TUIOFX - A JavaFX Toolkit for Shared Interactive Surfaces

MIRKO FETTER, Human-Computer Interaction Group, University of Bamberg
DAVID BIMAMISA, Human-Computer Interaction Group, University of Bamberg
TOM GROSS, Human-Computer Interaction Group, University of Bamberg

Building multi-touch multi-user applications for Shared Interactive Surfaces is a complex endeavour that requires fundamental knowledge in touch enabling hardware, gesture recognition, graphical representation of digital information and multi-user interaction. While several specialised toolkits help developers in this effort, we identified a variety of challenges with these toolkits, as for example the lack of cross-platform support, the limited number of touch-enabled multi-user widgets, missing documentation, and lacking community support — all raising the barriers to entry. In this paper, we present TUIOFX, a toolkit for developing multi-touch, multi-user applications for Shared Interactive Surfaces in Java, which tackles all of the identified problems. The sophisticated implementation of TUIOFX adds support for TUIO-enabled hardware and multi-user interaction under the hood of JavaFX, and leaves the well-learned JavaFX API for the developers fully intact – thus allowing particularly novices a very quick start. In this paper we provide the technical insights, in the concepts and their elegant implementation.

CCS Concepts: • **Human-centered computing → User interface toolkits systems** • Human-centered computing → Collaborative and social computing devices

KEYWORDS

Multi-touch, multi-user, shared interactive surfaces, toolkit, SDK, JavaFX

ACM Reference format:

Mirko Fetter, David Bimamisa, Tom Gross. 2017. TUIOFX - A JavaFX Toolkit for Shared Interactive Surfaces. *Proceedings of the ACM on Human-Computer Interaction* (June 2017), 18 pages.
DOI: 10.1145/3095812

1 INTRODUCTION

Over the last two decades, many research projects and studies have pointed out that *Shared Interactive Surfaces* (*SIS*) like multi-touch tabletops offer a number of promising opportunities and interaction techniques. The most promoted and investigated opportunities associated to *SIS* involve intuitive interaction via direct-touch manipulation, easy-to-use and easy-to-learn user interfaces for novice users and support for multi-user interaction and collaboration [41].

However, despite the availability of affordable tabletop and wall-mounted solutions, to date interactive surfaces have not yet reached widespread adoption in productive settings beyond museum exhibitions,

Author's addresses: M. Fetter, D. Bimamisa and T. Gross, Human-Computer Interaction Group, University of Bamberg, Bamberg, Germany.

showrooms or research projects. One observation that can be made is, that despite the general availability of hardware, the amount of off-the-shelf productive software is almost zero. One reason that can be identified is that applications often need to be customised to the specific hardware. Many existing frameworks and toolkits for developing applications have very specific hardware requirements and are sometimes even limited to hardware from a specific vendor (e.g., MS Surface 2.0 SDK [27]). Also, the shortage of applications might be explained by the fact that developing user interfaces for interactive surfaces is still complex and often requires special knowledge, in spite of the existence of several multi-touch user interface toolkits.

In our attempt to explain, why existing toolkits are still inadequate in significantly reducing the effort and complexity involved in building multi-touch applications, we want to highlight three issues, we identified earlier [8] and discussed with the community:

