

Including multi-stroke gesture-based interaction in user interfaces using a model-driven method

Otto Parra González¹

Universidad de Cuenca
Av. 12 de Abril s/n y A. Cueva
01.01.168, Cuenca, Ecuador
+593 07 405 1000

otto.parra@ucuenca.edu.ec

Sergio España

¹Universitat Politècnica de València
Camino de Vera s/n
46022, Valencia, Spain
+34 96 387 7000

sergio.espana@dsic.upv.es

Oscar Pastor

Universitat Politècnica de València
Camino de Vera s/n
46022, Valencia, Spain
+34 96 387 7000

opastor@dsic.upv.es

ABSTRACT

Technological advances in touch-based devices now allow users to interact with information systems in new ways, being gesture-based interaction a popular new kid on the block. Many daily tasks can be performed on mobile devices and desktop computers by applying multi-stroke gestures. Scaling up this type of interaction to bigger information systems and software tools entails difficulties, such as the fact that gesture definitions are platform-specific and this interaction is often hard-coded in the source code and hinders their analysis, validation and reuse. In an attempt to solve this problem, we here propose gestUI, a model-driven approach to the multi-stroke gesture-based user interface development. This system allows modelling gestures, automatically generating gesture catalogues for different gesture-recognition platforms, and user-testing the gestures. A model transformation automatically generates the user interface components that support this type of interaction for desktop applications (further transformations are under development). We applied our proposal to two cases: a form-based information system and a CASE tool. We include details of the underlying software technology in order to pave the way for other research endeavours in this area.

Categories and Subject Descriptors

H.5.2 [Information Interfaces and Presentation]: User interfaces. – Graphical User Interfaces, Interaction Styles, Input devices and strategies.

General Terms

Design, Experimentation, Human Factors, Verification.

Keywords

Customised gesture, model-driven engineering, user interface, gesture-based interaction.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.

Interacción '15, September 07-09, 2015, Vilanova i la Geltrú, Spain
© 2015 ACM. ISBN 978-1-4503-3463-1/15/09...\$15.00
DOI: <http://dx.doi.org/10.1145/2829875.2829931>

1. INTRODUCTION

Natural user interface (NUI) considers the use of many different interaction modalities, including multi-touch, motion tracking, voice, and stylus [1]. Users interact with computers employing intuitive action such as touching, gesturing and speaking.

Several issues may hinder the wide adoption of gesture-based interaction in complex information systems engineering. Currently, gesture-based interaction is limited to its specification in the implementation stage in the software development lifecycle (SDLC) using tools to write source code. Gesture-based interfaces have been reported to be more difficult to implement and test than traditional mouse and pointer interfaces [2], yet the development of gesture-based capabilities has to be done entirely in the source code by programmers with specific technical skills [3].

With the aim of overcoming this situation, this paper introduces a Model-Driven Development (MDD) approach to gesture-based information systems interface development. The method is intended to allow software engineers to focus on the key aspects of gesture-based information system interfaces; namely, defining gestures and specifying gesture-based interaction. Coding and portability efforts are alleviated by means of model-to-text (M2T) transformations. We focus on multi-stroke gestures as they are expressive and currently wide-spread.

Our proposal has the following benefits: (a) the solution is integrated with the existing source code of user interfaces; (b) the solution can be used on any target platform (platform-independence).

We also implemented a tool prototype to support our approach that defines multi-strokes gestures and includes them in a user interface. This tool prototype is coded in Java and Eclipse Modelling Framework.

The contributions of this paper are the following:

- We present gestUI, an MDD method for gesture-based IS interface development consisting of: (i) a modelling language to represent multi-stroke gestures, and (ii) a set of multi-platform model transformations.
- We provide a tool support for the method that captures multi-stroke gestures sketched by the users, transforms them into a model, and automatically generates (i) the gesture catalogue and (ii) the source code of the gesture-based IS interface.
- We validate our approach by applying it to two cases in different types of information systems for desktop-computing.

This paper is organized as follows: Section 2 includes the definitions of items contained in the proposal. Section 3 reviews

related work. Section 4 describes our proposal. Section 5 introduces the tool prototype designed to apply this approach. Section 6 describes its application in two cases and Section 7 contains our conclusions and projected future work.

2. BACKGROUND

2.1 Stroke gestures

Nacenta et al. [4] 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 [5]. According to the taxonomy of gestures in [6], 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 (see Fig. 1).

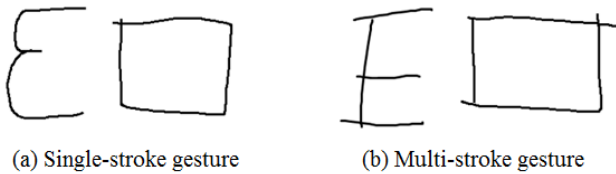


Fig. 1. Types of semaphoric gestures

According to [7] 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. In this work, stroke gestures are used to issue commands, which are the names of the executable computing functions issued by the user.

