A Model-driven Method and a Tool for Developing Gesture-based Information System Interfaces

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

[1] PROS Research Centre, Universitat Politècnica de València, Spain
[2] Computer Science Department, Universidad de Cuenca, Ecuador

otpargon@upv.es,{sergio.espana, opastor}@pros.upv.es

Abstract. Considering the technological advances in touch-based devices, gesture-based interaction has become a prevalent feature in many application domains. Information systems are starting to explore this type of interaction. Currently, gesture specifications are hard-coded by developers at the source code level, hindering its reusability and portability. Similarly, defining new gestures in line with users' requirements is further complicated. This paper describes a model-driven approach to include gesture-based interaction in desktop information systems and a tool prototype to: capture user-sketched multi-stroke gestures and transform them into a model, automatically generating the gesture catalogue for gesture-based interaction technologies and gesture-based interface source code. We demonstrate our approach in several applications, ranging from case tools to form-based information systems.

Keywords: Model-driven architecture, gesture-based interaction, multi-stroke gestures.

1 Introduction

New devices come together with new types on interfaces (e.g. based on gaze, gesture, voice, haptic, brain-computer interfaces). Their aim is to increase the naturalness of interaction [1], although this is not exempt from risks. Due to the popularity of touch-based devices, gesture-based interaction is slowly gaining ground on mouse and keyboard in domains such as video games and mobile apps. Information systems (IS) are likely to follow the trend, especially, in supporting tasks performed outside the office [2].

Several issues may hinder the wide adoption of gesture-based interaction in complex information systems engineering. Gesture-based 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 a

great coding and maintenance effort when multiple platforms are targeted, has a negative impact on reusability and portability, and complicates the definition of new gestures. Some of these challenges can be tackled 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 gesture-based IS interface development, which is intended to allow software engineers focusing 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.

2 State of art

2.1 State of the art on gesture representation

The representation of gestures according to related literature can be classified into some categories: (a) **based on regular expressions** [5] [6]: a gesture is defined by means of regular expressions formed by elements such as ground terms, operators, symbols, etc.; (b) **based on a language specification** [7]: XML is used as reference to describe gestures; (c) **based on demonstration** [8]: developers to define gestures, test the generated code, refine it, and, once they are satisfied with them, include the code in the applications.

In this research work we propose a model-driven approach in order to represent gestures with a high-level of abstraction, enabling platform-independence and reusability. By providing the proper transformations, it is possible to target several gesture recognition technologies. We focus on user-defined, multi-stroke, semaphoric gestures [9].

2.2 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 developers' efficiency, whereas in the latter the aim is to increase IS usability, especially in operations in the field, where the lack of a comfortable office space reduces the ergonomics of mouse and keyboard. In both cases, gesture-based interface development methods and tools are

needed. Some examples of methods and tools are described in [10], [11], where the authors propose a method to integrate gesture-based interaction in an interface.

In this work, we propose a similar flow to that of [10], but automate the implementation of gesture-based interfaces by means of model transformations. In future work, we plan to provide support to the ergonomic principles by [11].

3 The gestUI method

gestUI is a user-driven and iterative method that follows the MDD paradigm. The main artefacts are models which are conform to the Model-Driven Architecture, a generic framework of modelling layers that ranges from abstract specifications to the software code (indicated at the top of Fig. 1).

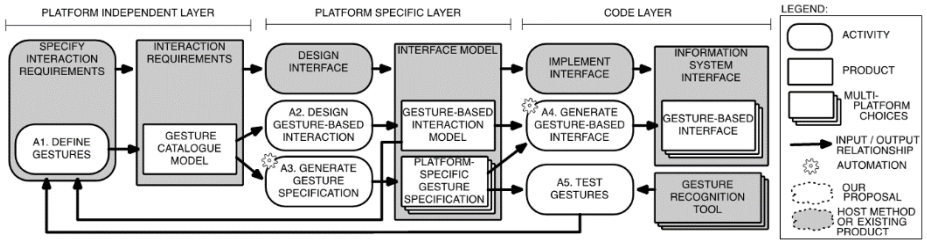

Fig. 1. gestUI method overview

The **computation-independent layer** is omitted because gestUI already assumes that the IS is going to be computerised. Note that gestUI is expected to be integrated into a full interface development method (represented with generic activities and artefacts in grey). Such a method can either be model-driven or code-centric. gestUI is user-driven because users participate in all non-automated activities; and it is iterative because it intends to discover the necessary gestures incrementally and provides several loopbacks. In the **platform-independent layer**, the gestures are defined (activity A1 in Fig. 1) by the developer but, preferably, in collaboration with representative users of the IS. Gestures are defined by sketching and are stored in the 'Gesture catalogue model', and is part of a larger 'Interaction requirements' specification. In the **platform specific layer**, a concrete gesture-recognition platform is selected (we currently support three platforms: quill [12], $N [8] and iGesture [13]). The 'Platform-specific gesture specification' (PSGS) is a machine-readable file format that can be interpreted by a gesture recognition tool. This specification can be automatically generated from the 'Gesture catalogue model'

(during A3). The interface is also designed in this layer, so now the gesture-based interaction can also be determined (A2) in collaboration with the user. This mainly consists of defining correspondences between gestures and interface actions. In the **code layer**, the developer and the user can test the gestures using the gesture recognition tool (A5). The 'Gesture based interface' is automatically generated from the platform-specific layer artefacts (A4). The tool generates components (e.g. Java code) that are embedded into the IS interface.

