Complex Systems Informatics and Modeling Quarterly
CSIMQ, Issue 6, March/April 2016, Pages X-XX
Published online by RTU Press, https://csimq-journals.rtu.lv
http://dx.doi.org/10.7250/csimq.2016-6.XX
ISSN: 2255-9922 online

gestUI: A Model-driven Method and Tool for Including Gesture-based Interaction in User Interfaces

Otto Parra[1, 3], Sergio España[2] and Oscar Pastor[3]

[1] Computer Science Department, Universidad de Cuenca, Ecuador
[2] Department of Information and Computing Sciences, Utrecht University. The Netherlands
[3] PROS Research Center, Universitat Politécnica de València, Spain

otpargon@posgrado.upv.es, s.espana@uu.nl, opastor@pros.upv.es

Abstract. Among the technological advances in touch-based devices, gesture-based interaction have become a prevalent feature in many application domains. Information systems are starting to explore this type of interaction. As a result, gesture specifications are now being hard-coded by developers at the source code level that hinders their reusability and portability. Similarly, defining new gestures that reflect user requirements is a complex process. This paper describes a model-driven approach to include gesture-based interaction in desktop information systems. It incorporates a tool prototype that captures user-sketched multi-stroke gestures and transforms them into a model by automatically generating the gesture catalogue for gesture-based interaction technologies and gesture-based user interface source codes. We demonstrated our approach in several applications ranging from case tools to form-based information systems.
Keywords: Model-driven architecture, gesture-based interaction, multi-stroke gestures, information systems, gesture-based user interface.

1 Introduction

New devices are now appearing with new types of user interfaces (e.g., interfaces that are based on gaze, gesture, voice, haptic, and brain-computers). Although the aim is to increase the naturalness of the interaction [1], the effort is not exempt from risks. Due to the popularity of touch-based devices, gesture-based interactions are steadily increasing in mouse and keyboard applications as well as for video games and mobile apps. Information systems (IS) are likely to follow the trend, especially in supporting tasks performed outside the office [2].

Several issues may hinder the wider adoption of gesture-based interaction in complex information systems engineering. Gesture-based user interfaces have been reported to be more difficult to implement and test than traditional mouse and pointer interfaces [3]. Gesture-based interaction is supported at the source code level (typically third-generation languages) [4]. This involves an extensive coding and maintenance effort when multiple platforms are targeted. Moreover, it has a negative impact on reusability and portability and also complicates the definition of new gestures. Some of these challenges can be resolved by following a model driven development (MDD) approach, provided that gestures and gesture-based interaction can be modelled and that it is possible to automatically generate the software components that support them.

This paper introduces an MDD approach and a tool for the inclusion of gesture-based interaction in developing user interfaces for information systems which is intended to allow software engineers to focus on the key aspects of these interfaces. Namely, to define the gestures and to specify the gesture-based interaction. Coding and portability efforts are alleviated by means of model-to-model (M2M) transformations using ATL[1] and model-to-text (M2T) transformations by using Acceleo[2].

The contributions of this paper are: (a) gestUI [5], an MDD method for including gesture-based interaction in IS user interface development. The method consists of a modelling language to represent multi-stroke gestures and a set of multi-platform model transformations. (b) We provide a tool that supports the method by capturing the multi-stroke gestures sketched by the users, transforming these gestures into a model, and automatically generating the gesture catalogue and the source code to include gesture-based interaction in IS user interfaces. (c) The approach was evaluated in two stages: (i) by applying it to three different gesture-based interaction technologies; namely, $N [6], quill [7][8] and SiGeR/iGesture [9], and (ii) by applying it to include gesture-based interaction in user interface development process in a forms-based IS.

In this research we applied the design science framework, since the purpose was to gather new knowledge and to design new artefacts (i.e. methods and tools). The research methodology have been structured at regulated cycles [10] to meet the following aims: 1) to perform an initial problem investigation that describes the problem; 2) to provide a solution design suitable for solving the problem; and 3) to verify if the proposed solution satisfies the previously analysed problem.

This paper is organised as follows: Section 2, clarifies the definitions of terms used throughout the paper. Section 3 reviews related research work. Section 4 introduces the gestUI method. Section 5 describes the gestUI tool support system. Section 6 demonstrates the use of the method and tool support system. Section 7 concludes the paper and outlines directions for future work. Appendix A describes the inclusion of gestUI in: (i) MARIA [11], an existing model-driven method for user interface development and (ii) Model-Based User Interface Design (MBUID) Specification [12] defined by World Wide Web Consortium (W3C).

2 Background

In this section general gesture-related concepts are introduced and modelling related definitions are given.

2.1 Stroke-based Gestures

Karam et al. describe a gesture taxonomy and consider semaphoric gestures, which are strokes or marks made with a mouse, pen or finger on a touch-sensitive surface [13].

A stroke-based gesture is defined as the trajectory sketched by a finger or stylus on a touch-sensitive surface that can be further classified into single-stroke or multi-stroke gesture, according to the number of strokes required to sketch it (Figure 1). Note that the multi-stroke gestures categorize single-stroke gestures.

[1] http://www.eclipse.org/atl/

[2] http://www.eclipse.org/acceleo/

Figure 1. Types of stroke-based gestures: single-stroke (left), multi-stroke (right)

A stroke-based gesture is defined by a set of points, each point is represented by coordinates (X, Y) and, optionally, a timestamp (t) [14]. In this work we consider stroke gestures represented by coordinates and a timestamp (X, Y, t) that are used to issue commands, which are the names of the executable computing functions issued by the user.

Gesture-based interaction relies on gestures to select and use the functions provided by applications in touch-based devices.

2.2 Model-driven Related Concepts

In Model-driven development, models are used as the primary source for documenting, analysing, designing, constructing, deploying and maintaining a system [15]. In this paper we focus on software systems.

A model is a formal specification of the function, structure and behaviour of a system within a given context from a specific point of view [15].

A platform is the set of resources on which a system runs. This set of resources is used to implement or support the system [16].

Model Driven Architecture (MDA) is an architectural framework for model-driven development. One of its fundamental aspects is its ability to address the complete development lifecycle, covering analysis and design, programming, testing, and component assembly as well as deployment and maintenance [17]. MDA specifies three default models of a system: a computation independent model (CIM), platform independent model (PIM) and a platform specific model (PSM) [16].

Model transformation is an important activity in Model Driven Engineering (MDE). Model transformation is the process of converting one model to another within the same system employing a model transformation language [15]. Model transformation language is a language envisioned specifically for model transformation. Acceleo is a common model transformation language employed to specify M2T transformations. It offers a template-based language for defining code-generation templates [18] to specify M2T transformations. Acceleo is a code generator based on templates that implement the OMG's M2T specification [19]. ATL (ATLAS Transformation Language) is a language and a toolkit to enable M2M transformations. The field of MDE, provides ways to produce a set of target models from a set of source models. ATL is hybrid model transformation language that allows both declarative and imperative constructs to be used in enabling transformation definition [20].

2.3 Gesture Test Frameworks

We considered three existing gestures test frameworks:

- $N is a lightweight, concise multi-stroke gestures recogniser that uses only simple geometry and trigonometry [6]. Its goal is to provide a useful, concise, easy-to-incorporate multi-stroke recogniser deployable on almost any platform to support rapid prototyping. $N allows the user to define gestures through demonstration.

- Quill supports Rubine, one of the first algorithms to recognise mouse and pen-based gestures [7]. It employs a statistical method of gesture recognition based on a set of 13 geometric features [21]. It has been used for recognising single-stroke gestures like the unistroke or Grafitti alphabets [22]. It also allows the user to define a gesture through demonstration.
- iGesture supports the SiGeR algorithm that classifies gestures based on regular expressions and describes them according to the eight cardinal points and statistical information indicators [9].