2.2 Model related definitions

A model is a description of a system or part of a system written in a well-defined language. A well-defined language is a language with well-defined form (syntax), and meaning (semantics), which is suitable for automated interpretation by a computer [8]. A model, both source and target, is expressed in a language [9], for example, XML.

Model-Driven Architecture® (MDA®) is an architectural framework for software development. One of its fundamental aspects is its ability to address the complete development lifecycle, covering analysis and design, programming, testing, component assembly as well as deployment and maintenance. MDA specifies three default viewpoints on a system: computation independent, platform independent and platform specific. MDA specifies three default models of a system corresponding to these three viewpoints: computation independent model (CIM), platform independent model (PIM) and a platform specific model (PSM) [10].

Model-Driven Engineering (MDE) describes software development approaches in which abstract models of software are created and systematically transformed into concrete implementations [11]. Model-driven development (MDD) automates the transformation of models from one form to another.

Model-driven describes an approach to software development whereby models are used as the primary source for documenting,

analysing, designing, constructing, deploying and maintaining a system [10].

Model-to-Text (M2T) transformation is a model management operation that involves generating text (e.g., source code, documentation, configuration files) from models [12].

3. RELATED WORK

This section briefly reviews the role of model-driven engineering (MDE) in HCI 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.

Aquino et al. [13] 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 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. [14] 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.

Vanderdonckt [15] describes a set of variables related to the development of user interfaces, one of which considers interaction devices and styles. Interaction styles include the gesture recognition. However, he point out that an abstract user interface is independent of any interaction modality [16] so that an explicit reference to a specific type of interaction style is not considered.

In [17] is included 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 [18] 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, CTTe), one being the interaction model considered in the process of defining user interfaces. However, the interaction modality is not considered.

In [19], the authors 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.

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.

4. OUR PROPOSAL

This section describes gestUI [20], a user-driven iterative method based on the MDD paradigm. The system defines multi-strokes gestures, creates a gesture catalogue model and using model transformations to obtain the interface source code, including gesture-based interaction. gestUI is expected to be integrated into a full interface development method (shown in Fig. 2 by generic activities and artefacts in grey). The method can either be model-driven or code-centric. gestUI is user-driven because the users participate in all non-automated activities and is iterative because it aims to discover the necessary gestures incrementally and provides several loopbacks.

Our proposal is based on Model-driven Architecture (MDA) considering three layers: the platform-independent layer, platform-specific layer and code layer. gestUI consists of five activities and their products which are distributed in the three layers, as shown in Fig. 2.

In the platform-independent layer, during the A1 activity, “Define gestures”, the stakeholders and developers have meetings to discuss gesture definition for the user interface. The stakeholders try the gestures in the interface implemented in the tool to verify that these gestures are suitable for the tasks in their information

system. Finally, a gesture catalogue model is obtained conforms to the gesture catalogue metamodel (Fig. 3). In the gesture catalogue each gesture is formed by one or more strokes defined by postures, described by means of coordinates (X, Y). The sequence of strokes in the gesture is specified by means of precedence order. Each posture in a gesture is related to a figure (line, circle, etc.) with an orientation (up, down, left, right), and is qualified by a state (initial, executing, final).

In the platform-specific layer there are two activities: (i) Activity A2, “Design gesture-based interaction” which consists of a process to define the relationship between a gesture and an action (command) to be executed in an information system. This activity produces the gesture-based interaction model. If the stakeholders consider that the definition of gestures is incomplete or inadequate the process can return to Activity 1 to be redefined. (ii) Activity A3, “Generate gesture specification”, employs an M2T transformation to generate a platform-specific gestures catalogue specification.

There are two activities in the code layer: (i) Activity A4, “Generate gesture-based interface”, which employs the gesture-based interaction model and the platform-specific gestures catalogue specification to obtain the source code of the gesture-based user interface by applying a model transformation. (ii) Activity 5, “Test gestures”, tests the gestures defined by the stakeholders and included in the information system. This is an iterative process because it is possible to redefine the gestures if the users are not satisfied with their definition.

If the gesture catalogue is complete and the users agree with the definition of the correspondence gesture-action in the information system (IS), then the process is finished. The IS then contains gesture-based interaction as a complementary interaction modality to the traditional interaction.

