A Language to Define Multi-Touch Interactions

Shahedul Huq Khandkar
Department of Computer Science
University of Calgary, Canada
s.h.khandkar@ucalgary.ca

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

ABSTRACT

Touch has become a common interface for human computer interaction. From portable hand held devices like smart phones to tabletops, large displays and even devices that project on arbitrary surfaces support touch interface. However, at the end, it is the applications that bring meaning for these technologies to people. Incorporating a touch interface in application requires translating meaningful touches into system recognizable events. This process often involves complex implementations that are sometimes hard to fine tune. Due to the lack of higher-level frameworks, developers often end up writing code from scratch to implement touch interactions in their application. To address this, we present a domain-specific language to define multi-touch interaction that hides the low level implementation complexities from application developers. This allows them to focus on designing touch interactions that are natural and meaningful to the application context without worrying about implementation complexities.

ACM Classification: H5.2 [Information interfaces and presentation]: User Interfaces. - Graphical user interfaces.

General terms: Algorithms, Performance, Design, Reliability, Human factors, Languages.

Keywords: Gesture, Domain-Specific Language.

INTRODUCTION

While multi-touch interfaces exist in research labs since 1980's [2], they have been introduced to the general public quite recently. However, many of the devices we use in our daily life already started adapting this technology. For example, smart phones, interactive displays, digital tabletops in schools for kids and so on. As these devices become increasingly affordable, it is essential to create new applications and extend existing ones to support touch-based interaction.

The motion of meaningful touch(s) to interact with the system is called gesture. A gesture may include touches of multiple fingers, hands or arbitrary tangible objects. Gestures can be as simple as a single touch or complex set of touches in multiple steps. At present, application developers predominantly use software development kits (SDK) provided by hardware vendors that are hardware specific. These SDKs provide the necessary infrastructure to communicate between hardware and software as well as to some extent touch enabled user interface widgets. However, they provide limited high level frameworks for programming against the touch interactions. As a result, developers often end up building touch interaction related modules and gestures from scratch.

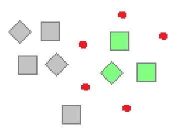
We present a gesture definition language (GDL) that significantly simplifies the process of defining new gestures and allows them to be used across multiple hardware platforms. This helps to hide the low level implementation details from application developers and therefore helps them focus more on the functionality and usability of the application. Our approach also provides an extensibility framework to easily extend the language to adapt new features and functionalities as devices evolve. GDL is part of TouchToolkit [1]- an application framework that provides a device independent platform for multi-touch applications.

RELATED WORK

Wobbrock [3] proposed a gesture recognition system that recognizes gestures by comparing them with base templates. While it allows the definition of new gestures by adding additional templates, it does have a number of limitations. For example, it cannot detect gestures in a continuous motion stream and only gestures with explicit start and end points can be processed. Gestures like a lasso which can be of an arbitrary shape cannot be detected using this approach. GDL provides the flexibility to use a combination of primitive conditions to define such gestures.

Kratz [4] proposed another approach for 3D acceleration sensors that relies solely on simple trigonometric and geometric calculations. His approach requires considerably less training data than some other recognizers. However, it too suffers from limitations such as a smaller gesture vocabulary size and it cannot process gestures with continuous motion. GDL also provides primitive conditions for trigonometric and geometric calculations that can be used with other conditions.

Furthermore, they were proposed for single point systems and are often not suitable for multi-touch scenarios that involve sequences of multiple concurrent touches. In GDL, the gesture processor internally handles the concurrency issues that may arise in multi-user multi-touch scenario.