- First, most toolkits rarely provide a full set of ready-to-use widgets—let alone complex and customisable, controls—optimised for touch interaction. This is for once, because the developers of such toolkits expect tabletop applications to rely on highly customised widgets and therefore largely focus on giving application developers the freedom to create any type of widgets themselves, rather than providing novel and solid multi-touch widgets suitable for productive tasks on tabletops. Further, often such toolkits are developed from scratch, meaning that the toolkit developers have to implement every needed widget by themselves (e.g., MT4j [24]). We argue that a toolkit for tabletops should at least provide a set of traditional desktop widgets optimised for touch. This way designers and programmers are able to draw on well-known and already learnt interaction techniques with a solid foundation similar when developing with traditional widget toolkits like WPF, Cocoa, or Java Swing for desktop software. The programmers can than develop own, specialised widgets and controls on top.
- Second, strong concepts to support multiple users are often missing or are only rudimentary implemented in such toolkits [41]. Most times, it is left completely to the developers of an application, how they deal with parallel input and resolve input conflicts (e.g., [32]). We share the belief of Roseman and Greenberg [36] that well-defined concepts are needed on a toolkit level to support and guide developers when realising multi-user applications.
- Third, a lot of prototypes that emerged from the research community or developed by individuals unfortunately lack comprehensive documentations and extensive support in form of forums and news groups. This often leaves developers on their own, without help, with their specific problem. Sometimes, this goes hand in hand with constantly evolving, semi-structured APIs and immature architectures that are the results of growing prototypes. The wide adoption of the now discontinued Surface SDK [27] by Microsoft, partly showed how a good documented and well-structured toolkit can help developers, even though it offered only rudimentary multi-user support and was limited to one platform. Also Groupkit [36] showed how founding on a well-known UI Framework can benefit the spreading of a toolkit.

To address the mentioned shortcomings, we propose TUIOFX, a toolkit for building multi-touch multi-user cross-platform applications for *Shared Interactive Surfaces*. The TUIOFX toolkit was conceived to enable developers to easily develop appealing cross-platform applications. To achieve this, TUIOFX is implemented on top of JavaFX, a cross-platform UI toolkit that comes with a huge number of ready-to-use standard UI widgets (called *controls*). To support multi-touch and multi-user interaction on *Shared Interactive Surfaces* TUIOFX optimises the look-and-feel of theses JavaFX controls and enables means for parallel interaction of multiple users for these traditional JavaFX widgets. Also JavaFX's event handling infrastructure offers support for handling multi-touch inputs and some standard gestures (i.e. scroll, swipe, zoom and rotate gestures) out of the box. However, JavaFX only listens to multi-touch input events provided by the underlying operating system (OS), thus the availability and interpretation of some gestures may differ between OSs. Therefore, TUIOFX also includes its own platform-independent gesture recognition engine based on the TUIO protocol [16]. This ultimately removes the limitation that almost all OS only allow detecting one gesture at a time—as they depart from a single user perspective—and thus lays the ground for enabling JavaFX as a development tool for *Shared Interactive Surfaces*.