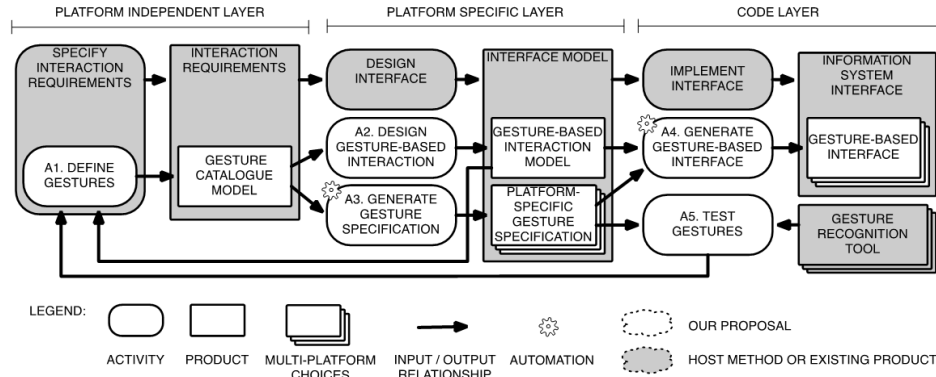


Fig. 2. gestUI method overview

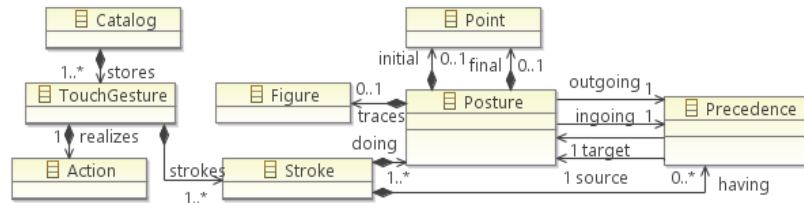
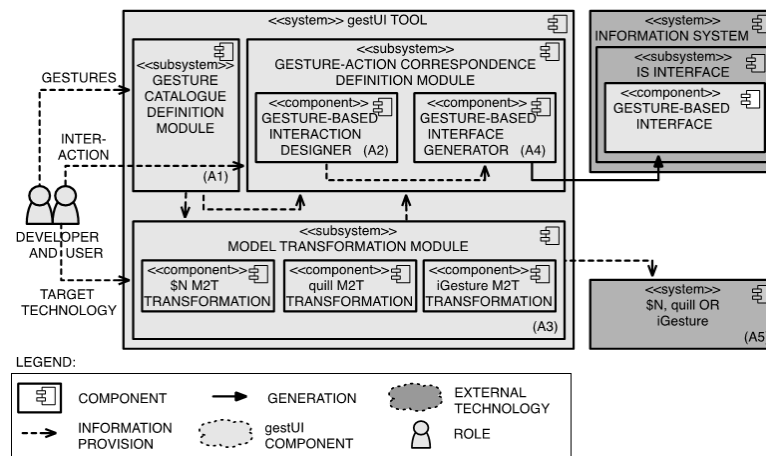


Fig. 3. Gesture catalogue metamodel



5. TOOL

This section describes the tool that implements our proposal. This tool is structured in three modules as shown in Fig. 4. The number in brackets indicates the method activity each component supports. The method’s internal products are not shown. The relationship with the external gesture recogniser is represented.

The first module “Gesture catalogue definition” contains a process that obtains the gesture catalogue model from the users’ gesture definition made by the users. Activity A1 is included in this module.

The second module “Gesture-Action correspondence definition” includes a process to specify the correspondence between gestures and actions (commands) available in the IS. Activities A2 and A4 are included in this module.

The third module “Model transformation” contains the definition of the M2T transformations for some platforms as targets for gesture-based interaction in the user interface. Activity A3 is included in this module.

The tool prototype is developed in Java and Eclipse and permits the proposal to be applied by means of its modules: (i) to define multi-strokes gestures, (ii) to obtain a gesture catalogue model using model transformations, (iii) to define gesture-action correspondence and, (iv) to generate source code to include gesture-based interaction in user interfaces. At present, the tool generates Java source code embedded into the IS interface source code.

6. DEMONSTRATING THE PROPOSAL

This section describes an application of the method with the implemented tool to include gesture-based interaction in existing information systems (IS). *gestUI* is integrated into a code-centric interface development method considering two legacy desktop-computing IS: forms-based IS and case tools.

The process of obtaining a user interface with gesture-based interaction involves the following steps:

- a. Define the gestures according to the users' requirements, and then obtain the gesture catalogue model.

- b. Obtain the platform-specific gesture catalogue by means of model transformation using the gesture catalogue model and the target platform as source for the transformation.
- c. Specify the user interface to which we will add the gesture-based interaction by definition of gesture-action correspondence. The user interface source code which includes the gesture-based interaction is generated.
- d. Add the generated source code to the existing user interface source code and recompile the system.