3 Related Work

Below we cite some of the most important studies published on gesture representation and gesture-based user interface development.

3.1 Gesture Representation

According to the related literature, there are many types of gesture representation that can be used to incorporate gestures into information systems:

- **Representation based on regular expressions**. A gesture is defined by means of regular expressions formed by elements such as ground terms, operators, symbols, etc. Spano [23] defines a gesture as a declarative and compositional model that presents regular expressions containing ground term elements as well as composition operators based on Petri Nets that describe the gestures. In [24] the authors present a formal semantic analysis of iconic gestures employing a multidimensional matrix whose rows contain values that describe aspects of a gesture's forms. Proton [25] allows a declarative and customised definition of multi-touch gestures using regular expressions composed of touch event symbols. Proton++ is declarative multi-touch framework that includes a custom declarative gesture definition system [26] and is based on the Proton framework. GestIT [27] employs a declarative and compositional approach to define gestures using regular expressions. SiGeR (Simple Gesture Recogniser) describes gestures with eight cardinal points (e.g., N, NE, E) and provides statistical information [9].
- **Representation based on a language specification**. Gesture ML [28] or Gesture Markup Language (GML) is an extensible XML-based language used to define multi-touch gestures that describes interactive object behaviour and the relationships between objects and applications. The Gesture Description Language (GDL) [29] enables body postures and gestures to be described under the assumption that gestures can be partitioned into a sequence of postures. The description is contained in a script written in a proprietary language.
- **Representation based on demonstration**. In this case, developers define gesture by generating the code to represent it, refine it, and, once the developer is satisfied with its definition, include it in an IS. Gesture Coder [30] allows a gesture to be defined by demonstration, tests the generated code, refines it, and, once the developer is satisfied with this definition, incorporates the code into IS. Other solutions in this group are $1 [31], $N [6], and $P [32], which base the definition of single-stroke ($1) and multi-stroke ($N and $P) gestures on the trajectory of a finger or pen. In this case, developers can define gestures, generate the code to represent a gesture, refine it, and, once they are satisfied with it, can include this code in an IS.

Although the gesture representations described above permit developers to include gestures into information systems, none of them addresses model-driven gesture representation. In this

work we propose a model-driven approach for representing gestures with a high-level of abstraction, thus offering platform-independence and reusability. By providing the proper transformations it is possible to target several gesture recognition technologies. In this study we focus on user-defined, multi-stroke, semaphoric gestures [13].

3.2 The Role of Gesture-based Interfaces in IS Engineering

Gesture-based interfaces can play two major roles in IS engineering, depending on whether we intend to incorporate this natural interaction into (i) CASE tools or (ii) into the IS themselves. In the former case, the interest is to increase the IS developer's efficiency, whereas in the latter the aim is to improve IS usability, especially in operations in the field, where the lack of a comfortable office space hinders the ergonomics of mouse and keyboard. In both cases, gesture-based user interfaces development methods and tools are needed. Some examples of methods and tools are described in [33] and [34] in which the authors propose a method of introducing gesture-based interaction into an interface.

Some studies have reported on the definition of methods to generate a user interface: UsiGesture [35] allows a designer to integrate gesture-based interaction into an interface, but it lacks the techniques to model, analyse or recognise gestures. The authors applied the method to developing a restaurant management tool. In [33] the authors propose a method that includes requirements definition, design, implementation and evaluation and apply it to creating a puzzle game. In [34], the authors describe a method with two variants, technology-based and human-based, and provide guidelines for the definition and selection of gestures based on ergonomic principles. GestureBar [36] 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 components. The aim of Open Gesture [37] is to facilitate inclusive interface designs that are usable by the elderly and the disabled as applied to an interactive television project.

In this study we propose a similar flow to that offered by Guimaraes in [33], but automating the implementation of gesture-based interfaces by means of model transformations. In future work we plan to provide support to the ergonomic principles proposed in [34].

4 The gestUI Method

4.1 Introduction

The study applied an existing user interface development method to include gesture-based interaction in WIMP[3] user interfaces by means of gestUI. The existing method can be code-centric or model-driven. In this section, we describe the process to include gestUI in a code-centric method. Appendix A explains the process to include gestUI in a model-driven method to user interface development. In Figure 2, activities and products are shown in grey, and gestUI activities and products are shown in white. The flow of the process of the existing method is shown with blue arrows and in the case of gestUI is shown with black arrows.

[3] WIMP: Window, Icon, Menu, Pointer

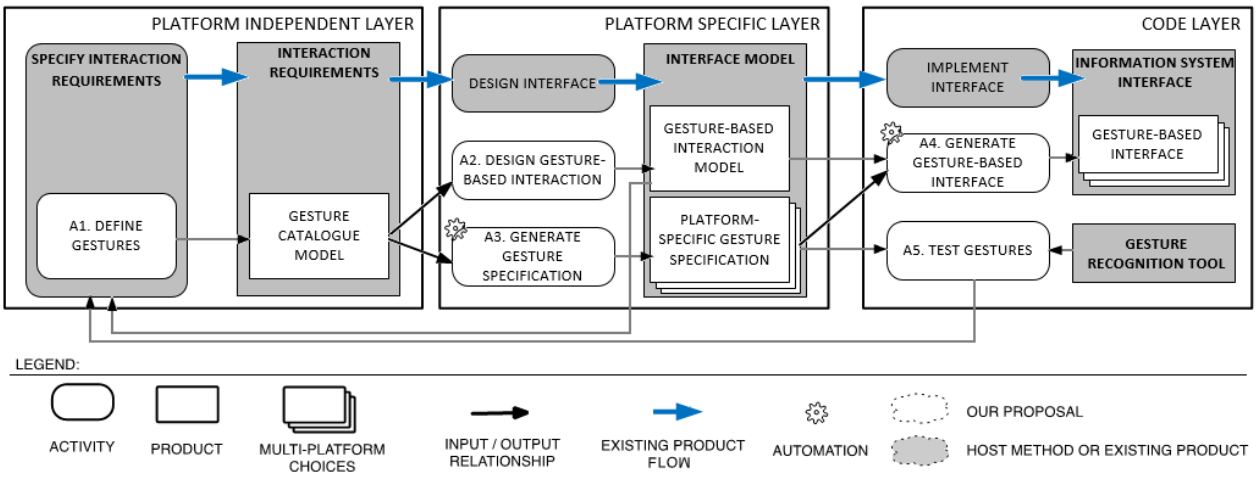

Figure 2. gestUI method overview

The existing method begins with the interaction requirements specification, then continues with the user interfaces design which are implemented in a programming language to obtain the information system user interfaces. The model-driven gestUI method is inserted into the existing method containing activities and products to help in defining the custom gesture catalogue and to include gesture-based interaction in a user interface.

4.2 gestUI

gestUI is a user-driven iterative method that follows the MDD paradigm. It is user-driven because the users participate in all non-automated activities and iterative because it aims to discover the necessary gestures incrementally and provides several loopbacks.

The main artefacts in gestUI are models which conform to MDA, a generic framework of modelling layers that ranges from abstract specifications to the software code. Figure 2 shows a view of the method from an MDA perspective. The PIM that drives the process is the gesture catalogue metamodel. Using a M2M transformation with ATL we obtain the platform-specific gesture specification (model). This PSM is converted into gesture-based user interface source code using M2T transformation rules defined by Acceleo. Moreover, PSM is converted into the gesture catalogue by the gesture recognition tool (i.e. quill, $N, iGesture) using transformation rules defined by Acceleo.

According to Figure 2, gestUI employs M2T transformations to obtain the information required to define (i) custom gestures and (ii) to include gesture-based interaction in an information system user interface. The gesture catalogue obtained in each execution of the M2T transformation can be stored in a repository for reuse in other similar processes.