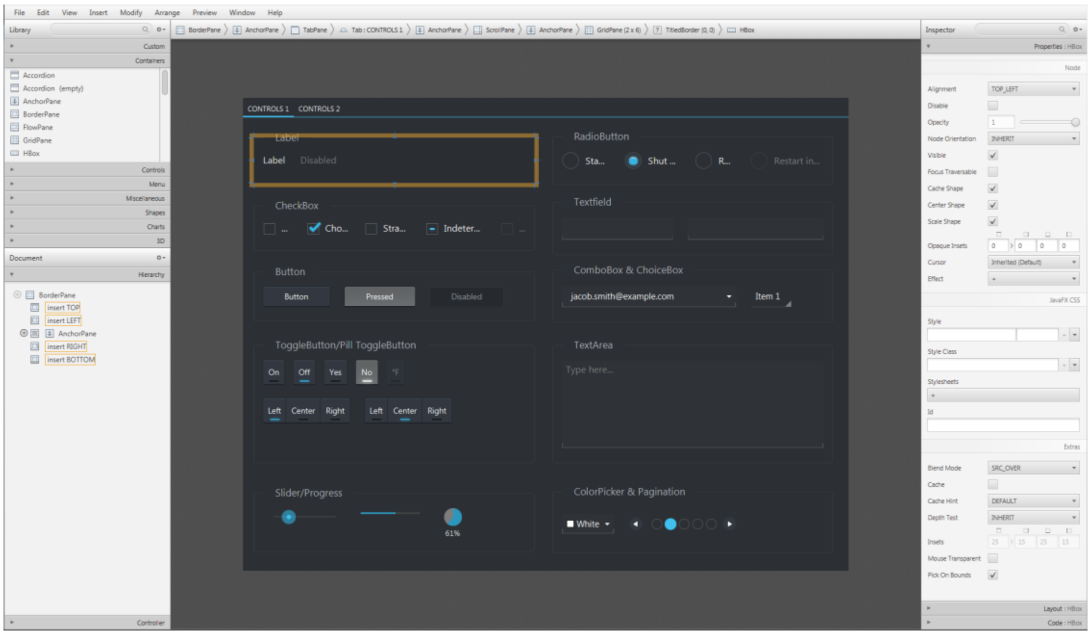


Fig. 1. Visual editing TUIOFX apps and custom controls in Oracle's JavaFX Scene Builder.

2 REQUIREMENTS AND DESIGN DECISIONS

Ideally, the interaction design of exploits knowledge people have from face-to-face collaboration, where people use a shared surface like a table or a whiteboard as supporting medium to work simultaneously on a common activity. They share information and objects placed on the surface and they use them as conversational props during the collaboration [3,22,44]. It is obvious that these characteristics of physical tables and whiteboards, which afford multiple users to come together to work collaboratively on common tasks around a single shared display, also hold for *SIS*. With the exciting prospect of tabletop and whiteboard affordances in group collaboration, supporting effective multi-user interaction on and around interactive surfaces has evolved to an important topic in the HCI research community (e.g., [10,11,29,33,35,40,41]). In fact, many interactive surfaces applications leverage concurrent group collaboration by allowing multiple users to share and manipulate digital objects simultaneously without solely relying on turn-taking activities. In this work, we use the term *Shared Interactive Surfaces* to refer to multi-touch tabletops and large wall-mounted surfaces with support for true multi-user interaction (i.e. parallel group work that is relatively free from turn-taking). Thus, they enable multiple users at the same location to jointly work together on a common task in parallel. In this sense SIS can be seen as an evolution of the concepts behind the idea of single display groupware SDG [43]—with the significant difference that interactive surfaces enable simultaneous input from several users via *multi-touch* opposed to multiple connected mice. That is, the underlying technology of a *SIS* (e.g., Frustrated Total Internal Reflection (FTIR), Diffused Illumination (DI), In-Cell Touch) is able to sense multiple simultaneous points of contact on their surface. Founding on these multi-touch input capabilities, interactive surfaces provide a new design space for novel interaction techniques that encompass a wide range of input modalities, from direct-touch manipulation (i.e. using fingertips, whole-hand and pen) to interaction with physical objects (tangibles). Thereby one downside of most current commercial hardware is the inability to identify from which user a touch originated—a condition we refer to as *anonymous touch input* [9].

So far a considerable effort has been put into providing multi-touch toolkits, to support developers in their endeavour of programming applications for *SIS* and their specific characteristics. Based on our own experience with those toolkits and prior research in the HCI and CSCW community, we established a set of requirements and design decisions to guide the design of a new toolkit: TUIOFX.

2.1 Building on Existing Technology

The key requirement guiding the design of TUIOFX was to build on existing technology to minimise the learning effort for new users and maximise the available support from a vivid developer community. From our own experience with various toolkits, we learned that such toolkits often are niche solutions, sparsely documented and supported, and with a limited lifespan. By building on JavaFX our aim was to leverage on a huge, *active community* that is continuously evolving. The aim was to benefit from this *well-documented* base technology, by smoothly implementing everything beneath the surface of JavaFX and leaving the original API untouched to the greatest possible extent. This way, also the documentation and support requirements for TUIOFX can be kept minimal, as existing documentation materials and examples provided by JavaFX itself are sufficient to understand most aspects of programming applications with TUIOFX. From our experience, when handing out TUIOFX to developers in a few smaller projects and workshops so far, the adoption effort—even for inexperienced JavaFX programmers—is negligible. Also, the minimal invasive API design of TUIOFX allows that many existing tools—like the JavaFX Scene Builder for *visual editing* (cf. Fig. 1)—can be reused for developing multi-user, multi-touch applications.