6.1 Form-based Information System

In the first case, we consider a form-based IS to manage information for the computing department of a business company, formed by departmental offices that have offices. In each department there are employees who use computers with one or more operating systems and printers.

The IS manages the information of: (i) the company, (ii) departments, (iii) devices (computers, printers), (iv) the operating systems installed in the computers, (v) the operation status of the devices, (vi) the employee that uses the devices. For the sake of brevity, we will only consider the option to manage information of operating systems in the process of inclusion of gesture-based interaction. Fig. 5 shows the domain class diagram of this IS.

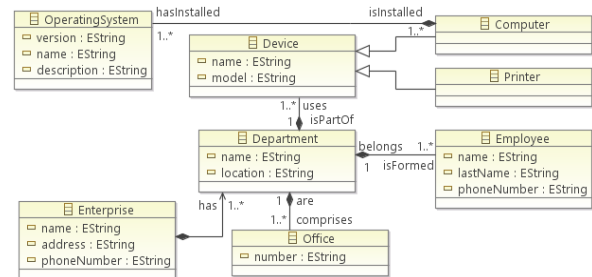


Fig. 5. Domain class diagram of the form-based IS

This software, written in Java using Swing in Eclipse, employs a three-tier architecture: data access logic, business logic, and presentation logic [20]. The set of operations that can be executed in each component are called CRUD operations (create, read, update, and delete). In this case we need to define gestures in

order to execute these operations in each option (component) of the IS.

The first step is the creation of new gestures using the interface implemented for this task, (see in Fig. 6). The users define gestures according to their preferences in order to execute actions in an information system. For instance, in a database (a) C, to create a new record to enter in the IS, (b) U, to update the information of a record, (c) D, to delete a record, and (d) R, to read the information of a record. Using these gestures, we obtain the gesture catalogue model by means of a model transformation.

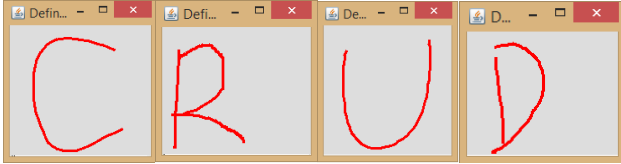


Fig. 6. Gestures defined for form-based IS

Therefore, in order to define the gesture catalogue model the user selects gestures from a gesture repository containing all the gestures that the user has defined. This catalogue contains the description of each gesture, but does not contain the action (command) definition to execute it (in Fig. 7 the action is assigned to “null”) because this feature (to assign an action) is platform-specific must be specified in the next step.

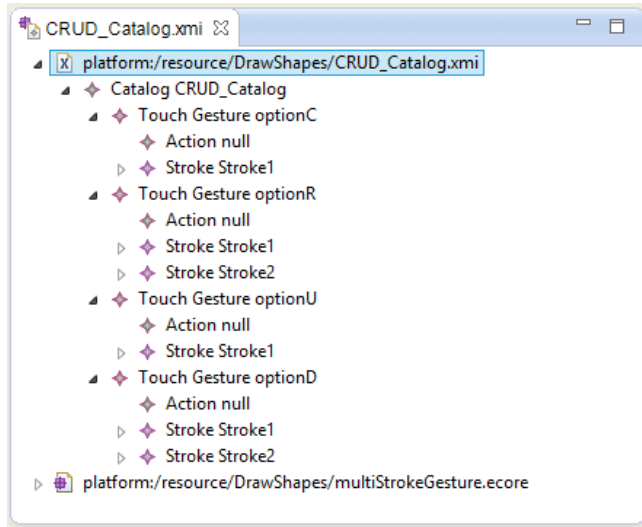


Fig. 7. Gesture catalogue model

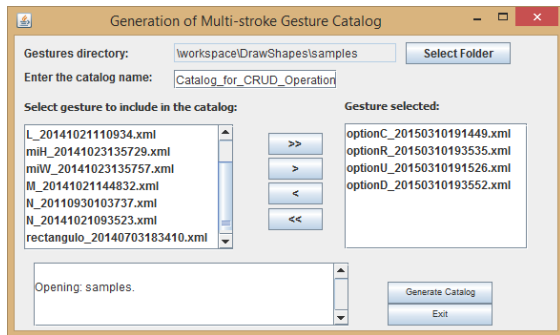


Fig. 8. Obtaining platform-specific gesture specification

The next step is the generation of platform-specific gesture specification by choosing the gestures that are used in the IS (Fig. 8). Using this specification and entering the target platform (\$N) and the target language (XML) as input data we apply a model transformation to obtain the gestures to be used in the gesture recognition process using \$N [21].

Fig. 9 shows an excerpt from a gesture definition generated in XML to be used with \$N. In this case, the gesture which represents the letter “D” is formed by two strokes (shown in Fig. 6), each of which contains points with coordinates (X, Y) and a timestamp (T).

