A Flexible Mapping Editor for Multi-touch Musical Instruments

Greg Kellum
Geneva Music Conservatory
Rue de l'Arquebuse 12
CH-1211 Genève
greg.kellum@cmusge.ch

Alain Crevoisier
University of Applied Sciences Western
Switzerland (HES-SO / HEIG-VD & HEM-GE)
Rue Galilée 15, CH-1400 Yverdon,
alain.crevoisier@heig-vd.ch

Abstract

This paper introduces a flexible mapping editor, which transforms multi-touch devices into musical instruments. The editor enables users to create interfaces by dragging and dropping components onto the interface and attaching actions to them, which will be executed when certain user-defined conditions obtain. The editor receives touch information via the non-proprietary communication protocol, TUIO [9], and can, therefore, be used together with a variety of different multi-touch input devices.

Keywords: NIME, multi-touch, multi-modal interface, sonic interaction design.

1. Introduction

The SurfaceEditor is a mapping editor for multi-touch surfaces. It enables users to divide a surface up into different regions containing different components for interaction. These components may consist of shapes like squares and triangles, traditional components like buttons and sliders, or user-created plug-ins for particular application domains.

The SurfaceEditor has a very flexible activation architecture. Users can define different conditions under which a component will execute actions like sending a MIDI message. For instance a user could configure a button to trigger a MIDI note-on when they press on it, but they could just as well configure it to trigger when a finger starts dragging inside the right side of the button.

The SurfaceEditor works with any multi-touch device that supports the TUIO protocol [9]. It can be used with a table-top multi-touch instrument such as the reacTable [8], with a multi-touch screen with a TUIO bridge, or on any arbitrary flat surface using the MultiTouch Everywhere technology presented in Crevoisier (2008) [4]. It supports not only touch interaction but also interaction with tangible objects.

The SurfaceEditor is freely available in a community edition at: www.future-instruments.net.

2. Prior work

The development of multi-touch technologies as well as component based interfaces has already been extensively documented elsewhere [25]. So, in this section we will consider solely editors created for multi-touch systems.

The Lemur, a multi-touch screen from Jazz Mutant, comes with an editor for creating interfaces for the screen called the Jazz Editor. The Jazz Editor (in the current version 2.0) contains components such as buttons, sliders, knobs, multi-balls, ranges, switches, and so on, which can be configured to send MIDI and OSC messages. Users create interfaces using the Jazz Editor on their computer, and then, they transfer these interfaces to the Lemur.

The SurfaceEditor differs from the Jazz Editor in a number of ways. The Jazz Editor is a proprietary application, which works only with the Lemur multi-touch screen, while the SurfaceEditor can be used with any number of input devices as long as they support the TUIO protocol. The SurfaceEditor support interaction with not only fingers but also with objects marked with fiducials [3]. The SurfaceEditor is used on the user's computer while the Jazz Editor runs on the Lemur screen's processor, and for this reason the Jazz Editor is not extensible with plug-ins while the SurfaceEditor can be extended with three different categories of plug-ins. And the SurfaceEditor's activation architecture is more flexible than that of the Lemur's in that it allows users to define conditions for the execution of actions as well as to define new actions if they wish.

3. Activation Architecture

One central aim when designing the SurfaceEditor was to find the model of surface interaction that would best allow users to define configurable conditions for the execution of actions.

When considering this aim, we found it helpful to think about a handful of actual use cases. For example, it would be useful for musicians to be able to trigger MIDI notes only when the contact energy of their fingers hitting the surface was above or below a certain level. Or one could imagine a scenario in which a user might want to use one finger to control an application but might accidentally or

intentionally put more fingers down in a component. One could also imagine a scenario in which a user might want to trigger different actions when there were different numbers of fingers in a component just as one can with the multi-touch mouse pad of Apple's MacBook Pro.

One way to analyze these scenarios would be to say a number of events are occurring, one is waiting for a particular type of event or set of events to occur, and one would like to filter out or otherwise ignore all other events. And when the particular type of event that one is waiting for occurs, one would like a particular action to be performed. Based on this analysis we implemented an event, filter, and action paradigm of handling user input in the hopes of achieving the desired level of user configurability.