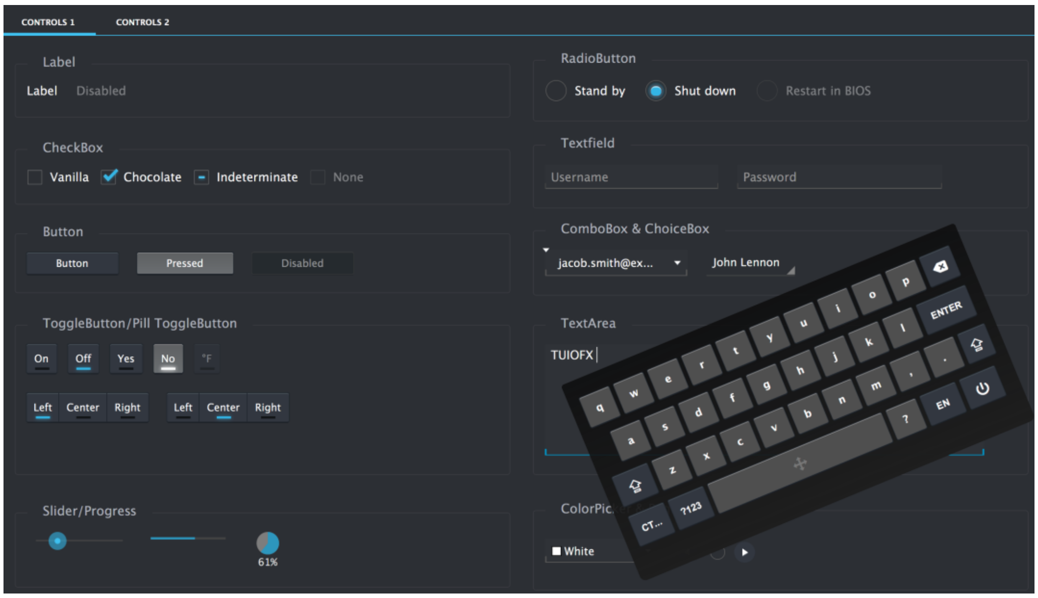


Fig. 2. A selection of only a few of the available TUIOFX widgets—with one open TUIOFX on-screen keyboard.

2.2 Providing Platform Independence

An important design goal of TUIOFX is to provide a platform-independent toolkit for building multi-touch multi-user applications. The aim was to provide developers with the freedom to choose from different technologies for the SIS hardware as well as for the underlying operating system. To achieve this goal, first, TUIOFX is simply implemented in the cross-platform language Java. Second, TUIOFX provides a platform-independent gesture recognition engine based on the TUIO [16] protocol. This way, TUIOFX does not rely on the gesture data acquired by the underlying OS, as most desktop OS are expecting one user and therefore are limited to detecting one single gesture at a time. Hence, using TUIOFX ensures that the same types of gestures are recognised in the same way on different platforms with TUIO support.

2.3 Providing Touch-Enabled Traditional Widgets

While direct-touch manipulation often resonates with an urge for a radical new interaction styles, we argue that many traditional GUI widgets are based on metaphors from physical entities, which now get back their tangibility. Controls like buttons, switches, or sliders, and form elements, like text fields, radio-buttons, or checkboxes, all exploit knowledge that we have from other domains. Most of these controls win back some of their original qualities when they are used through direct-touch rather than through a mouse pointer. These traditional widgets are widely accepted and appreciated beyond desktop computers. It is hardly surprising that many multi-touch interaction techniques used on mobile devices largely rely on these traditional desktop widgets. For instance, on touch-enabled devices buttons, check boxes and text entry widgets are intensively used with little customisation of their look-and-feel. In this way, desktop users get instantly familiar with the touch-enabled user interface, since they can draw on already learned interaction techniques. And finally, programmers are able to quickly support a number of standard interactions by relying on those standardised and reusable UI building blocks [38].