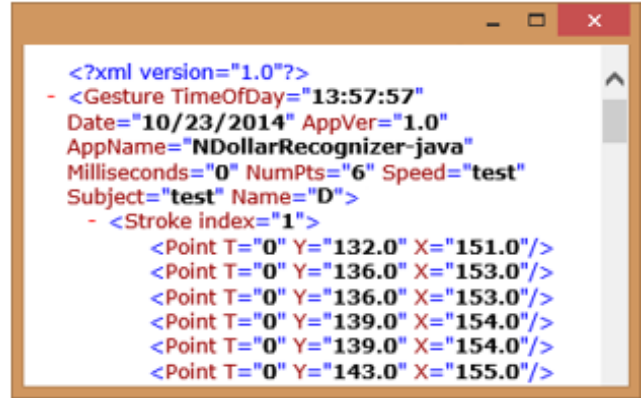


Fig. 9. An excerpt from a gesture written in XML

The next step concerns with the definition of the gesture-action correspondence in order to complete the gesture definition. In this case, the process consists of the selection of a user interface source code to apply a parsing process and determine the actions to be included in this source code. We consider code which contains methods in Java containing “action perform” structures (Fig. 10), which typically define the actions to be executed by selecting buttons or options in the menu in a form-based IS.

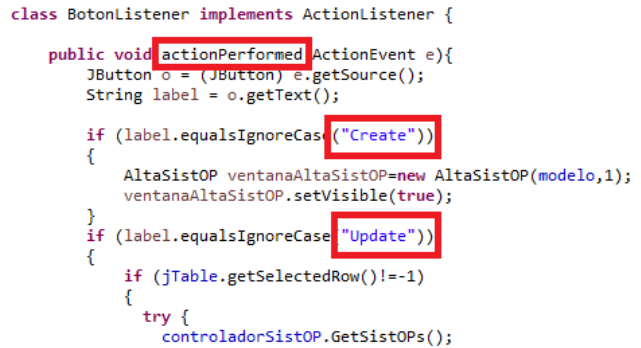


Fig. 10. An excerpt of source code of user interface

The developer assign the gesture-action correspondence in collaboration with the user, supported by the Gesture-action correspondence definition module. The process consist in select one gesture (Fig. 11, left) for each action specified in the user interface (Fig. 11, centre). The correspondence gesture-action is defined (Fig. 11, right). The next step is the source code generation by selecting button “Generate” in the interface shown in Fig. 11. In this case, we obtain Java source code to include gesture-based interaction in the user interface source code

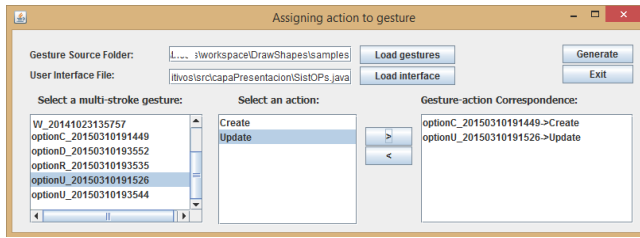


Fig. 11. Gesture-Action correspondence definition

When generating the user interface Java source code, many references are included: (i) references to libraries to manage gestures (e.g. listeners to manage mouse events such as move, down, up, release, drag), (ii) references to the libraries of the gesture-recognition technology (e.g. \$N), (iii) references to the method of executing the gesture-action correspondence (i.e. executingActions in Fig. 12), (iv) references to the methods of capturing gestures (i.e. mouseDragged, mouseReleased), and (v) references to the method of loading the multi-stroke gesture catalogue.

```
void executeActions(String gesture){
    if (gesture.equals("C")) {
        AltaSistOP ventanaAltaSistOP=new AltaSistOP(modelo,1);
        ventanaAltaSistOP.setVisible(true);
    }
    else
        if (gesture.equals("U")) {
            ActualizaSistOP ventanaActualizaSistOP=
                new ActualizaSistOP(modelo,1);
            ventanaActualizaSistOP.setVisible(true);
        }
}
```

Fig. 12. An excerpt of source code to execute actions

Additionally, the definition of some classes have been changed in order to implement `MouseListener` and `MouseMotionListener` to detect mouse events.

Finally, when we execute the form-based IS (Fig. 13), we use the gestures included in the source code to execute the actions specified in the process. In this case, we can use traditional interaction and gesture-based interaction.

