Learning Gestures for Interacting with Low-Fidelity Prototypes

Tulio de Souza Alcantara
Department of Computer Science
Calgary, Canada
University of Calgary
tuliosouza@gmail.com

Jörg Denzinger
Department of Computer Science
Calgary, Canada
University of Calgary
denzinge@cpsc.ucalgary.ca

Jennifer Ferreira
Department of Computer Science
Calgary, Canada
University of Calgary
jen.ferreira@ucalgary.ca

Frank Maurer
Department of Computer Science
Calgary, Canada
University of Calgary
frank.maurer@ucalgary.ca

***Abstract*—This paper presents an approach to help designers create their own application-specific gestures and evaluate them in user-studies based on low fidelity prototypes of the application they are designing. In order to learn custom gestures, we developed a machine learning tool that uses an anti-unification algorithm to learn based on samples of the gesture provided by the designer.**

Keywords-custom gestures; anti-unification; low-fidelity prototyping

I. Introduction

Designing WIMP-based applications is a well-known challenge. This challenge becomes even bigger with the increasing popularity of touch-based devices and gesture based applications. The popularity of these devices requires designers to learn how gesture-based and touch-based interactions can improve the user experience. The constant evolution of these touch-based devices allows users to interact in new and different ways, which gives possibilities to user experience designers to try different approaches for users to interact with the software.

For touch-based devices a preferable user interface integrates gesture-based interactions into the applications [1]. Frameworks like *Windows Presentation Foundation* (WPF) [21] provide a set of pre-defined gestures that application developers can use easily [20]. However, the literature shows many examples of gestures that are not available out of the box, e.g. [1],[6].

When creating new gestures for interacting with touch-based applications, user experience designers have to determine if users consider them natural, understandable and easy to use [1].

In an effort to help interaction designers to create more intuitive and user oriented gestures, the present paper proposes Intelligent Gesture Toolkit (IGT) which will allow users with no previous programming knowledge

- to create their own custom gestures by providing samples and
- to evaluate these gestures in the context of a low-fidelity prototype of the application.

As defined by Mitra and Acharya in 2007, gestures can be static or dynamic [10]. In static gestures, a user assumes a certain pose using hands and fingers, while a dynamic gesture can be a stroke or set of strokes on the touch surface. Some gestures combine both dynamic and static elements. IGT works with both types of gestures.

A designer creates custom gestures in IGT by training the tool through examples of the gesture (samples) he wants to create. Through an anti-unification process IGT learns the most compact way to represent the gesture. The tool is integrated with a low-fidelity prototyping tool *Active Story Touch* (AST) [3] so that a designer can draw UIs and associate custom gestures that will trigger specific page transitions in the prototype.

For brevity of the paper, the user experience designer or user interaction designer will be referenced as designer and the user that will evaluate the low-fidelity prototypes will be referenced as user.

The next section discusses related work. This is followed by a tool description that will explain the architecture of IGT. The fourth section focuses on the learning aspect of the tool, explaining the anti-unification process in detail.

The fifth section explains the role of IGT in the low-fidelity prototype evaluation process. The last section presents the conclusion and future work of this research.