Our third requirement accordingly is, that a multi-touch toolkit should provide an extensive number of *touch-enabled traditional* GUI widgets. The availability of touch-enabled traditional widgets gives developers something to reuse and to extend when exploring novel multi-touch, multi-user widgets and interaction techniques that involve similar components to those used in already existing traditional widgets. Having a large number of widgets helps developers to reduce the implementation efforts needed to create user interfaces, since they can reuse—and when necessary extend—already available widgets instead of creating them from scratch. More importantly, developers do not have to deal with the complexity of specific widget implementation details just in order to simulate traditional interaction techniques.

By building TUIOFX on top of JavaFX, we are able to leverage from the many existing widgets from simple buttons and text fields (Fig. 2) to more complex UI elements like interactive charts, editable tables and tree views, colour picker, up to even a WYSIWYG-HTML5 editor. Additionally, many third-party widgets and UI controls can also be *reused*, providing great *extensibility*.

2.4 Optimising Widgets for Interactive Surfaces

However, just providing traditional widgets that are originally designed for desktop applications with keyboard and mouse input limits their usability on large *SIS*. Fortunately, the JavaFX platform is able to support multi-touch interaction on touch-enabled devices, by providing an event handling infrastructure for multi-touch inputs as well as augmenting the look-and-feel of some basic controls for multi-touch interaction on embedded systems. Thus, all JavaFX controls are already touch-enabled and are designed to response to touch inputs. Yet, adding multi-user multi-touch support still poses new challenges concerning the look-and-feel of these widgets:

- Due to the small size of desktop widgets, the large contact area of human fingers may occlude parts of a standard widget and thereby hiding visual information and making the selection of targets harder. A toolkit for large surfaces should provide at least a simple solution that improves the selection of target elements.
- Another challenge that needs to be addressed is the support for free orientation of widgets. Free orientation means *"the rotation of an item to any desired angle"* [23, p. 603]. This is an important requirement for the TUIOFX toolkit because people tend to occupy different positions around the horizontal display of tabletops and they will be viewing the same digital objects from different angles [22]. As a result, for some participants the digital object may be difficult to read and thus affects the comprehension of the content [22,47]. Besides the comprehension and the readability

problem, content orientation plays a mediating role in the coordination of actions and communication between individuals doing collaborative work [2,11,22,40,44].

To deal with these challenges, TUIOFX optimises the look-and-feel of JavaFX standard widgets for touch interaction on large touch screens by:

- Increasing the size of JavaFX widgets' target elements by default (developers can still change the size of any target element using the JavaFX API and CSS);
- Providing a dark CSS styling theme to prevent users from eye strains and eye fatigue when working in close range of the brightly lit displays; and
- Supporting free orientation of all standard widgets.

Through skinning JavaFX's standard widgets, we are able to provide a clean looking and functional out-of-the-box CSS styling theme for all widgets. Accordingly, developers can easily achieve a uniform design, without the need of designing an own theme.

2.5 Supporting Text Entry

Text entry is one of the most frequent tasks people do when working on desktop applications [14]. This is also true for several tabletop applications presented in the literature (e.g., [4,29]). Accordingly, means for concurrent text input for multiple users are needed. However, most hardware keyboards proposed for SIS so far [13,20,46] often come with specialised technological requirements that are not always available for off-the shelf hardware and applications. Hardware keyboards are also impractical for multi-user interaction due their lack of support for fast duplication and shareability [14]. Another possibility is to provide a soft keyboard [25] (virtual keyboard), as it has no special requirements to the underlying system and hardware. A soft keyboard can be easily removed from the screen, shared among participants, flexibly tailored to the application and allows for fast creation of multiple instances for multiple people [14]. Although, it might not be the best choice for extensive and accurate text entry, since it suffers from several drawbacks (e.g., lack of tactile feedback), we still regard it as a minimum requirement for a multi-touch toolkit that tends to support most general tabletop systems. Therefore, TUIOFX provides a soft keyboard implementation for text input supporting multiple users.

