65-72, 2010.
- [123] S. S. Hasan and R. K. Isaac, "An integrated approach of MAS-CommonKADS, Model-View-Controller and web application optimization strategies for web-based expert system development," *Expert Systems with Applications*, vol. 38, pp. 417-428, 2011.
- [124] F. Buschmann, R. Meunier, H. Rohnert and P. Sommerlad, *Pattern-oriented Software Architecture. A System of Patterns*, England: John Wiley and Sons Inc., 2001.
- [125] O. Parra, S. España and O. Pastor, "Including multi-stroke gesture-based interaction in user interfaces using a model-driven method," in *Proceedings of the XVI International Conference on Human Computer Interaction - INTERACCION '15*, Vilanova i la Geltrú (Barcelona), 2015.
- [126] L. Anthony and J. O. Wobbrock, "A Lightweight Multistroke Recognizer for User Interface Prototypes," *Proc. of Graphics Interface*, pp. 245-252, 2010.
- [127] M. Teimourikia, S. Comai, H. Saidinejad and F. Salice, "Personalized Hand Pose and Gesture Recognition System for the Elderly," in *Universal Access in Human-Computer Interaction. Aging and Assistive Environments*, Springer, 2014, pp. 191-202.
- [128] J. Liu, Z. Wang, L. Zhong, J. Wickramasuriya and V. Vasudevan, "uWave: Accelerometer-based personalized gesture recognition and

- its applications,” in *IEEE International Conference on Pervasive Computing and Communications*, Galveston, TX, 2009.
- [129] C. Rolland, “Capturing System Intentionally with Maps,” *Conceptual Modeling in Information Systems Engineering*, pp. 141-158, 2007.
- [130] P. Soffer and C. Rolland, “Combining Intention-Oriented and State-Based Process Modeling,” in *Proceedings Conceptual Modeling - ER2005*, 2005.
- [131] K. Krugler, “Krugle Code Search Architecture,” in *Finding Source Code on the Web for Remix and Reuse*, New York, USA, Springer Science+Business, 2013, p. 348.
- [132] I. Sommerville, *Software Engineering*, Boston, USA: Addison-Wesley, 2011.
- [133] D. Moody, “The Method Evaluation Model: A Theoretical Model for Validating Information Systems Design Methods,” in *ECIS 2003 Proceedings*, Naples, Italy, 2003.
- [134] ISO/IEC, “Ergonomics of human-system interaction,” 28 09 2015. [Online]. Available: <https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-1:v1:en>. [Accessed 10 11 2016].
- [135] ISO/IEC/JTC 1/SC 7, ISO/IEC 25062:2006, *Software engineering - Software product Quality Requirements and Evaluation (SQuaRE) - Common Industry Format (CIF) for usability test reports*, Geneva: ISO, 2006.
- [136] E. de Queiroz, J. Fachine and A. Barbosa, “Towards a multidimensional approach for the evaluation of multimodal application user interfaces,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2009.
- [137] F. D. Davis, “Perceived Usefulness, Perceived Ease of Use, and User Acceptance,” *MIS Quarterly*, vol. 13, pp. 319-339, 1989.
- [138] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell and A. Wesslèn, *Experimentation in Software Engineering*, Berlin: Springer, 2012.
- [139] N. Juristo and A. Moreno, *Basics of Software Engineering Experimentation*, Springer US, 2001.
- [140] D. Kieras, “Using the Keystroke-Level Model to Estimate Execution Times,” University of Michigan, Michigan, USA, 2001.
- [141] S. Card, A. Newell and T. Moran, *The Psychology of Human-Computer Interaction*, Hillsdale, NJ, USA: L. Erlbaum Associates Inc., 1983.
- [142] J. H. Kim and R. C. Miller, “6.813/6.831 User Interface Design,” MIT, 02 09 2009. [Online]. Available:

<http://courses.csail.mit.edu/6.831/2009/handouts/ac18-predictive-evaluation/klm.shtml>. [Accessed 28 10 2015].