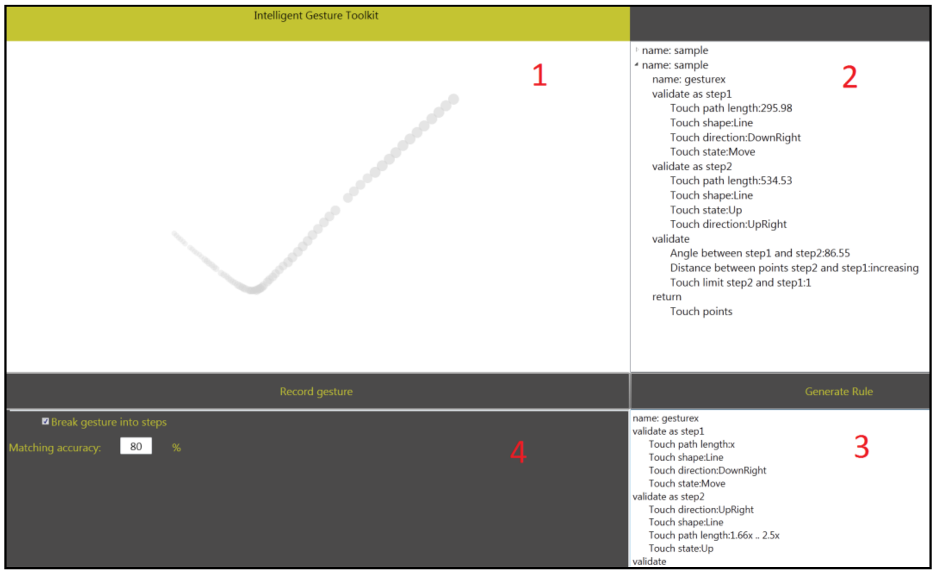
II. Related Work

In the current literature there are several solutions for gesture creation and recognition. In special, gesture recognition that uses statistical model approaches such as *Hidden Markov Models* (HMM) [11], [12], [7] and machine learning methods [13] has received a lot of attention.

In 2008, Damaraju and Kerne proposed a tool for gesture definition and recognition using trained HMM [7]. In comparison with IGT, this tool defines and recognizes finger gestures but lacks support for hand gestures.

Some studies focus on the gesture recognition and gesture definition as a specific problem. In 2007, Mitra *et al.* [10] published a survey that covers several approaches used in gesture recognition and in 1997 Bobick *et al.* [18] proposed a state-based approach to the representation and recognition of gestures.

In 2005, Karam proposed a taxonomy of gestures in the human computer interaction field [15]. A design model for gesture interactions was proposed by Lao and Wang in 2009 [16]. Finally, in 2010, Morris *et al.* compared a set of gestures created by end-users and HCI researchers to understand user preferences for gestures [17].



```
name: sample
name: sample
    name: gesturex
    validate as step1
        Touch path length:295.98
        Touch shape:Line
        Touch direction:DownRight
        Touch state:Move
    validate as step2
        Touch path length:534.53
        Touch shape:Line
        Touch state:Up
        Touch direction:UpRight
    validate
        Angle between step1 and step2:86.55
        Distance between points step2 and step1:increasing
        Touch limit step2 and step1:1
    return
        Touch points
```

```
name: gesturex
validate as step1
    Touch path length:x
    Touch shape:Line
    Touch direction:DownRight
    Touch state:Move
validate as step2
    Touch direction:UpRight
    Touch shape:Line
    Touch path length:1.66x .. 2.5x
    Touch state:Up
validate
```

Figure 1 Gesture definition using IGT

Wobbrock *et al*. published in 2009 a study to gather the gestures which people would use for specific tasks [1]. For the same tasks, they compared gestures created by Human Computer Interaction experts and users. The result showed the gestures created by the experts did not cover half of the ones created by users, meaning that designer opinion or expertise alone is not sufficient and user opinions are also required. This enforces the need for a user-centered design process when creating gestures.

To show the importance of user experience designers taking into consideration various aspects when defining gestures, Hinrichs and Carpendale found in a study using interactive tabletops, that the choice and use of multi-touch gestures are influenced by the action and social context in which these gestures occur [2]. Their study suggests that previous gestures influence the formation of subsequent gestures and that evaluating gestures requires contextualizing them in the application in which they will be used. Our use of the low-fidelity evaluations is in line with this approach.

In 2010, Khandkar and Maurer proposed a gesture definition language to be used by developers to define new gestures [6], which is the language used by the present paper to define the gestures.

Also in 2010, Khandkar et al. proposed a tool to support testing complex multi-touch gestures [9]. This enables automated testing for gesture based applications. This solution differs from the one proposed in this paper as it focuses more on development testing, while the solution proposed in this paper focuses on low-fidelity prototype evaluations involving the users.

Instead of having a user to train the tool, the paper in 2009 by Freeman et al. [8] proposes a tool to help users learn multi-touch and whole hand gestures in a tabletop surface by showing an example and feedback of the gesture for the user to learn.

For the machine learning approach used in this paper, an approach in a similar fashion is applied in 2007 by Cottrell *et al.* that presents a proof-of-concept plug-in to Eclipse for automatically determining correspondences as an early step in a generalization task [23].

III. Tool Contributions

IGT is based on the *Gesture Toolkit* (GT), a tool proposed by Khandkar *et al.* in 2010 [6]. To help understand IGT, some concepts of the GT framework and how IGT extends GT will be explained in the following.