The activities and products shown in Figure 2 are as follows:

The **computation-independent layer** is omitted because gestUI already assumes that the IS is going to be computerised.

In the **platform-independent layer** is included the Activity A1, “Define gestures”, in which the developer specifies the gestures in collaboration with representative IS 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 depicted in Figure 3. 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 are 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, 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.

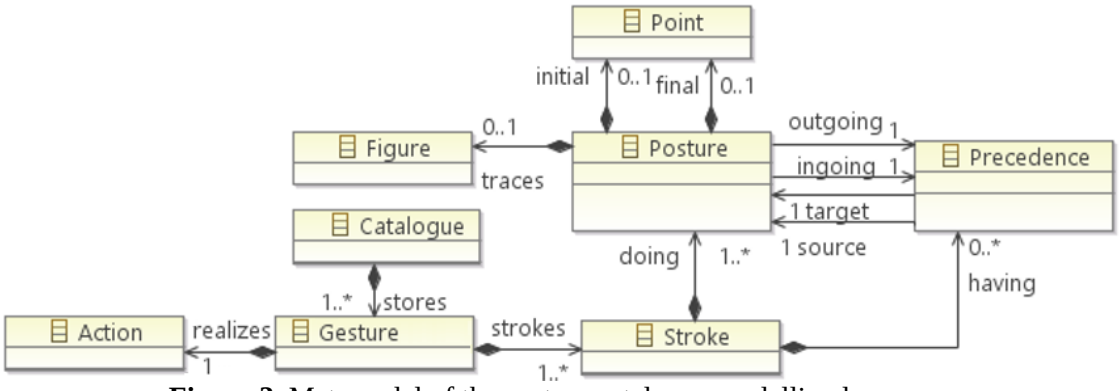

Figure 3. Metamodel of the gesture catalogue modelling language

In the **platform specific 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 or information systems. The description of each of these activities are as follows:

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 to the class "Gesture" 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 IS user interface development the actions are specified in the interface model (see Appendix A). In a code-centric interface development they are implemented on 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 user interface and enable it to be executed by each previously defined gesture. The product obtained in this activity is stored in the "gesture-based interaction model".

Activity A3, "*Generate gesture specification*", consists in an M2M transformation using ATL as model transformation language. Figure 4 shows the M2M transformation that is executed by means of a transformation definition (script.atl) which contains the transformation rules written in ATL. In Figure 4, Ma is a gesture catalogue model which conforms to gesture catalogue metamodel, MMa. Mb is the platform-specific gesture specification (model) which conforms to gesture specification model, MMb.

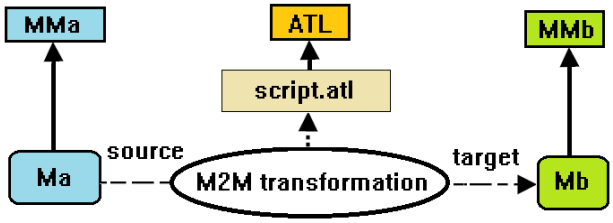

Figure 4. M2M transformation using ATL

An example of the transformation definition written in ATL is included here:

```
-- rule to create Gesture in multiStrokeGesture
rule Gesture {
        from
          s1: MMMTG!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)))
                )
}
```

This definition contains the rule to create the class "Gesture" in the target model. In this transformation definition, the input is the gesture catalogue model and the output is platform-specific gesture specification.

In the **code layer**, we have two activities.

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 IS interface '*Gesture based interface*' is automatically generated by the platform-specific layer artefacts.

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 [8], $N [6] and iGesture [21]). We apply M2T transformation to generate the platform-specific gesture catalogue for each gesture recognition tool. This transformation is executed via a script containing the transformation rules written in Acceleo, applying a script that specifies information such as the classes and components participating in the generation, output folders, etc. The combination of the components that support the code generation process is depicted in Figure 5. The template definition, which drives code generation, constitutes the most important part of the transformation process. Appropriate templates have been defined for the platforms considered in our work: XML ($N and iGesture), GDT (quill) and Java.

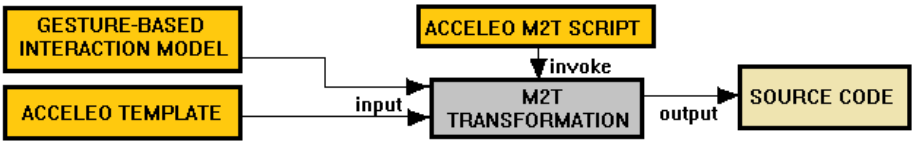

Figure 5. The code generation process

The next paragraph 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.

```
[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]
```

5 The gestUI Tool

In order to demonstrate the applicability of the proposed method we implemented tool support using Java and Eclipse Modelling Framework (Figure 6). In this figure, the acronym included in brackets in each subsystem (A1, A3) and component (A2, A4 and A5) corresponds to the number of the activity that each supports (see Section 4). The method's internal products are not shown but the relationship with the external gesture recogniser is represented. In Figure 6, the components showed with dark grey shapes belong to an existent IS. The light grey shapes belong to our proposal. The subsystems and components are described in this section.

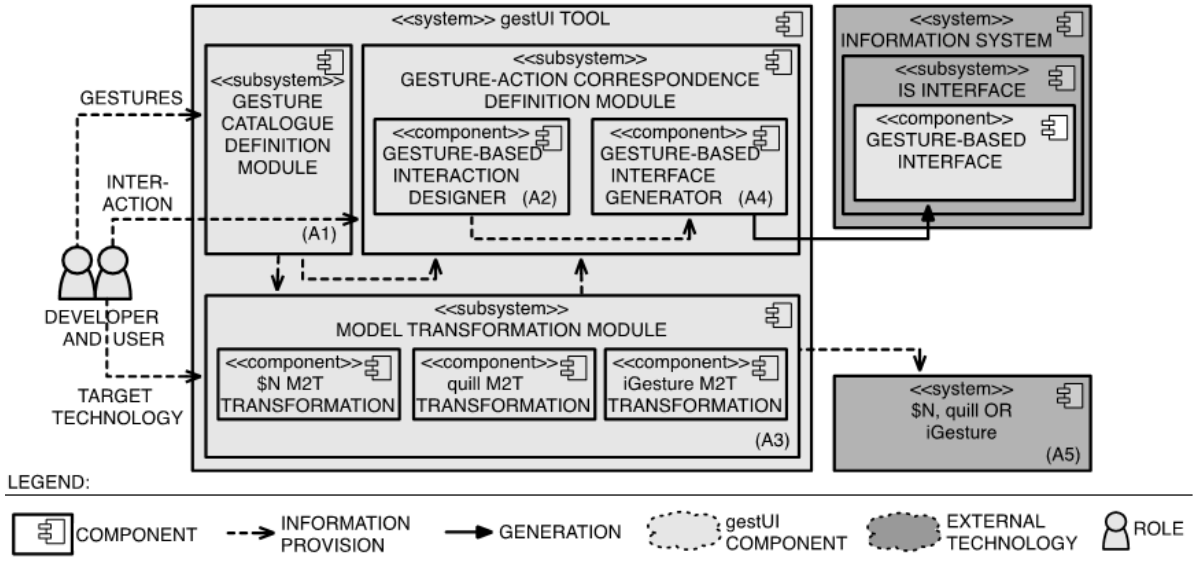

Figure 6. gestUI tool overview

The implemented tool support has three options (Figure 7): (i) "New Catalogue" to define gesture catalogue model, (ii) "Specific Catalogue" associated with platform-specific gesture specification and, (iii) "Gesture-Action" to define gesture-action correspondence and source code generation.