- [143] P. Runeson, "Using Students as Experiment Subjects – An Analysis on Graduate and Freshmen Student Data," *Proceedings 7th International Conference on Empirical Assessment & Evaluation in Software Engineering*, pp. 95-102, 2003.
- [144] M. Svahnberg, A. Aurum and C. Wohlin, "Using students as subjects - an empirical evaluation," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*, Kaiserslautern, Germany, 2008.
- [145] F. Faul, E. Erdfelder, A. G. Lang and A. Buchner, "G\*Power3: A flexible statistical power analysis program for the social, behavioural, and biomedical sciences," *Behavior Research Methods*, vol. 39, pp. 175-191, 2007.
- [146] K. Maxwell, *Applied Statistics for Software Managers*, Prentice-Hall, 2011.
- [147] H. Boone and D. Boone, "Analyzing Likert Data," *Journal of Extension. Sharing Knowledge, Enriching Extension*, vol. 50, no. 2, 2012.
- [148] J. Kotrlik, "The Incorporation of Effect Size in Information Technology, Learning, and Performance Research," *Information Technology, Learning, and Performance Journal*, vol. 21, no. 1, pp. 1-7, 2003.
- [149] D. Lakens, "Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs," *Frontiers in Psychology*, vol. 4, no. Article 863, pp. 1-12, 2013.
- [150] R. Wieringa and A. Morali, "Technical Action Research as a Validation Method in Information Systems Design Science," in *7th International Conference - DESRIST 2012*, Las Vegas, USA, 2012.
- [151] S. Bērziša, G. Bravos, T. Cardona, U. Czubayko, S. España, J. Grabis, M. Henkel, L. Jokste, J. Kampars, H. Koç, J. Kuhr, C. Llorca, P. Loucopoulos, R. Juanes, O. Pastor, K. Sandkuhl, H. Simic, J. Stirna, F. Valverde and J. Zdravkovic, "Capability Driven Development: An Approach to Designing Digital Enterprises," in *Business & Information Systems Engineering*, 2015.
- [152] S. España, T. González, J. Grabis, L. Jokste, R. Juanes and F. Valverde, "Capability-driven development of a SOA platform: a case study," in *First International Workshop on Advances in Services Design based on the Notion of Capability (ASDENCA 2014)*, 2014.
- [153] M. Schrepp, A. Hinderks and J. Thomaschewski, "Applying the User Experience Questionnaire (UEQ) in Different Evaluation Scenarios," *Proceedings of the Third International Conference, DUXU 2014, Held as Part of HCI International 2014*, vol. 8517, pp. 383-392, 2014.

- [154] C. Barnum and L. Palmer, "Tapping into Desirability in User Experience," in *Usability of Complex Information Systems: Evaluation of User Interaction*, Boca Raton FL, CRC Press, 2011, pp. 253-279.
- [155] H. Santoso, M. Schrepp, R. Kartono Isal, A. Yudha Utomo and B. Priyogi, "Measuring User Experience of the Student-Centered e-Learning Environment," *The Journal of Educators Online-JEO*, vol. 13, no. 1, pp. 58-79, 2016.
- [156] M. Rauschenberger, M. Schrepp, M. Pérez Cota, S. Olschner and J. Thomaschewski, "Efficient Measurement of the User Experience of Interactive Products. How to use the User Experience Questionnaire (UEQ). Example: Spanish Language Version," *International Journal of Artificial Intelligence and Interactive Multimedia*, vol. 2, no. 1, pp. 39-45, 2013.
- [157] A. Nawaz, J. L. Helbostad, L. Chiari, F. Chesani and L. Cattelani, "User Experience (UX) of the Fall Risk Assessment Tool (FRAT-up)," in *IEEE 28th International Symposium on Computer-Based Medical Systems*, Sao Carlos, 2015.
- [158] B. Laugwitz, T. Held and M. Schrepp, "Construction and Evaluation of a User Experience Questionnaire," in *HCI and Usability for Education and Work USAB 2008. Lecture Notes in Computer Science*, Berlin, Heidelberg, 2008.
- [159] S. Adikari, C. McDonald and J. Campbell, "Quantitative Analysis of Desirability in User Experience," in *Australasian Conference on Information Systems*, Adelaide, 2015.
- [160] J. Stirna, J. Grabis, M. Henkel and J. Zdravkovic, "Capability Driven Development – An Approach to Support Evolving Organizations," in *PoEM 2012*, 2012.
- [161] T. Merčun, "Evaluation of information visualization techniques: analysing user experience with reaction cards," *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pp. 103-109, 2014.
- [162] L. Vogel, Eclipse Rich Client Platform. The complete guide to Eclipse application development, Lars Vogel, 2015.
- [163] P. Kardasis and P. Loucopoulos, "Expressing and organising business rules," *Information and Software Technology*, vol. 46, pp. 701-718, 2004.
- [164] S. Jamieson, "Likert scales: How to (ab)use them," *Medical Education*, vol. 38, pp. 1217-1218, 2004.



## VALENCIA, SPAIN

The research reported and discussed in this thesis represents gestUI, a novel approach to define custom gestures and to include gesture-based interaction in user interfaces of the software systems with the aim of help to solve the problems found in the related literature about the development of gesture-based user interfaces.

The design and implementation of gestUI are presented following the Model-driven Paradigm. Then, we performed two evaluations of gestUI: (i) an empirical evaluation and (ii) a Technical Action Research.

Finally, contributions of our thesis, limitations of the tool support and the approach are discussed and further work are presented.

## CUENCA, ECUADOR