The *Gesture Toolkit* is a toolkit that simplifies the multi-touch application development; it consists of a predefined set of common primitive gestures, a domain-specific language to define new gestures and a device-independent architecture that allows the application to run on different devices [6]. The definition of a gesture is created using the *Gesture Definition Language* (GDL) [6].

The GDL provides a set of *primitive* conditions to define the gesture. *Primitive* conditions include trigonometric and geometric calculations that can be used in GDL with other conditions. Primitives can contain values and they can be of two types: a fixed value or a range.

For example in Figure 1 the *primitive Touch path length* in *step1* (see quadrant 2) contains a fixed value of 295.98.

A gesture definition in the GT has three sections;

- a name that uniquely identifies a definition within the application;
- one or more validation blocks that contain combinations of primitive conditions; and
- a return block that contains one or more return types.

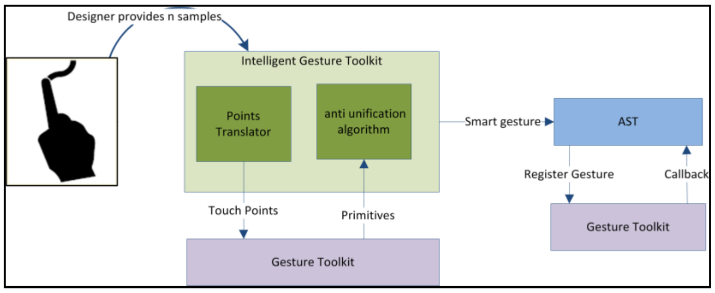


Figure 2 IGT Architecture

After defining a gesture in GT, a designer, with the help of functions and methods provided by the toolkit, needs to bind the gesture detection to trigger an event and to create the event (or *callback*). This step requires programming skills from the designer in GT, but is not necessary in IGT, as events are provided out of the box and just need to be selected from a list of events to be bound with gesture detection.

In IGT a value can also be a variable related to another *primitive* in the gesture definition, meaning that values in the definition of the language can be proportional between them. Gestures with proportional values are called smart gestures and will be explained in the following section.

Figure 1 shows a snapshot of IGT for the definition of a *check* gesture. In Figure 1, quadrant 1 is the canvas where the gesture should be performed for training the tool; it shows a feedback of the gesture. Quadrant 2 shows the definitions generated by the tool for the gestures performed. As samples are added, the previous sample is collapsed and the new one expanded. Quadrant 3 shows the smart gesture that was generated based on the samples provided in quadrant 2. Finally, quadrant 4 is a control section that to date has only two properties; to control the accuracy for the anti-unification algorithm and a checkbox to control whether the tool should try to break the gesture into steps or not.

The main modifications between IGT and GT are in the gesture definition phase. IGT creates another layer of abstraction in order to allow designers to create new gestures by training the tool through examples of the gesture performed by the designer. This allows designers with no programing skills, to create custom gestures. An example can be seen in Figure 1: by providing one sample of the *check* gesture on the left, IGT produces the definition on the right.

IV. Learning Gestures

A gesture definition needs to be as broad as necessary to surpass the nuances of different users performing the gesture in different moments. A gesture also needs to be as precise as possible to avoid conflict with other gestures and to be detected only when a gesture is intended by the user.

This problem requires a definition of gestures that is the most specific pattern among certain variations. In order to achieve this solution, IGT sees the problem as an anti-unification problem that, based on the definition given by Bulychev *et al.* in 2009 [4], finds the most specific template or pattern between two terms.

The implementation proposed in this paper, is an approximation of the anti-unification modulo theory, but as the anti-unification modulo theory is in general undecidable, the proposed implementation is a subset of the theory [24].

In order to gather the terms and the different nuances for the anti-unification process, IGT asks the designer to train the tool by performing examples of the gesture he wants to create. It is up to the designer to provide examples that cover all the nuances he desires the gesture definition to cover. It is in the designer's best interest to train the tool to learn the gesture in a way that provides the best outputs of the tool.

There is no conflict handling mechanism between different gestures necessary, as there is only one designer training the tool at a time and the samples of the gesture exist without any interaction with other gestures already defined.