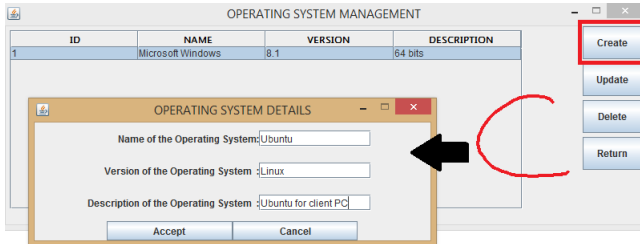


Fig. 13. Form-based IS with gesture-based interaction

In the form-based IS shown in Fig. 13, the user draws the gesture “C” with the aim of creating a new operating system record. When the gesture recognizer analyses the gesture sketched by the user as correct, then a new window for the user to fill in the form related to the operating system details.

6.2 Case tool

Now, we describe the application to the second type of IS: in this case, we consider Graphical Editing Framework (GEF), an Eclipse project which provides a development framework for graphical representation of information. The GEF project is developed in

JFace and Standard Widget Toolkit (SWT) with Eclipse Modelling Framework. GEF is designed with a Model-View-Controller (MVC) architecture. Briefly, the function of each component is [22]:

- **Model:** holds the information (underlying objects being represented graphically) to be displayed and is persisted across sessions.
- **View:** renders information on the screen and provides basic user interaction, in this case, a canvas containing a collection of figures.
- **Controller:** coordinates the activities of the model and the view, passing information between them as necessary. The controller has a collection of GEF edit parts.

According to [23], the GEF project is formed by three subsections:

- **Draw2D,** a lightweight framework layered on top of SWT for rendering graphical information.
- **GEF Framework,** an MVC framework layered on top of Draw2D to facilitate user interaction with graphical information.
- **Zest,** a graphing framework layered on top of Draw2D for graphing.

In this work, we employ the example called Shapes, included in GEF project, to apply the method. Considering that GEF has a plug-in structure, then the Shapes source code is comprised by two components: (i) source code of GEF which provides the functionality of an editor (to display and draw elements in a diagram), and, (ii) source code of the example, which define the canvas and the palette of the tool and employs the methods defined in the plug-in to use the editor. The palette defined in Shapes permits rectangles and ellipses to be drawn. Therefore, we need to modify the View component both in the source code of GEF and the example Shapes, in order to define the finger gestures for drawing ellipses and rectangles.

We describe below the process applied to the GEF framework in order to include gesture-based interaction in this case tool.

Table 1. Gestures and elements related


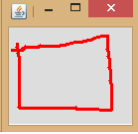
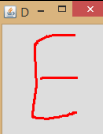
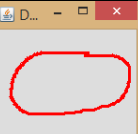
Gestures		Element related
		Rectangle
		
		Ellipse

Table 1 shows the gestures used to draw the elements defined in the Shapes palette. In the process of defining gestures, in some cases it is possible to “reuse” gestures that the users have sketched previously, for instance, we use the gesture “C” to execute different actions: (i) in the form-based IS to create a record, and (ii) in the case tool to draw a rectangle.

Using the process described in this paper, we obtain the gesture catalogue model, then in the next step we get the platform-specific

gesture specification and finally, the definition of gestures using XML to use in the \$N gesture recognizer.

The gesture-action correspondence definition is executed in the source code of Shapes. Fig. 14 includes an excerpt from the source code that contains the reference to the aforementioned actions: draw rectangles and ellipses.

```
/**
 * Return a IFigure depending on the instance of the current model element.
 * This allows this EditPart to be used for both subclasses of Shape.
 */
private IFigure createFigureForModel() {
    if (getModel() instanceof EllipticalShape) {
        return new Ellipse();
    } else if (getModel() instanceof RectangularShape) {
        return new RectangleFigure();
    }
}
```

Fig. 14. An excerpt of source code of Shapes

We apply the process of parsing in this source code to determine the actions defined to draw the elements contained in the palette. The user selects the gestures in the catalogue (Fig. 15, left), and defines the relation with the actions that were found in the parsing process (Fig. 15, centre). Finally, the gesture-action correspondence is obtained (Fig. 15, right).

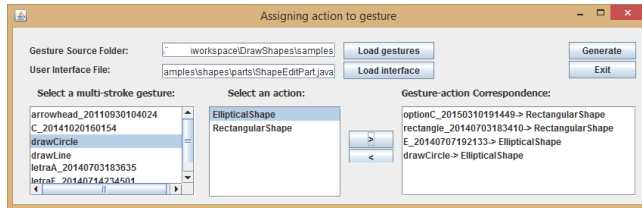


Fig. 15. Gesture-action correspondence definition

When generating the user interface Java source code, many references are included; for instance, in the GEF source code we add references to gesture management libraries (e.g. listeners to manage mouse events such as mouseUp and mouseDrag), to the gesture-recognition technology libraries (e.g. \$N) and the method of executing the gesture-action correspondence (Fig. 16).