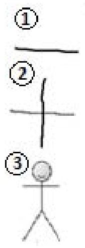


```
name: multi-select

validate
    Touch state: TouchDown TouchMove and
    Touch limit: 3..5 and
    Distance between points: 50..250 and
    Distance between points: unchanged 10%

return
    Touch points
```

Figure 1: Defining a gesture to select an area and allow to drag all elements inside the selected area



```
name: Actor
validate as firstLine
    Touch state: TouchUp and ...
validate as secondLine
    Touch state: TouchUp and ...
validate
    firstLine perpendicularTo secondLine and..
return
    Position, Bounding box
```

Figure 3: Defining a multi-step gesture using GDL

LANGUAGE DESIGN

GDL is a domain-specific language designed for defining gestures. More details including video tutorials are available at the project website [1]. A gesture definition contains three sections: a *name* that uniquely identifies a definition within the application; one or more *validate* blocks that contain primitive conditions; and finally the *return* block that contains one or more return types that the application receives when a gesture is detected.

Figure 1 shows a gesture definition for selecting an area using multiple touch points (e.g. all fingers of one hand to select a region) and drag all elements inside the selected area. The validation logic is defined using a combination of primitive conditions (e.g. Touch state). The return type "Touch points" means that the framework will provide the touch point data to the application via the callback method when the gesture is detected. An application developer can use the utility functions to get the objects within the boundary points and update its position inside the callback to provide drag functionality. Figure 2 shows how you can subscribe to a gesture event in a WPF application. The first parameter defines the scope of the gesture (i.e. LayoutRoot), next is the name of the gesture definition and finally the reference of the callback method.

```
G.EventManager.AddEvent(LayoutRoot, "multi-select", Callback);
void Callback(UIElement sender, GestureEventArgs e)
{
    var touchPoints = e.Values.Get<TouchPoints>();
    ...
}
```

Figure 2: A code snippet to use a gesture in WPF application

Table 1 shows some examples of primitive conditions. Multiple primitive conditions are virtually connected like a chain using the logical operators (e.g. and). The validation process follows the lazy evaluation approach where it starts from the first primitive condition in the chain and it only passes the valid data set (or multiple possible sets) to the next condition in the chain. This allows the system to improve performance.

Table 1: Example of primitive conditions

Primitive Condition	Description
Touch limit: 1..4	Range of touch points allowed in gesture
line1 perpendicularTo line2	Geometric condition between two partial results
Distance between points: increasing	Pattern in change of distance between touch points

GDL also supports multi-step gestures. Simply add multiple validate blocks and the compiler will consider each block as a step and perform the validation in the order it is defined. Figure 3 shows the code snippet of a multi-step gesture that represents the "actor" object of Use Case diagram in a Unified Modeling Language (UML) designer tool. The last *validate* block uses the results of previous *validate* blocks which may also contain multiple sets of valid results. The framework internally handles concurrency scenarios that may arise in multi-user scenarios.

EXTENSIBILITY

Multi-touch devices are evolving at a great speed. Until recently, devices were mostly providing touch points and user identification for some specific devices (i.e. Diamond Touch). Now some devices can provide touch directions (i.e. Microsoft Surface) and information about the pressure of a touch (i.e. UnMousePad). GDL provides necessary interfaces to create new primitive conditions and return types to extend the language. It also provides Visual Studio project templates to simplify the process.

FUTURE WORK

At present, GDL allows to define gestures for touch interactions. In the future, we would like to include support for other touch based inputs like Smart Tags and tangible objects. While IDE support to some extent includes syntax highlighting, we also like to provide debugging support in the future.

CONCLUSION

We introduced a domain-specific language to define gestures for multi-user, multi-touch scenarios. It helps the developers to focus on designing gestures that are most natural and meaningful to application's context without worrying about low level implementation details. It also separates the concerns into different levels. As a result, each of them can be improved separately and shared across multiple projects.

REFERENCES

1. TouchToolkit, 2010. http://touchtoolkit.codeplex.com

2. Lee, SK., Buxton, W., and Smith K.C. A Multi-Touch Three Dimensional Touch-Sensitive Table. CHI '85

3. Wobbrock, J. O., Wilson, A. D and Li, Y. Gestures without libraries, toolkits or training: a 1 recognizer for user interface prototypes. UIST '07

4. Kratz, S., and Rohs, M. A $3 gesture recognizer: simple gesture recognition for devices equipped with 3D acceleration sensors. IUI'10