IGT has a feedback mechanism that informs when a sample of a gesture provided by the designer to train the tool is outside the standard and confirms with the designer whether he really meant to provide that sample or if it was a mistake. A sample is considered outside the standard when less than 20% of its *primitives* match any of the previous samples.

The designer can keep the different sample, which creates a more general gesture, or the designer can also remove the sample and add another one. This implies a scheme where there is a dialog between the designer and the tool.

The samples of the gesture to train the tool can be a sequence of 2D points that can use fingers or the hand.

IGT doesn't need any previous training set to be calibrated or to function properly, but also, there is an established moment when the designer uses the tool to define the gesture. This means that once the definition of a gesture is finished, it will not be updated when it is used.

The anti-unification process in IGT works in two main steps; first as proposed by Nilsson in 2005 [5], it analyzes the *primitives* for one sample of the gesture and creates relations between the different steps of that gesture, creating relational rules.

Given a number n of samples of a gesture $\{x_1, x_2, \dots, x_{n-1}, x_n\}$, the algorithm creates an anti-unifier of the first two samples and then anti-unifies the analyzed anti-unifier with the next sample to create the next anti-unified pattern. If in this process it finds a sample that is outside the standard, it asks the user to confirm this sample or replace it for a new one. It repeats this process until it compares with the sample x_n. Then, based on the final pattern, the algorithm generates a rule that is the most specific pattern between all the samples provided.

In IGT the sequence of points from the samples are pre-analyzed and converted by the point translator as shown in Figure 2. This step looks for patterns in the sequence of the points to see if the gesture can be broken into smaller steps in order to help the anti-unification process.

For example, in Figure 1 there is one single stroke performed by a finger gesture that makes the *check* gesture. The point translator tries to identify patterns and breaks the *check* gesture into two steps, each one containing one line.

After the analysis, the point translator creates *Touch Points* that can be interpreted by the *Gesture Toolkit* framework which identifies the *primitives* that match the *Touch Points* (Ex: If the *shape* is a *line* or if it is a *closed loop*) and returns these *primitives* to the anti-unification algorithm.

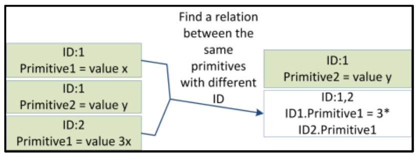


Figure 3 Relational rules

The *primitives* are grouped in a data structure as seen in Figure 3 that contains:

- Step in which the primitive is detected (ID:1);
- Name of the primitive (primitive2);
- Value of the primitive (value x).

As seen in Figure 3, relational rules involve one or more steps and allow the definition of relational values for the *primitives* of the gesture.

The second step of the anti-unification process analyzes the rules among the samples and creates the most compact way to represent the rules in the samples, creating a smart rule.

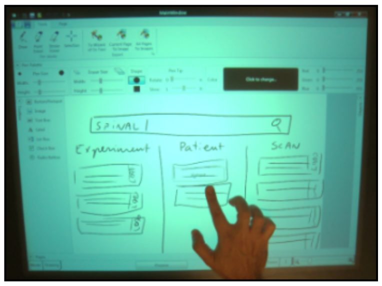


Figure 4 Example of a low-fidelity prototype in AST

With two samples provided for the same gesture, the second step of the anti-unification process looks for the same rules in the different samples and check if the proportions and relational rules found in the first step are consistent among the other samples.

V. IGT in a Low-Fidelity Prototype Evaluation

The current literature shows that low-fidelity prototypes improve the design of touch-based applications for multi-touch surfaces and allows a more user-centered design of applications [3], [22].

The study published by Derboven et al. in 2010 [19] introduces two low-fidelity prototype methods for multi-touch surfaces consisting of "*physical*" materials such as paper, cardboards and markers.

This paper proposes a solution to create and evaluate gestures contextualized in a low-fidelity prototype. To use low-fidelity prototypes, IGT is embedded in an existing tool called *Active Story Touch* (AST).