When TUIO messages arrive, they are analyzed with regards to a particular component and converted into events. There are both discrete as well as continuous events, due to the fact that there are both discrete as well as continuous types of actions; for example a discrete action might send a MIDI note-on message while a continuous action might send a MIDI continuous control message. Discrete events include when a user touches down in a component, starts moving his finger in a component, slides out of the component, or lifts up in the component. Continuous events are sent continuously while a finger is in a component or while a finger is moving in a component. Converting the TUIO messages into these semantically meaningful events provides the first level of configurability, because users must choose which type of event they would like an action to be triggered by.

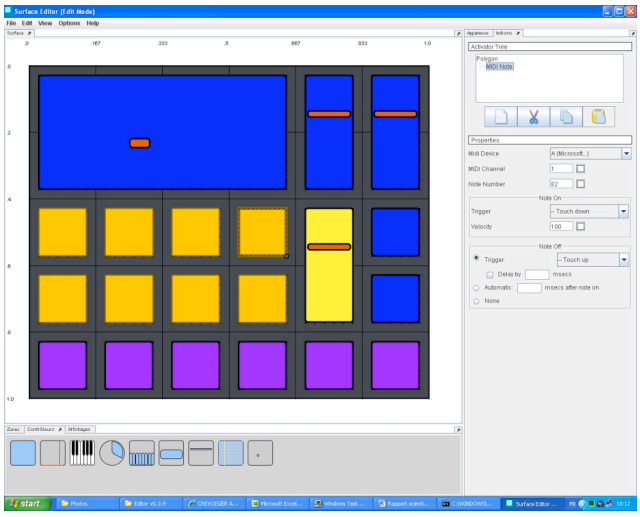

Figure 1 The Surface Editor's main page.

These events are then passed through a filter chain before they reach an action. There are parameter filters, which allow only events with a parameter between a certain range of values to pass through, e.g. an energy value between 0.5 and 1. There are ordering filters, which order a list of events using a given ordering criteria such as each event's age or horizontal position and then selects an event with a particular index out of the list. These can be used to specify for example that only the finger that was placed in a component first should be passed. There are count filters, which only allow lists of events of a certain size to pass through. These can be used to specify for example that the events should be passed through if and only if there are two fingers in the component. And there are temporal filters which allow only a certain number of events to pass through in a given time interval, e.g. one every 50 milliseconds.

After the events are filtered, if any events remain, they cause an action to be executed, and this action may possibly use the events' parameters while executing. Actions can do anything from sending a MIDI or OSC message to jumping to the next page of an interface or saving an interface's state.

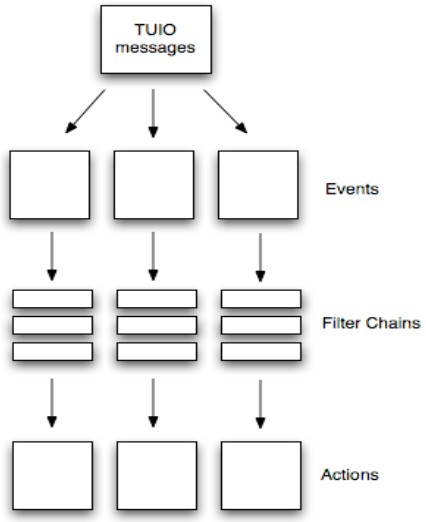

Figure 2 Event, filter, and action design pattern.

This architecture for handling the user interaction did provide much of the user configurability that we sought, but it was necessary to make a few departures from the event model to incorporate some important use cases. The ordering filter for instance did not work as users expected. One user wanted to use this filter to configure a component to respond to only one finger's touch down event. This filter would only filter out a second touch down event, however, if two events arrived in the same frame, but usually, one touch down event would arrive in one frame while a second one would arrive some frames afterward. So, it became apparent that users wanted to use this filter to filter out additional fingers not additional finger events, and events, which are by their nature instantaneous, were in this case not a good way to model these persistent objects. It was possible to solve this problem and a similar problem that arose with the count filter by generating dummy events that made these filters behave in the desired manner.

This activation architecture is accessible to the users of the SurfaceEditor in two pathways: simple and advanced. In the simple pathway actions are wrapped up together with the most likely default activation condition for the selected component, and the user is not given access to the event filters. So, when the user selects a MIDI note action for a button for instance, it is already preconfigured to send the note-on message when the button is pressed and the note-off message when the button is released. This pathway allows users to quickly and easily set up components to do the things that they most often will want them to do. In the advanced pathway on the other hand, users have access to all of the activation conditions including the filters, but they must do a little bit more work setting things up.