Figure 7. Main interface of the tool support

5.1 Gesture Catalogue Definition Module

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. Each gesture sketched by the user consists of one or more strokes, each stroke is defined by a set of points described by coordinates (X, Y) and a timestamp (t). In applying $N as the gesture recognizer when the gesture is sketched on a canvas (Figure 8, left), the following data is captured: number of strokes specified during the sketching of the gesture, number of points contained in each stroke and the value of each point (X, Y) together with the timestamp (t) of each point.

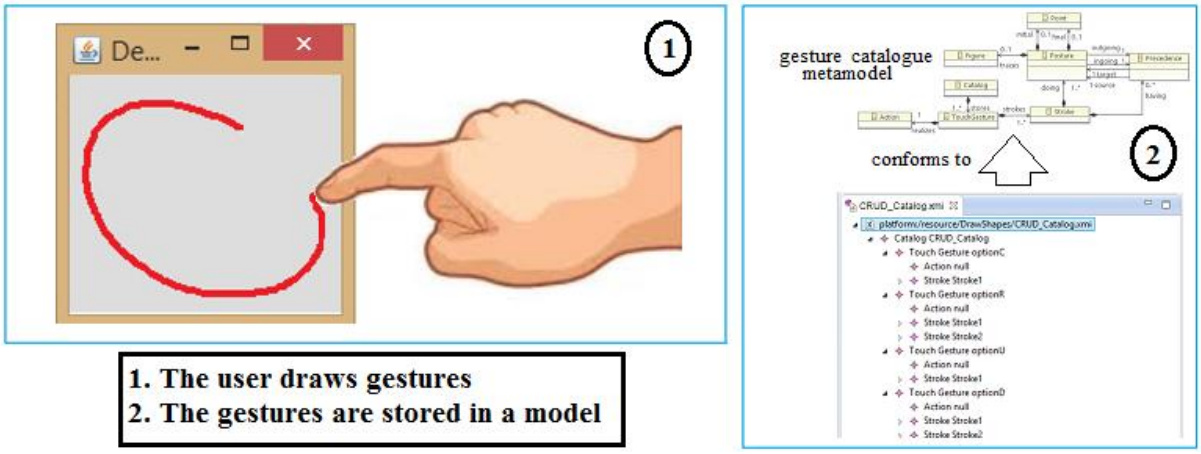

Figure 8. Platform-independent gesture catalogue definition

After capturing the data required by $N to analyse each gesture, the data is stored in the 'Gesture catalogue model' which conforms to the metamodel defined in this study (Figure 8, right).

5.2 Model Transformation Module

This module makes it possible to obtain the platform-specific gesture specification by means of an M2M transformation. The transformation rules are written in ATL. The user must specify the parameters in the interface showed in Figure 9. The parameters required to execute the M2M transformation are: gesture catalogue model (input) must conform to the gesture

catalogue metamodel (input) and platform-specific gesture specification model (output) must conform to gesture specification metamodel output).

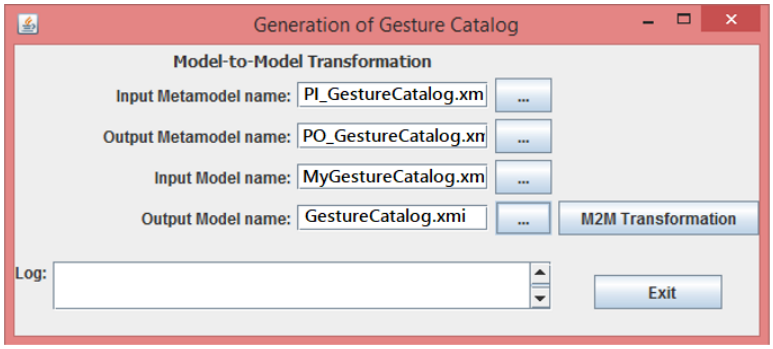

Figure 9. M2M transformation parameters

5.3 Gesture-action Correspondence Definition Module

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 10): the previously created 'Gesture catalogue model' and the user interface (e.g., a Java source code). The output of this module is the source code of the previously specified user interface, but now includes a source code to support the gesture-based interaction. In order to apply the parsing process in the user interface source code, we included methods in the implementing 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.

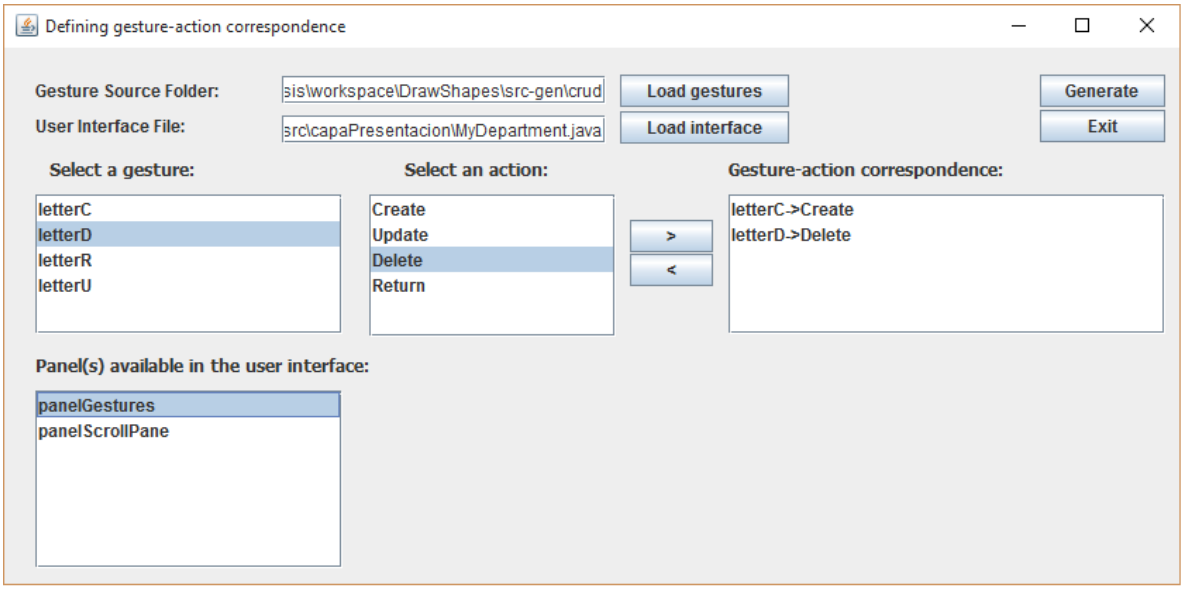

Figure 10. Interface for defining gesture-action correspondence and to generate source code

In the former type, SWT provides widgets (controls and composites) to be included in the user interface with the aim of assigning actions [38]. 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 11). The parsing process then searches for these actions in order to complete the gesture-action correspondence definition.

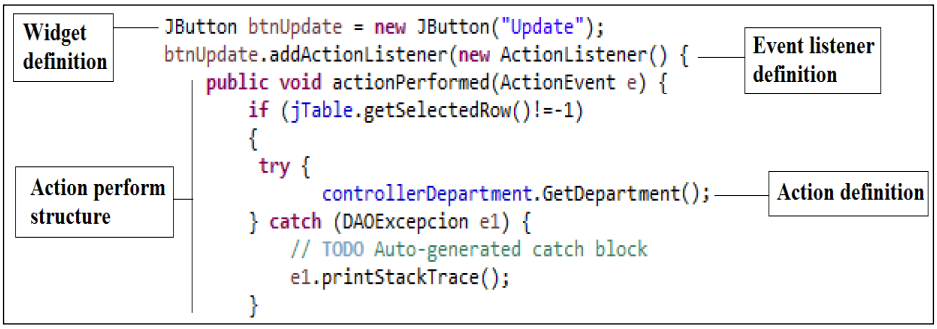

Figure 11. SWT components to define actions

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. [38]. 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 12).