4 The gestUI tool

The gestUI tool is developed using Java and Eclipse Modelling Framework. As shown in Fig. 2, the tool is structured in three modules. The numbers in brackets indicate the method activity each component supports. The method's internal products are not shown. The relationship with the external gesture recogniser is represented.

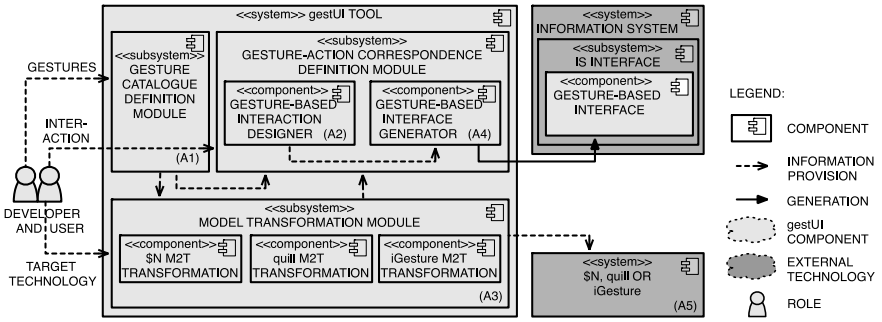

Fig. 2. gestUI tool overview

4.1 Gesture catalogue definition module

It supports the definition of new multi-stroke gestures by means of an interface implemented in Java in which the user sketches the gestures. The set of gestures sketched by the user constitutes the 'Gesture catalogue model', conforms to the metamodel defined in this work (*Fig. 3*).

4.2 Model transformation module

It requires as data: the 'Gesture catalogue model', the target technology specified by the developer, and the target folder to save the output. Depending on the target technology, a different M2T transformation is executed which creates the PSGS, in the corresponding file format (i.e. XML for $N and iGesture

and GDT 2.0 for quill). The transformation rules are written in Acceleo. The PSGS can be imported in a third-party gesture recogniser to test the gestures.

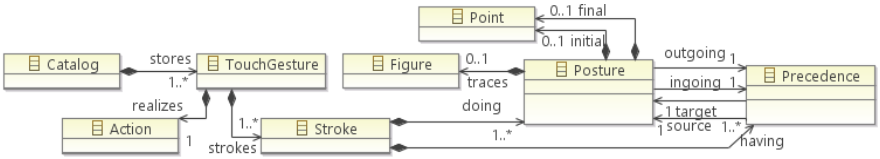

Fig. 3. Metamodel of the gesture catalogue modelling language

4.3 Gesture-action correspondence definition module

It allows the developer and the user to specify what action to execute whenever the gesture-based IS interface recognises a gesture. In a model-based IS interface development, the actions are specified in the interface model. In a code-centric interface development, they are implemented in the interface itself. We currently provide automated support to code-centric developments made in Java; that is, the gestUI module parses the source code of the user interface to obtain a list of actions. This module therefore requires two inputs: the previously created 'Gesture catalogue model' and the user interface (e.g. a Java code). The output of this module is the 'Gesture-based interaction model' and the same source code but now supporting the gesture-based interaction.

When generating the user interface Java source code, many references are included (e.g., to libraries to manage gestures, to libraries of the gesture-recognition technology (e.g. $N)), and some methods are added (e.g., to execute the gesture-action correspondence, and to capture gestures). Additionally, the class definition is changed to include some listeners, then the source code should obviously be compiled.

5 Demonstration of the method and tool

We demonstrate the integration of gestUI within a code-centric interface development method. For illustration purposes, we use a fictional, simple university management case and we narrate the project as if it actually happened. Fig. 4 shows the domain class diagram of a university with several departments, to which teachers are assigned and which manage classrooms. For the sake of brevity, we will just consider the two screens; namely, the initial and department management screens.

In the first method iteration, the university representatives tell the developer that they would like the gestures to resemble parts of the university logo.

Thus, they use the Gesture catalogue definition module to create a first version of the 'Gesture catalogue model' containing these three gestures: △ for departments, || for teachers and □ for classrooms. However, when the first interface design is available (see sketch in Fig. 5), they soon realise that other gestures are needed. This way, by defining new gestures, and after testing them, they determine that navigation will be done by means of the above-mentioned gestures, but that similar actions that appear across different screens shall have the same gestures (e.g. the gesture + shall be used to create both new departments and teachers).