4. Components

Users create interfaces with the SurfaceEditor by dragging and dropping components. There are different categories of components. There are traditional components, such as knobs, buttons, keyboards, multi-sliders, and so on. There are plug-ins, which users can create in the Java language or by using the Processing language to generate Java classes. And there are zones, which are components for creating shapes.

4.1.1 Controllers

The SurfaceEditor contains a number of traditional control components including buttons, envelopes, keyboards, ranges, dials, sliders, multi-sliders, sequencers, and XY pads. Some of these controllers can be used in new ways, however, due to the underlying tracking technology. For example one can use drag information to control the pitch bend or volume of notes triggered on a keyboard. There are also some new controllers made possible by multi-touch. For example in the figure below there is a 3D wave controller; the XY position is given by pressing down in the controller while the Z value is given by moving the adjacent slider.

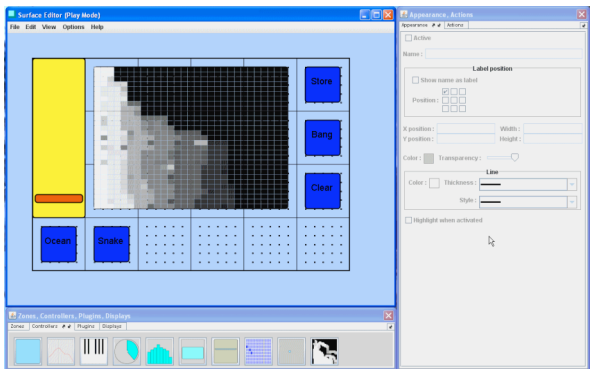

Figure 3 A 3D wave controller in the SurfaceEditor.

4.1.2 Plug-ins

The SurfaceEditor supports three kinds of plug-ins: controllers, surface components, and actions. Controllers and surface components are both visual components, which are placed on the interface. Controllers differ from surface components, however, in that controllers use the SurfaceEditor's activation architecture while surface components do not. Controllers are components that produce some sort of output messages, e.g. MIDI or OSC messages, while surface components are primarily used for visual display. (Surface components can also produce output messages as well, but then they do so without going through the SurfaceEditor's activation architecture.) The SurfaceEditor can also be extended with new types of actions that can send new types of messages. For example if someone wanted to use the SurfaceEditor to control lighting, they could create an action to send out DMX messages, which would interface with the driver of a particular USB to DMX converter. In the figure below two surface components are shown, which are used for drawing: a color chooser and a canvas. This example shows that the SurfaceEditor is not limited to musical applications and can be used for other purposes as well.

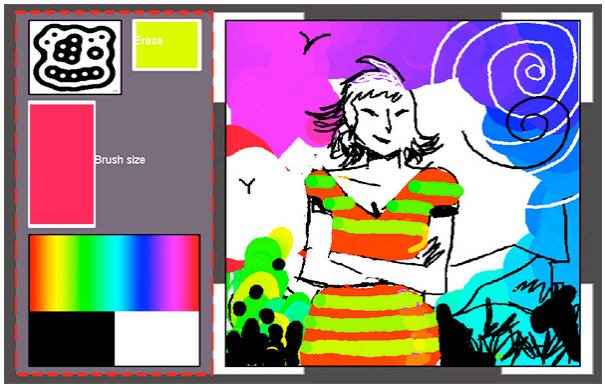

Figure 4 A surface component plug-in used for drawing.

4.1.3 Zones

Zones are components that can be given an arbitrary shape. There are five different ways of creating zone components. There are rectangular, elliptical, and triangular shapes, which can be stretched and rotated into different sizes. There is a tool for creating polygons by connecting line segments, and there is a free hand drawing tool for drawing arbitrary shapes. Using these five tools one can draw any shape, which means that one can create interfaces by dividing existing images into different parts. In the figure below a replica of the map of Europe was created using zones for a children's game. Interfaces could be made in a similar manner to enable the sonification of images.