Also, in the methods for managing mouse events we add source code to capture gestures (i.e. mouseDrag (Fig. 17), mouseUp), and the method of loading the multi-stroke gesture catalogue is added.

```
private IFigure createFigureForModel() {
    if (getModel() instanceof EllipticalShape) {
        return new Ellipse();
    } else if (getModel() instanceof RectangularShape) {
        return new RectangleFigure();
    } else if (getModel() instanceof GestureShape) {
        if (gestureType.equalsIgnoreCase("1")) // (C o rectangle)
            return new RectangleFigure();
        else if (gestureType.equalsIgnoreCase("2")) // circle or E
            return new Ellipse();
        else
            throw new IllegalArgumentException();
    }
}
```

Fig. 16. An excerpt of source code to execute actions

```
public void mouseDrag(MouseEvent mouseEvent, EditPartViewer viewer) {
    Tool tool = getActiveTool();
    if (tool != null) {
        if (isGesture == 1)
            ManagingGestureAndMouse.detectingMouseDrag(mouseEvent);
        tool.mouseDrag(mouseEvent, viewer);
    }
}
```

Fig. 17. An excerpt of the source code of GEF

We then consider the source code of the example in order to add the option of including gesture-based interaction in the palette. In the source code containing the definition of the palette we add the

source code shown in Fig. 18. The option added is shown in a red circle in Fig. 19. The users thus have two styles of interaction in GEF: (i) traditional interaction (using mouse and keyboard) and (ii) gesture-based interaction (using fingers to sketch gestures).

```
*)ShapesEdito... GestureShape... LightweightS... AbstractGrap... DiagramEditP...
96 tool = new GestureTemplateCreationEntry("Gesture",
97 "Create a shape with gestures", GestureShape.class,
98 new SimpleFactory(GestureShape.class),
99 ImageDescriptor.createFromFile(ShapesPlugin.class,
100 "icons/gesture16.png"), ImageDescriptor.createFromFile(
101 ShapesPlugin.class, "icons/gesture24.png"));
102 toolbar.add(tool);
103 palette.setDefaultEntry(tool);
```

Fig. 18. An excerpt from the source code to modify the palette

Obviously, then the source code should be compiled to obtain the case tool with gesture-based interaction included.

When the user employs the gestures defined in this case tool to draw an ellipse/rectangle on the canvas using his/her finger, the gesture recognizer analyses the sketch. If it is correct, then the system draws an ellipse/rectangle defined in the GEF palette.

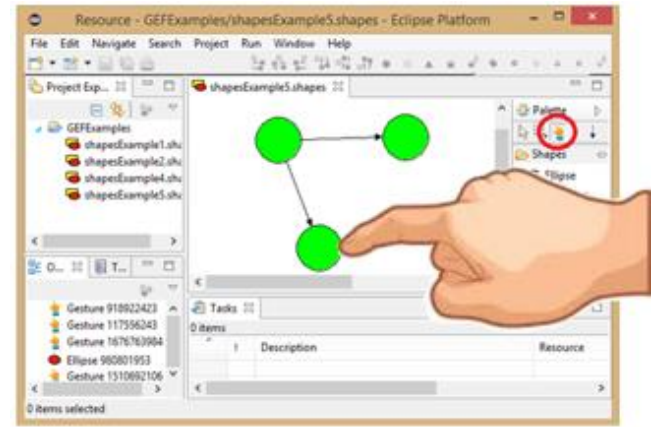


Fig. 19. Case tool with gesture interaction

7. CONCLUSIONS AND FUTURE WORK

gestUI is a model-driven method for developing multi-stroke gesture-based user interfaces. We validated the method and supporting tools by applying them to two types of applications: a form-based application and a CASE tool built using GEF (an Eclipse framework with the structure of a plug-in) that can be used to draw simple diagrams. We generated the 'Platform-specific gesture specification' for the two types of applications to illustrate the multiplatform capability of the approach. The gestures were successfully recognised by the corresponding tools. We then automatically generated the final gesture-based interface components and integrated them into the application interfaces.

The advantages of the gestUI proposal are: its platform independence enabled by the model-driven development paradigm, the convenience of including user-defined symbols and its iterative and user driven approach.

Future work will be developed along the following lines: (i) a user study to determine user preferences in defining gestures according to the task to be executed, (ii) applying this method to extending the Capability Design Tool (CDT) of the "Capability as a Service" Project (CaaS project) in order to incorporate gesture-based interaction into this framework.

8. ACKNOWLEDGMENTS

Otto Parra is grateful to his supervisors Sergio España and Óscar Pastor for their invaluable support and advice. This work has been supported by Secretaría Nacional de Educación, Ciencia y Tecnología (SENESCYT) and Universidad de Cuenca of Ecuador, and received financial support from Generalitat Valenciana under Project IDEO (PROMETEOII/2014/039).