The parsing process analyses the user interface source code searching for keywords corresponding to widgets available in the 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.

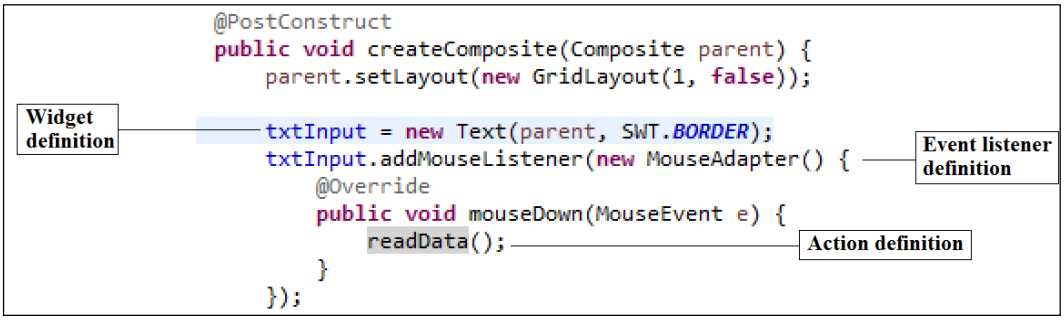

Figure 12. JFace and SWT components used to define an action in a user interface

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). In addition, the definition of the classes is changed to include some event listeners. Finally, the source code obtained from the completed process should be inserted in the complete source code of the user interface and, of course, be compiled again.

6 Demonstration of the Method and 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.

6.1 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 13).

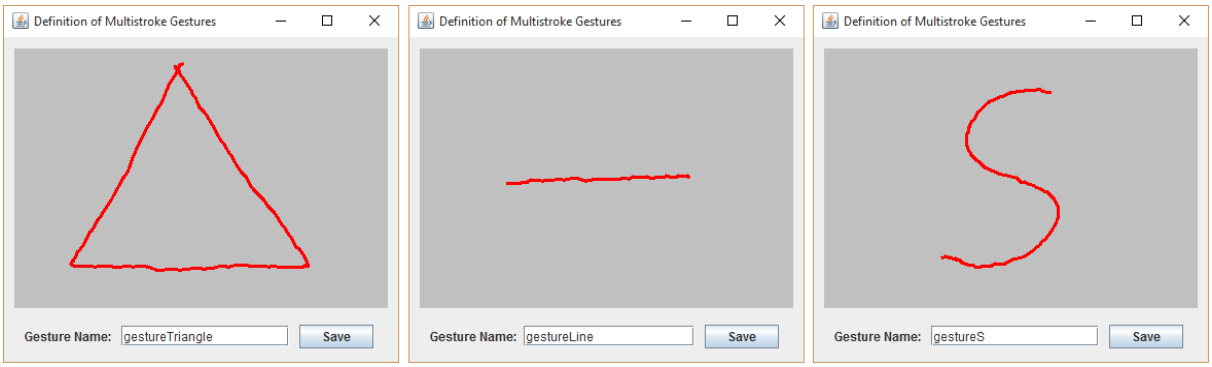

Figure 13. Gesture catalogue defined by gestUI

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 14).

To test the gestures we used the M2T transformation described in Section 5.2, considering successively $N, quill and iGesture as the target platform. Our aim was to obtain the gesture catalogue in the structure specified for each framework (Figure 14). In this case, we specified the transformation rules with Acceleo and then we performed the M2T transformation for each framework.

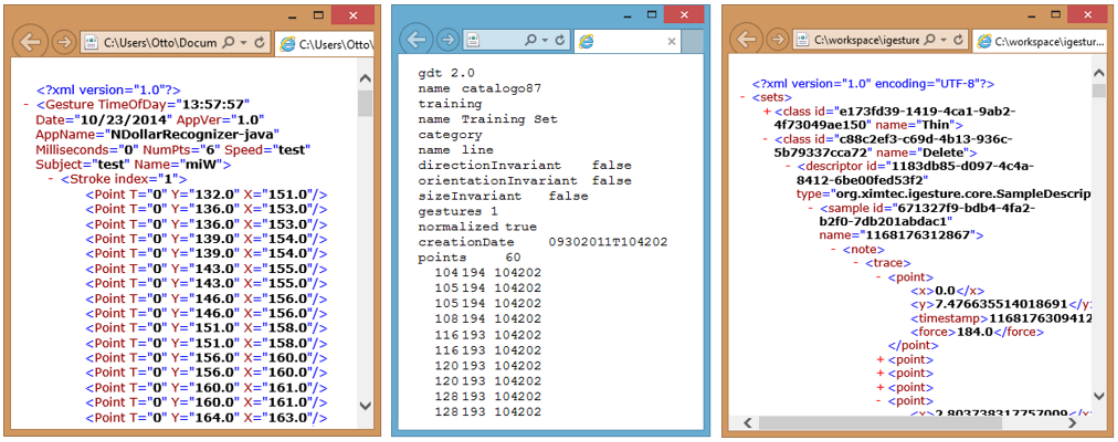

Figure 14. Gesture description files: $N (left), quill (centre), iGesture (right)

In the next step we use each framework to test the gestures. For instance, we included some screenshots of quill interface. The quill interface was used to import the gesture catalogue obtained in the model transformation that is shown on left side of Figure 15. On the right, the gesture catalogue already included in the framework can be seen.

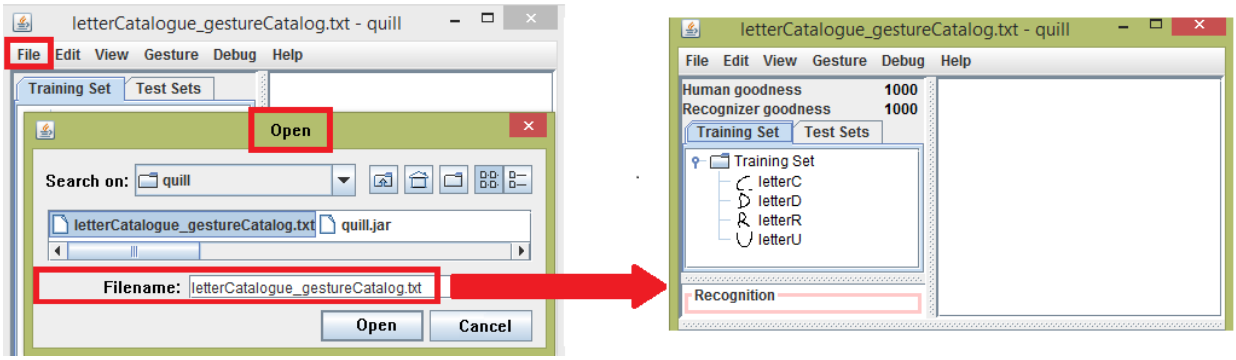

Figure 15. Importing the gesture catalogue to the quill framework

In the last step the user sketched the gestures contained in the gesture catalogue using the sketch area defined in the interface of each framework. All the frameworks included the algorithm (not described here) were used to recognize the gestures sketched by the users. Figure 16 shows how the gesture catalogues are effectively recognised when imported to SN, quill and iGesture frameworks.