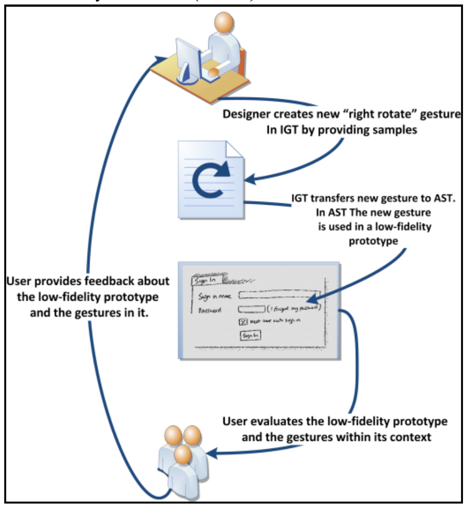

Designer creates new "right rotate" gesture In IGT by providing samples
IGT transfers new gesture to AST. In AST The new gesture is used in a low-fidelity prototype
User provides feedback about the low-fidelity prototype and the gestures in it.
User evaluates the low-fidelity prototype and the gestures within its context

Figure 5 Evaluation cycle of AST and IGT

AST is a tool proposed by Hosseini-Khayat et al. in 2011 that targets the creation of low-fidelity prototypes for gesture-based applications [3]. Figure 4 shows a low-fidelity prototype being created in AST.

To better illustrate a gesture evaluation in a low-fidelity prototype context, Figure 5 shows how a designer can create gestures using IGT and how this new gesture fits in a low-fidelity prototype evaluation using AST [3].

When creating a low-fidelity prototype in AST a designer can use IGT to create a custom gesture that will be used in the prototype. The designer creates the definition of the gesture in IGT as previously explained by providing samples of the gesture. IGT will create a definition of the gesture in GDL and make the new gesture available to be used in AST.

Having the newly created gesture available, the designer can bind it to some event and use it in the low fidelity prototype to trigger a behavior or functionality. It is important to mention that all this binding can be done without writing one single line of code.

As illustrated in Figure 5, a designer can create a custom "right rotate" gesture and bind it to switch between pages of the prototype. The low-fidelity prototype will be evaluated by the user which will use the custom "*right rotate*" gesture and give feedback to the designer about not only the prototype but also about the gestures used.

VI. Conclusion and Future Work

The approach given in this paper addresses a new challenge in the design of gesture based applications for multi-touch devices. Our tool provides designers with an easy way to create gestures that does not require any programming skills in order to come up with a bigger variety of gestures. Also this paper provides a scheme for the tool to evaluate the gesture in the application's context.

Another contribution of this paper is the anti-unification approach used here to learn new gestures, allowing definition of gestures that cover all the different nuances required for a more accurate recognition of the gesture.

The tool we described is currently in development, having the learning and detection of finger and hand gestures available.

By adopting one device, there are possibilities of different features that can be applied to the gesture learning tool, such as the learning of tag-less tangibles.

In order to analyze the tool's performance, an evaluation will be done in two parts. First, designers will evaluate IGT without any context. This will be done to obtain feedback on the anti-unification approach. This feedback will allow the comparison of performance of IGT with other similar tools in the field of gesture learning.

Second, designers will evaluate IGT in a low-fidelity prototype experiment and evaluate how important it is to define gestures for a low-fidelity prototype evaluation. This will provide feedback on the evaluation cycle proposed in this paper.

References

[1] Wobbrock, J. O., Morris, M. R., & Wilson, A. D. (2009). User-defined gestures for surface computing. *Proceedings of CHI '09*. Pages 1083-1092. New York, New York, USA: ACM Press. doi:10.1145/1518701.1518866.

[2] Hinrichs, U., & Carpendale, S. (2011). Gestures in the Wild : Studying Multi-Touch Gesture Sequences on Interactive Tabletop Exhibits. CHI 2011, Pages 3023-3032. May 7–12, 2011, Vancouver, BC, Canada.

[3] Hosseini-, A., Seyed, T., Burns, C., Maurer, F. (2011). Low-Fidelity Prototyping of Gesture-based Applications. EICS'11,Pages 289-294. June 13–16, 2011, Pisa, Italy.

[4] Bulychev, P. E., Kostylev, E. V., & Zakharov, V. A. (2009). Anti-unification algorithms and their applications in program analysis. In Proceedings of PSI 2009, Novosibirsk, Russia, June 15-19, 2009, pages 413-423, Springer, 2010.