9. REFERENCES

- [1] D. Wigdor y D. Wixon, Brave NUI world: designing natural user interfaces for touch and gesture, USA: Morgan Kaufmann Publishers Inc., 2011.
- [2] M. Hesenius, T. Griebel, S. Gries y 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.
- [3] K. Gerken, S. Frechenhäuser, R. Dörner y J. Luderschmidt, «Authoring Support for Post-WIMP Applications», *Human-Computer Interaction – INTERACT 2013. Lecture Notes in Computer Science*, vol. 8119, pp. 744-761, 2013.
- [4] M. Nacenta, Y. Kamber, Y. Qiang y P. Kristensson, «Memorability of Pre-designed & User-defined Gesture Sets», *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems*, pp. 1099-1108, 2013.
- [5] U. Oh y L. FindLater, «The challenges and potential of end-user gesture customization», *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI'13*, pp. 1129-1138, 2013.
- [6] M. Karam y M. C. Schraefel, «A taxonomy of Gestures in Human-Computer Interaction», de *Retrieved from <http://eprints.soton.ac.uk/261149/>*, 2005.
- [7] S. Zhai, P. O. Kristensson, C. Appert, T. H. Anderson y X. Cao, «Foundational Issues in Touch-Surface Stroke Gesture Design — An Integrative Review», *Foundations and Trends in Human-Computer Interaction*, vol. 5, nº 2, pp. 97-205, 2012.
- [8] A. Kleppe, J. Warmer y W. Bast, MDA Explained: The Model Driven Architecture : Practice and Promise, USA: Addison-Wesley Prof., 2003.
- [9] S. J. Mellor, T. Clark y T. Futagami, «Model-driven development: guest editors' introduction», *IEEE Software*, vol. 20, nº 5, pp. 14-18, 2003.
- [10] F. Truyen, «The Fast Guide to Model-Driven Architecture. The Basics of Model-Driven Architecture», Cephas Consulting Corp., 2006.
- [11] R. France y B. Rumpe, «Model-Driven Development of Complex Software: A Research Roadmap», *Future of Software Engineering - ICSE*, pp. 37-54, 2007.
- [12] B. Ogunyomi, L. M. Rose y D. Kolovos, «On the Use of Signatures for Source Incremental Model-to-text Transformation», *Model-Driven Engineering Languages and Systems. Lecture Notes in Computer Science*, vol. 8767, pp. 84-98, 2014.
- [13] N. Aquino, J. Vanderdonckt, J. I. Panach y O. Pastor, «Conceptual Modelling of Interaction», de *Handbook of Conceptual Modeling. Theory, Practice, and Research Challenges*, Berlin, Springer, 2011, pp. 335-358.
- [14] R. Deshayes, C. Jacquet, C. Hardebolle, F. Boulanger y T. Mens, «Heterogeneous modeling of gesture-based 3D applications», de *Proceedings of the 6th International Workshop on Multi-Paradigm Modeling*, 2012.
- [15] 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.
- [16] J. Vanderdonckt, «Model-Driven Engineering of User Interfaces: Promises, Successes, Failures, and Challenges», de *ROCHI'08*, Iasi, Romania, 2008.
- [17] J. Coutaz y G. Calvary, «HCI and Software Engineering for User Interface Plasticity», de *The Human-Computer Handbook - Fundamentals, Evolving Technologies, and Emerging Applications*, Julie, A. Jacko ed., CRC Press Taylor and Francis Group, 2012, pp. 1195-1220.
- [18] G. Calvary, D.-P. A., A. Ocelllo, R.-G. P. y 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.
- [19] F. Valverde, J. I. Panach y O. Pastor, «An Abstract Interaction Model of a MDA Software Production Method», de *Twenty-Sixth International Conference on Conceptual Modeling - ER 2007 - Tutorials, Posters, Panels and Industrial Contributions*, Auckland, New Zeland, 2007.
- [20] O. Parra, S. España y O. Pastor, «A Model-driven Method and a Tool for Developing Gesture-based Information Systems Interface», de *Proceedings of the CAiSE'15 Forum at the 27th International Conference on Advanced Information Systems Engineering*, Stockholm, Sweden, 2015.
- [21] M. Fowler, Patterns of Enterprise Application Architecture, Boston, MA: Addison-Wesley Professional, 2003.
- [22] L. Anthony y J. O. Wobbrock, «A Lightweight Multistroke Recognizer for User Interface Prototypes», *Proceedings of Graphics Interface (GI '10)*, pp. pp. 245-252, 2010.
- [23] E. Clayberg y D. Rubel, Eclipse Plug-ins, Massachusetts, USA: Addison-Wesley, 2009.