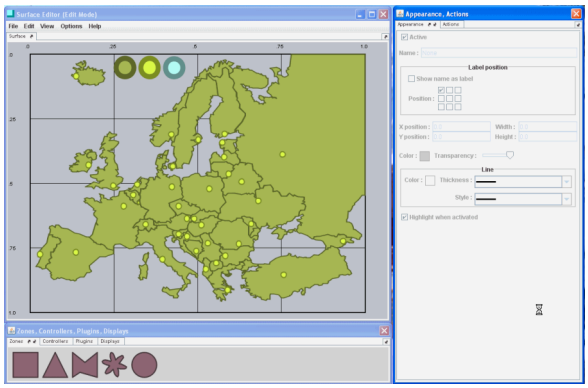

Figure 5 A map of Europe created using zones for a game.

5. Other features

The SurfaceEditor provides all of the editing functionality one would expect from an editor. It supports editing operations such as cut, copy, paste, undo, and redo. It supports user definable interface sizes and grid dimensions. It supports user-defined variables, which can be used to transform the values of input parameters. And it offers a number of features for using it as a musical instrument such as multiple interface pages, which may be changed with a MIDI foot pedal, and support for multiple MIDI and OSC outputs.

The SurfaceEditor supports communication between components. This allows users to control the behavior of complex components like sequencers with simpler components like buttons. And in the future it will allow the values of components to be updated by external applications such as Max/MSP.

6. Acknowledgments

The project presented here is supported by the Swiss National Funding Agency and the University of Applied Sciences. Special thanks to all the people involved in the developments presented here, in particular Pierrick Zoss his help in the programming of the SurfaceEditor.

References

[1] Aimi R.M. "New Expressive Percussion Instruments," Masters Thesis, Massachusetts Institute of Technology, 2002

[2] Blaine, T. and Perkis, T. Jam-O-Drum, A Study Interaction Design. *Proc. ACM DIS 2000 Conference*. NY: ACM Press (2000).

[3] Costanza E., Huang J., "Designable Visual Markers," in ACM CHI2009, April 2009, Boston, MA, USA.

[4] Crevoisier, A., Kellum, G., "Transforming Ordinary Surfaces into Multi-touch Controllers", in *Proc. of the Conf. on New Instruments for Musical Expression (NIME)*, 2008.

[5] Crevoisier, A. Future-instruments.net: Towards the Creation of Hybrid Electronic-Acoustic Musical Instruments, *Proc. of the CHI workshop on Sonic Interaction Design*, 2008.

[6] Hahn, J.Y. Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection, *Proc. of the ACM Symposium on User Interface Software and Technology (UIST)*, 2005.

[7] Jones, R., "Intimate Control for Physical Modeling Synthesis," M.Sc. Thesis, University of Victoria, 2008

[8] Jordà, S., Kaltenbrunner, M., Geiger, G. and Bencina, R., The reacTable*, *Proceedings of the International Computer Music Conference* (ICMC2005), Barcelona (Spain).

[9] Kaltenbrunner, M., Bovermann, T., Bencina, R. and Costanza, E., "TUIO - A Protocol for Table Based Tangible User Interfaces", *Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation* (GW 2005), Vannes (France).

[10] M. Mathews, "The Radio Drum as a synthesizer controller", In *Proc. Int. Computer Music Conference*, 42-45, 1989

[11] McAvinney, Paul, "The Sensor Frame - A Gesture-Based Device for the Manipulation of Graphic Objects", Carnegie-Mellon University, 1986.

[12] Oliver, J., Jenkins, M., The Silent Drum Controller: A New Percussive Gestural Interface, *Proceedings of the International Computer Music Conference*, 2008.

[13] Patten, J., Recht, B. and Ishii, H., Audiopad: A Tagbased Interface for Musical Performance. *Proc. Conference on New Interface for Musical Expression,* (2002), 24-26.

[14] SmallFish: http://hosting.zkm.de/wmuench/small_fish, Referenced October 20, 2006.

[15] Wilson, A. PlayAnywhere: A Compact Tabletop Computer Vision System, Proceedings of the ACM Symposium on User Interface Software and Technology (UIST), 2005.

[16] Wilson, A. TouchLight: An Imaging Touch Screen and Display for Gesture-Based Interaction, *Proceedings of the International Conference on Multimodal Interfaces*, 2004.

[17] http://www.future-instruments.net

[18] http://www.jazzmutant.com

[19] http://www.surface.com

[20] http://www.celluon.com

[21] http://www.lumio.com

[22] http://nuigroup.com

[23] http://www.naturalpoint.com

[24] www.tactex.com

[25] http://www.billbuxton.com/multitouchOverview.html