[5] Nilsson, N. J. (2005). INTRODUCTION TO MACHINE LEARNING AN EARLY DRAFT OF A PROPOSED TEXTBOOK Department of Computer Science. Stanford University,Stanford, CA 94305.

[6] Khandkar, S. H., & Maurer, F. (2010). A Domain Specific Language to Define Gestures for Multi-Touch Applications. DSM:10, 17-OCT-2010, Reno, USA.

[7] S. Damaraju and A. Kerne. Multitouch.(2008) Gesture Learning and Recognition System. In Poster session presented at TABLETOP '08.

[8] Freeman, D., Benko, H., Morris, M. R., Wigdor, D., & Way, O. M. (2009). ShadowGuides : Visualizations for In-Situ Learning of Multi-Touch and Whole-Hand Gestures. Proceedings of the ACM ITS '09.

[9] Khandkar, S. H., Sohan, S. M., Sillito, J., & Maurer, F. (2010). Tool Support for Testing Complex Multi-Touch Gestures. ITS '10.

[10] Mitra, S.; Acharya, T. (2009). "Gesture Recognition: A Survey," Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on , vol.37, no.3, pp.311-324, May 2007 doi: 10.1109/TSMCC.2007.893280.

[11] Rabiner, L.R.(1989), "A tutorial on hidden Markov models and selected applications in speech recognition," Proceedings of the IEEE , vol.77, no.2, pp.257-286, Feb 1989 doi: 10.1109/5.18626.

[12] Wilson, A.D.; Bobick, A.F.;(1999) , "Parametric hidden Markov models for gesture recognition," Pattern Analysis and Machine Intelligence, IEEE Transactions on , vol.21, no.9, pp.884-900, Sep 1999 doi: 10.1109/34.790429.

[13] Choi, H., Paulson, B., & Hammond, T. (2008). Gesture Recognition Based on Manifold Learning, 247-256. Proceeding SSPR & SPR '08.

[14] Wu, M., & Balakrishnan, R. (2003). Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. *Proceedings of UIST '03*, *5*(2), 193-202. New York, New York, USA: ACM Press. doi:10.1145/964696.964718

[15] M. Karam and M. C. Schraefel.(2005) A taxonomy of gesture in human computer interactions. Technical report, Technical Report ECSTR-IAM05-009, Electronics and Computer Science, University of Southampton, 2005.

[16] Lao, S., & Wang, P. (2009). A Gestural Interaction Design Model for Multi-touch Displays. *In Proceedings of the 2009 British Computer Society Conference on Human-Computer interaction, , 440-446.*

[17] Morris, M. R., Wobbrock, J. O., & Wilson, A. D. (2010). Understanding Users Preferences for Surface Gestures. *Interfaces*, 261-268.

[18] Bobick, A. F., & Wilson, A. D. (1997). A state-based approach to the representation and recognition of gesture. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, *19*(12), 1325–1337. IEEE.

[19] Derboven, J., Roeck, D. D., & Verstraete, M. (2010). Low-Fidelity Prototyping for Multi-Touch Surfaces. Presented in the workshop Engineering Patterns for Multi-Touch Interfaces held in EICS 2010, 2nd edition, Berlin, Germany.

[20] GestureWorks, a multitouch application framework for Adobe Flash and Flex. 2010. Available at http://gestureworks.com/ accessed in February 2012.

[21] Windows Presentation Foundation, Available at http://msdn.microsoft.com/en-us/library/ms754130.aspx, Accessed December 2010

[22] Olmsted-Hawala, E. L., Romano, J. C., & Murphy, E. D. (2009). The use of paper-prototyping in a low-fidelity usability study. *2009 IEEE International Professional Communication Conference*, 1-11. IEEE. doi:10.1109/IPCC.2009.5208693.

[23] Cottrell, R., Chang, J. J. C., Walker, R. J., & Denzinger, J. (2007). Determining detailed structural correspondence for generalization tasks. *Proceedings of ESEC-FSE '07*, Pages 165-174. New York, New York, USA: ACM Press. doi:10.1145/1287624.1287649.

[24] Jochen Burghardt. (2005). E-generalization using grammars. Artif. Intell. 165, 1 (June 2005), 1-35. DOI=10.1016/j.artint.2005.01.008 http://dx.doi.org/10.1016/j.artint.2005.01.008