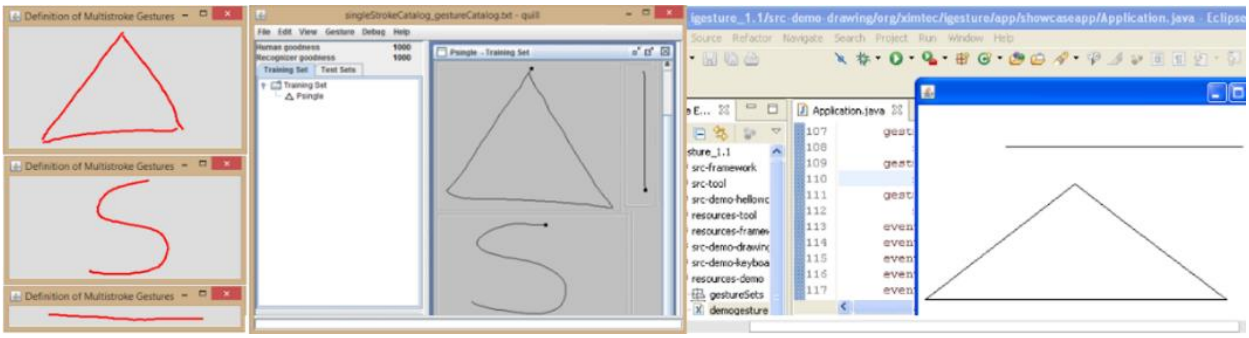

Figure 16. Examples of multi-stroke gestures: $N (left) and quill (centre) and iGesture (right)

6.2 Applying the Method and the Tool to integrate GestUI into User Interface Development

For illustration purposes, we used a form-based information system (IS). In this case a fictitious example of a university management system, and we narrate the project as if it had actually happened. Figure 17 shows the classroom management diagram of the fictitious university. In this section, we considered an IS with WIMP interfaces and for the sake of brevity, we only considered two interfaces for the demonstration: the main interface and the department management interface. The form-based information system was developed in Java in Microsoft Windows.

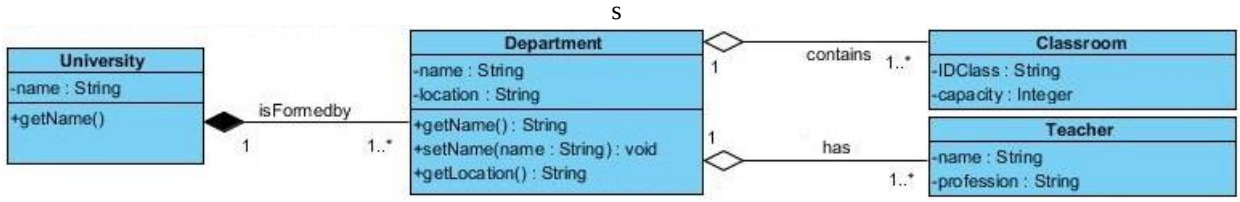

Figure 17. UML class diagram of the demonstration case

In the first iteration, the university tells the developers that it would like the gestures to resemble parts of the university logo. They therefore use the Gesture catalogue definition

module to create the first version of the 'Gesture catalogue model' containing these three gesture indicators: △ for departments, || for teachers and □ for classrooms. However, when the first user interface design is available (see Figure 18), they realise that other gestures are needed. After defining and testing new gestures, they decide that navigation will be by means of the above-mentioned gesture indicators, and that similar actions that appear on different screens will have the same gestures indicators (e.g., the gesture + will be used to create both new departments and teachers).

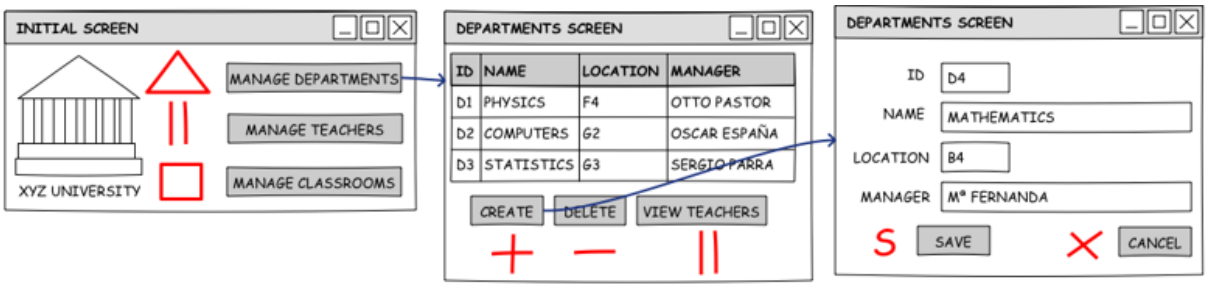

Figure 18. Screen mockups (gestures are shown in red, next to action buttons)

The developers would assign the gesture-action correspondence in collaboration with the user, supported by the Gesture-action correspondence definition module. The correspondences are informally shown in Figure 18, next to each action button and are described in Table 1.

Table 1. 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.

The user can employ the model transformation option to apply an M2T transformation and to obtain a platform-specific gesture catalogue. We found that if the Java source code of the user interface using the 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 selected was $N. Since the users felt more comfortable with multi-stroke gestures (especially when tracing certain letters and symbols), quill was discarded. The final IS interface consists of several screens for managing university information. In this scenario, the users can still interact with the IS 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 19 represents three interfaces from the IS: the task starts with the main interface (Figure 19, left) where the users can select one of the options on the menu. For the sake of simplicity, the menu is shown as an array of buttons. According to the requirements indicated above, if a user sketches the gesture "△" in the main interface of the IS then he/she obtains a

second user interface containing the information on the existing departments (Figure 19, centre). 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 (Figure 19, right). When the user finishes entering the information, sketching "S" on this third interface saves the information to a database.

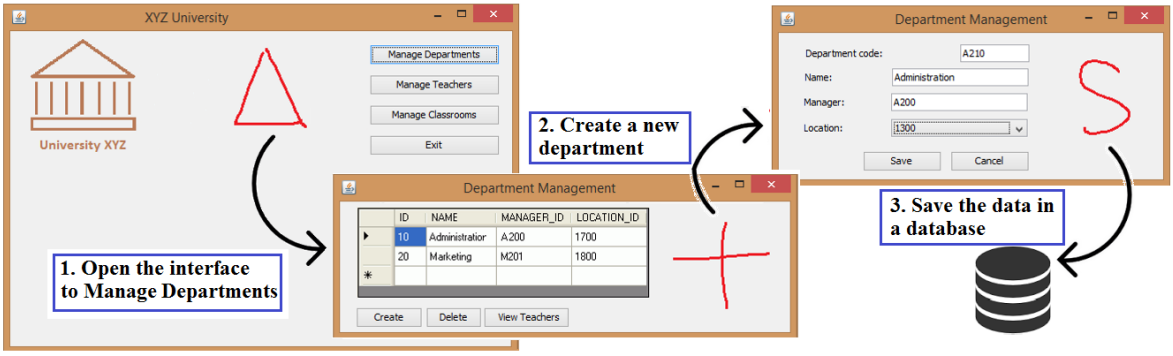

Figure 19. Using gestures to execute actions on the interfaces

7 Summary and Future Work

This paper describes gestUI, a model-driven method together with its tool support to specify multi-stroke gestures and automatically generate the information system components that support gesture-based interaction.

We assessed the method and tool support by applying them to a gesture testing case, generating the platform-specific gesture specifications for three existing gesture-recognition technologies 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. Its main current limitations are related to the target interface technologies (currently, only Java is used) and the fact that multi-finger gestures are not supported. These limitations will be addressed in future work.

We also plan further validation by applying the approach to the development of an actual IS and to extending a CASE tool with gesture-based interaction (the Capability Development Tool being developed in the FP7 CaaS project). We also plan to integrate gestUI into a fully-fledged model-driven framework capable of automatically generating the presentation layer and extend its application with gesture-based interaction modelling and code generation.

Acknowledgements

This work was supported by SENESCYT and Universidad de Cuenca from Ecuador, and received financial support from Generalitat Valenciana under Project IDEO (PROMETEOII/2014/039).

References

[1] D. Wigdor and D. Wixon, Brave NUI world: designing natural user interfaces for touch and gesture, USA: Morgan Kaufmann Publ., 2011.