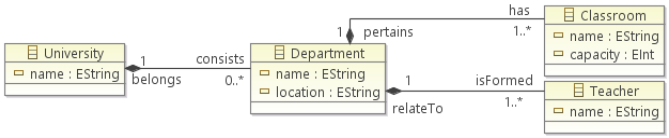

Fig. 4. UML class diagram of the demonstration case

The Model transformation module allows generating the PSGS for any of the available gesture-based recognition technologies (i.e. $N, quill and iGesture). The developer only needs to choose a single technology but we chose to demonstrate the multiplatform features of the gestUI method by generating the three gesture files. Using the appropriate tool, the users can test the gestures. Fig. 6 shows the gestures being recognised by the SN, quill and iGesture tools so the gestures have been properly converted by the Model transformation module.

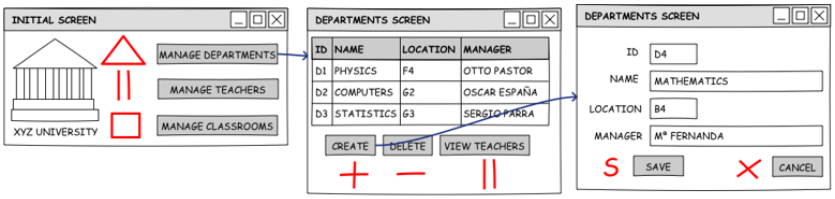

Fig. 5. Screen mockups (gestures are shown in red, next to action buttons)

The developer assigns the gesture-action correspondence in collaboration with the user, supported by the Gesture-action correspondence definition module. The correspondences are informally shown in Fig. 5, next to each action button. Once the Java source code of the traditional interface is available, then the components that support the gesture-based interaction are generated. In this case, the chosen underlying gesture-recognition technology is $N; the users felt more comfortable with multi-stroke gestures (especially with regards to tracing some letters and symbols) so quill was discarded. The final IS interface consists of several screens that allow managing university information. Users can still interact with the IS in the traditional way (i.e. using the

mouse), but now, they can also draw the gestures with a finger on the touch-based screen in order to execute the actions. Fig. 7 represents a specific interaction with the IS interface in which a department is being created.

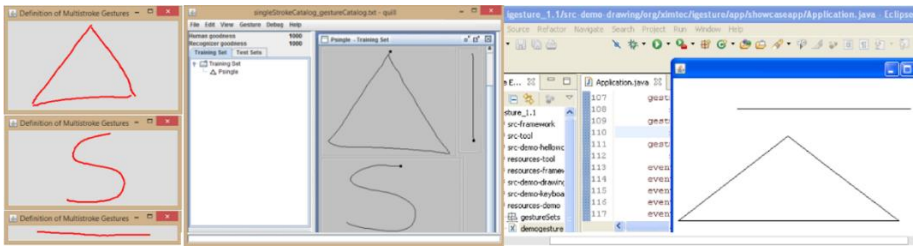

***Fig. 6**. An excerpt of multi-stroke gestures: $N (left) and quill (center) and iGesture (right)*

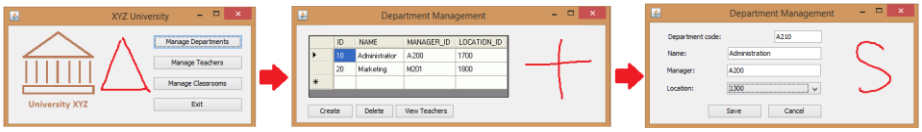

Fig. 7. Using gestures to execute actions on the interfaces

6 Summary and Future Work

We describe gestUI, a model-driven method, and the tool that supports it to specify multi-stroke gestures and automatically generating the information system components that support the gesture-based interaction. We validated the method and tool by applying them to a case and generated the Platform-specific gesture specification for three gesture-recognition technologies, to illustrate the multiplatform capability of the tool. The gestures were successfully recognised by the corresponding tools. We then automatically generated the final gesture-based interface components and integrated them into the IS interface. The advantages of the proposal 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) 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 a real 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 full-fledged model-driven framework capable of automatically generating the presentation layer, in order to extend it with gesture-based interaction modelling and code generation.

Acknowledgements

The author is grateful to his supervisors Sergio España and Óscar Pastor for their invaluable support and advice. This work has been supported by SENESCYT and Univ. de Cuenca - 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 Publishers Inc., 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] L. Spano, A. Cisternino and F. Paternò, "A Compositional Model for Gesture Definition," *LNCS Human-Centered Soft. Eng.*, vol. 7623, pp. 34-52, 2012.

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

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

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

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

[10] 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.

[11] 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.

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

[13] B. Signer, M. Norrie and U. Kurmann, "iGesture: A General Gesture Recognition Framework," in *Proceedings of ICDAR 2007, 9th Int. Conference on Document Analysis and Recognition*, Brazil, 2007.