[2] Fujitsu, "Touch- and gesture-based input to support field work," Fujitsu Laboratories Ltd., 18 02 2014. [Online]. Available: http://phys.org/news/2014-02-touch-gesture-based-field.html. [Accessed 24 11 2014].

[3] 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.

[4] 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.

[5] O. Parra, S. España and O. Pastor, "A Model-driven Method and a Tool for Developing Gesture-based Information Systems Interface," in *Proceedings of the CAiSE'15 Forum at the 27th International Conference on Advanced Information Systems Engineering*, Stockholm, Sweden, 2015.

[6] L. Anthony and J. O. Wobbrock, "A Lightweight Multistroke Recognizer for User Interface Prototypes," *Proc. of Graphics Interface,* pp. 245-252, 2010.

[7] D. Rubine, "Specifying gestures by example," *ACM SIGGRAPH Computer Graphics and Interactive Techniques,* vol. 25, no. 4, pp. 329-337, 1991.

[8] A. C. Long and J. Landay, Quill: a gesture design tool for pen-based user interfaces, Berkeley: University of California, 2001.

[9] S. Swigart, "Easily Write Custom Gesture Recognizers for Your Tablet PC Applications," Microsoft Corp., USA, 2005.

[10] R. Wieringa, "Design Science as Nested Problem Solving," in *DESRIST'09*, Malvern, PA, USA, 2009.

[11] F. Paterno, C. Santoro and L. D. Spano, "MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments," *ACM Transactions on Computer-Human Interaction,* vol. 16, no. 4, pp. 1-30, 2009.

[12] W. W. Group, "MBUI - Abstract User Interface Models," World Wide Web Consortium (W3C), 08 04 2014. [Online]. Available: http://www.w3.org/TR/abstract-ui/. [Accessed 16 11 2015].

[13] M. Karam and M. C. Schraefel, "A taxonomy of Gestures in Human-Computer Interaction," in *Retrieved from http://eprints.soton.ac.uk/261149/*, 2005.

[14] S. Zhai, P. O. Kristensson, C. Appert, T. H. Anderson and 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.

[15] F. Truyen, "The Fast Guide to Model-Driven Architecture. The Basics of Model-Driven Architecture," Cephas Consulting Corp., 2006.

[16] J. M. Siegel, "Model Driven Architecture (MDA) Guide rev. 2.0," OMG - Object Management Group, 2014.

[17] A. Kleppe, J. Warmer and W. Bast, MDA Explained: The Model Driven Architecture : Practice and Promise, USA: Addison-Wesley Prof., 2003.

[18] M. Brambilla, J. Cabot and M. Wimmer, Model-driven Software Engineering in Practice, USA: Morgan & Claypool, 2012.

[19] M. Gribaudo, Theory and Application of Multi-Formalism Modeling, Italy: IGI Global, 2013.

[20] F. Jouault, F. Allilaire, J. Bézivin and I. Kurtev, "ATL: A model transformation tool," *Science of Computer Programming,* vol. 72, pp. 31-39, 2008.

[21] B. Signer, U. Kurmann and M. Norrie, "iGesture: A General Gesture Recognition Framework," in *9th Conf. on Document Analysis and Recognition*, Brazil, 2007.

[22] 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.

[23] L. Spano, A. Cisternino and F. Paternò, "A Compositional Model for Gesture Definition," *LNCS Human-Centered Soft. Eng.*, vol. 7623, pp. 34-52, 2012.

[24] A. Lascarides, "Formal semantics for iconic gesture," in *Proceedings of the 10th Workshop on the Semantics and Pragmatics of Dialogue (Brandial)*, 2006.

[25] K. Kin, B. Hartmann, T. DeRose and M. Agrawala, "Proton: Multitouch Gestures as Regular Expressions,"

in *CHI'12*, Austin, Texas, USA, 2012.

[26] K. Kin, B. Hartmann, T. DeRose and M. Agrawala, "Proton++: A Customizable Declarative Multitouch Framework," in *UIST'12*, Cambridge, USA, 2012.

[27] L. D. Spano, A. Cisternino, F. Paternò and G. Fenu, "GestIT: A Declarative and Compositional Framework for Multiplatform Gesture Definition," in *EICS'13*, London, United Kingdom, 2013.

[28] Ideum, "GestureML," Ideum, 22 November 2014. [Online]. Available: http://www.gestureml.org/. [Accessed 6 December 2014].

[29] T. Hachaj and M. Ogiela, "Recognition of Human Body Poses and Gesture Sequences with Gesture Description Language," *JOURNAL OF MEDICAL INFORMATICS & TECHNOLOGIES,* vol. 20, pp. 129-136, 2012.

[30] H. Lü and Y. Li, "Gesture Coder: A tool for programming multi-touch gestures by demonstration," *Proceedings of the 2012 ACM on Human Factors in Computing Systems,* pp. 2875-2884, 2012.

[31] J. Wobbrock, A. Wilson and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," *Proceedings of the ACM Symposium on User Interface Software and Technology (UIST '07),* pp. pp. 159-168, 2007.

[32] R. D. Vatavu, L. Anthony and J. Wobbrock, "Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes," *Proc. of the 14th ACM international conference on Multimodal interaction,* pp. 273-280, 2012.

[33] 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.

[34] M. Nielsen, M. Storring, T. Moeslund and E. Granum, "A Procedure for Developing Intuitive and Ergonomic Gesture Interfaces for Man-Machine Interaction," Aalborg University, Aalborg, Denmark, 2003.

[35] F. Beuvens and J. Vanderdonckt, "Designing Graphical User Interfaces Integrating Gestures," in *SIGDOC'12*, Seattle, Washington, USA, 2012.

[36] A. Bragdon, R. Zeleznik, B. Williamson, T. Miller and J. LaViola, "GestureBar: improving the approachability of gesture-based interfaces," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems,* pp. 2269-2278, 2009.

[37] M. Bhuiyan and R. Picking, "A Gesture Controlled User Interface for Inclusive Design and Evaluative Study of Its Usability," *Journal of Software Engineering and Applications,* vol. 4, no. 9, pp. 513-521, 2011.

[38] D. Gallardo, E. Burnette and R. McGovern, Eclipse in Action. A guide for Java developers, Greenwich: Manning Publications Co., 2003.

[39] P. Bottoni, E. Guerra and J. de Lara, "Metamodel-based definition of interaction with visual environments," in *Proceedings of the Second International Workshop on Model Driven Development of Advanced User Interfaces - MDDAUI'06*, Genova, Italia, 2006.

Appendix A. Including gestUI in a Model-Driven User Interface Development Method

In order to include gestUI in a model-driven user interface development method, the platform-independent layer of gestUI (see Section 4) is different for the code-centric method. Activity A2, "Design gesture-based interaction" (see Section 4.2) is changed to consider an interaction model in order to include gesture-based interaction. Instead of including an attribute containing the filename of the user interface in the "Gesture" class, we link the gesture catalogue metamodel (shown with a red line in Figure A1.1) with an interaction metamodel (Figure A1.1).

In Figure A1.1, an excerpt of the interaction metamodel proposed by Bottoni et al. [39] is shown. In this figure, the classes and links presented in green are the minimum conditions required to include gestUI in a model-driven user interface development method. That is, with the aim of including gestUI in an existing model-driven user interface development method we

need that this method contains a class to manage events (or actions) in the interface and another class to manage the elements of the interface (canvas, buttons, etc.).

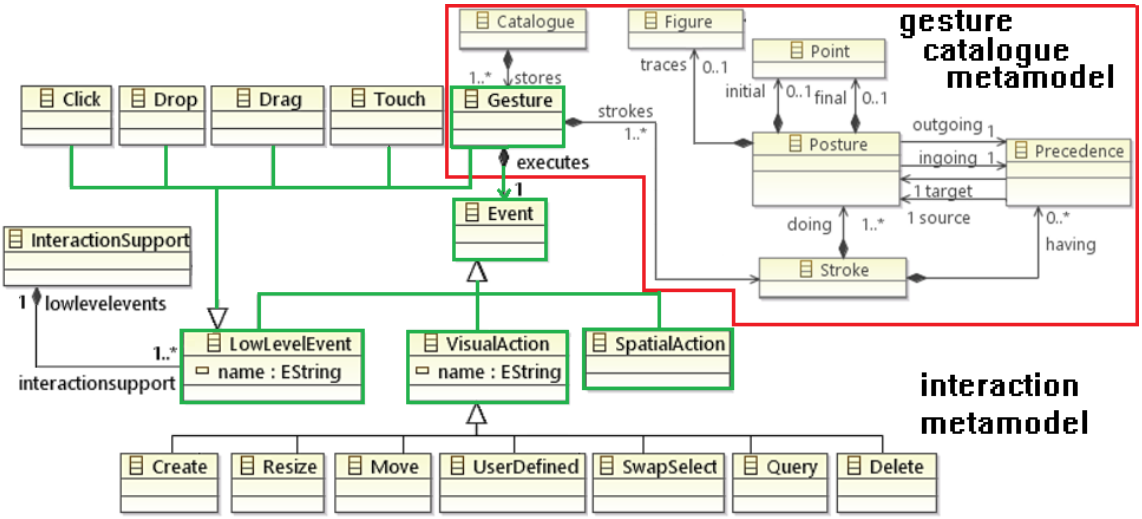

Figure A1.1. Minimum conditions to include gestUI in an interaction model

In order to show the process of including gestUI in an existing model-driven method, we consider two existing alternatives to define user interfaces: (i) MARIA (Modelbased lAnguage foR Interactive Applications) and (ii) the MBUID specification given by W3C. The description of the process in which gestUI is incorporated in these methods is briefly described in this Appendix.

In order to include gestUI in MARIA, we considered the AUI metamodel described in [11]. This metamodel includes an event model which enables specifying how the user interface responds to events triggered by users. Two types of events have been defined: "*Property Change Events*" and "*Activation Events*". We are interested in the latter because with this type of event it is possible to specify actions/commands for the execution by users. Figure A1.2 shows an excerpt of the MARIA AUI metamodel with the gesture catalogue metamodel linked by means of "event" and "Gesture" classes.

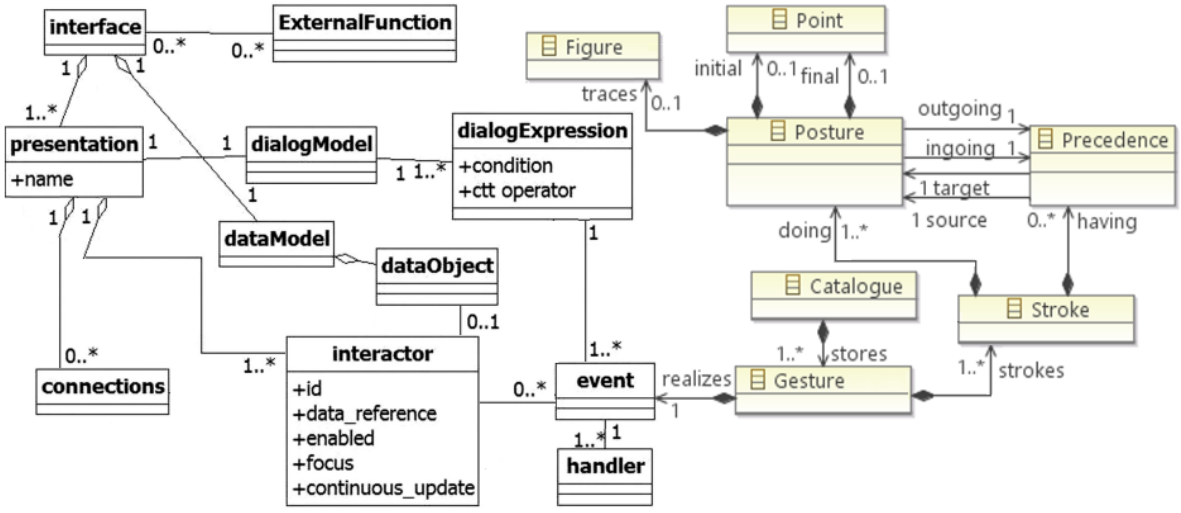

Figure A1.2. Including gesture catalogue metamodel in MARIA AUI metamodel

The inclusion of gesture catalogue metamodel in the AUI defined in the MBUID specification is shown in Figure A1.3. MBUID facilitates the interchange of designs through a layered approach that separates out different levels of abstraction in the user interface design. The AUI metamodel contains the "*InteractionEvent*"class, which defines an interaction event. This inclusion contains a generalization definition with TriggerEvent, SelectionEvent, DeselectionEvent, and InputEvent as classes that permit the specification of event types that can be executed by gestures. Considering gestUI, the class "Action" corresponds to the "InteractionEvent" class in order to define the action to be executed with a gesture sketched by the user.

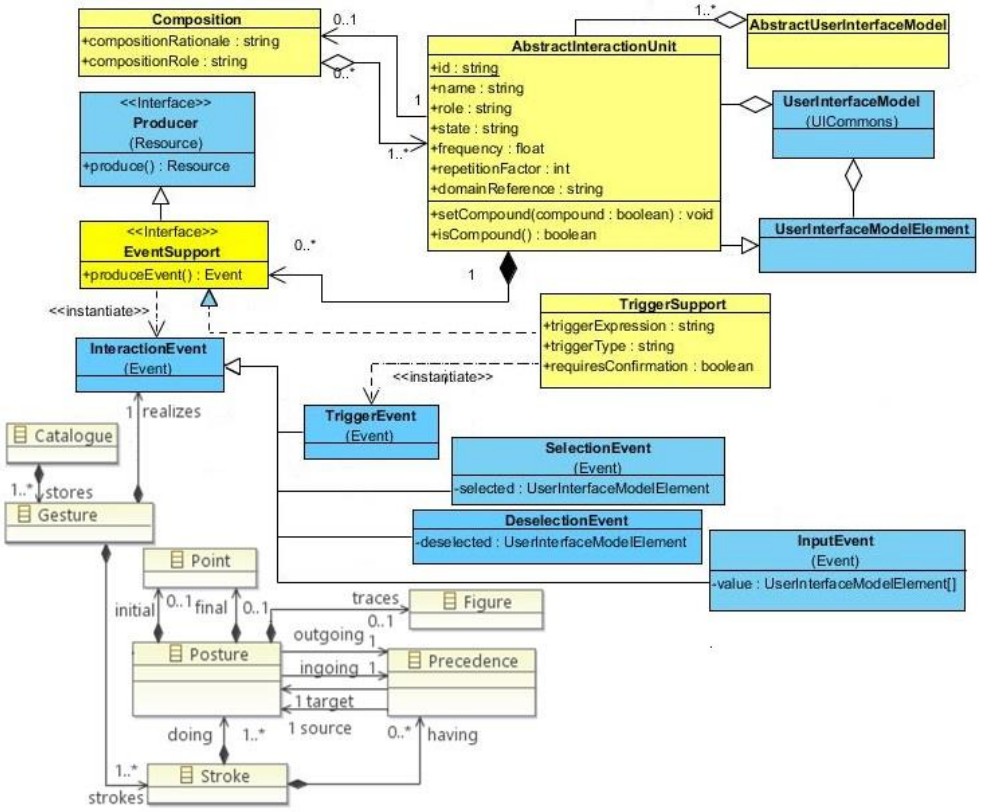

Figure A1.3. Including gesture catalogue metamodel in MBUID AUI metamodel