2.6 Supporting Multi-User Interaction

Finally, while multi-touch multi-user interaction on SIS increases the collaboration capabilities, they also introduce several issues when built on top of traditional single-user widgets [30,43,45]. We identified two important challenges and conflicts TUIOFX aims to manage:

2.6.1 Supporting Simultaneous User Actions

To support a range of multi-user interaction techniques, TUIOFX must handle simultaneous user actions correctly. However, JavaFX 's out-of-the-box multi-touch support does not allow multiple users to perform multiple gestures simultaneously. Designed as a single-user toolkit, JavaFX's event handling infrastructure is programmed to only listen to the entire stream of touch events provided by the underlying OS and thus can only interpret one gesture at a time. Hence, we cannot rely on the JavaFX's handling of gesture events when supporting simultaneous user gestures. Therefore, we developed a technique to split or group the single TUIO touch event stream (comparable to [17]) into multiple streams for each coherent gesture by a user, so that our gesture recognition engine can process each input stream independently and simultaneously in order to recognise multiple gestures at a time. As we want TUIOFX to support a variety of tabletop systems of which the majority comes without user identification capabilities, we provide a straightforward technique for grouping touch events without the need for user identification. This grouping technique works as follows: Each TUIO touch point is associated with the graphical object it currently hits. According to this principle, all touch points located on the same graphical object are grouped together and sent to gesture recognisers to be jointly interpreted into gestures. The underlying assumption is that a touch gesture is always directed to a specific graphical object and only one gesture performer manipulates a graphical object at a time.

2.6.2 Reducing Multi-User Interaction Conflicts

Allowing multiple people to interact simultaneously on a shared workspace can raise several conflicts. For example, one user's action may lead to global changes, which in turn, could impact or even interrupt the activities of other users. The prevalent interaction style for graphical user interfaces is, that a single user through a series of input events, invokes actions. This concept heavily builds on the *single-focus model*, that is: Single-user widgets rely on a *focus,* determining the single graphical object that can receive user input exclusively at a time. However, this focus model does not fit with most multi-touch and multi-user interaction concepts *"where there is no single focus model, and instead multiple touches may interact with multiple objects simultaneously"* [1, p. 255]. Additionally, under the presence of *anonymous touch input*, it is also unclear from which user, which touches originated. Therefore, most multi-user tabletop applications rely on user identification techniques [5,18,26,31,37,39] or territory-based approaches [28,40] to assign touches to users in order to reduce conflicts. However, setups with integrated user identification are rare, as they often require specialised hard- and software that so far mainly have been demonstrated in research prototypes [9]. While the territory-based approach can be achieved without user identification, fixed territories often limit the freedom of interaction, and might be inappropriate for many application types, as for example a multi-user graphical editor.

To overcome these limitations, we developed TUIOFX to support the design of applications that minimise conflicts also on hardware that only provide *anonymous touch input* and maximises the freedom of interaction. Therefore, we have conceived the *task-based focus model* [9], to resolve interaction conflicts caused by widgets that rely on the single user focus model when multiple users want to interact simultaneously.

3 RELATED WORK

Based on these requirements, we analysed existing toolkits from the research community as well as different commercial solutions (cf. Table 1).

DiamondSpin [42] offers a set of touch-enabled traditional widgets by reusing existing widgets of Java's Swing GUI Framework. DiamondSpin does not optimise the look-and-feel of most Swing widgets and thus they remain inappropriate for multi-touch multi-user interaction on tabletops. Further, it does not provide an event data structure for multi-touch input needed to notify the application about multi-touch gestures. Consequently, touch inputs must be translated to mouse events and restricting multi-touch interaction to multiple simultaneous mouse clicks and drag-and-drop operations. Built primarily to run on the DiamondTouch hardware, the toolkit leverages its user identification capabilities to support concurrent user inputs. This makes the DiamondSpin a special purpose toolkit with an excellent support for multiple users.

Microsoft Surface SDK [27]—a now discontinued product—is in the same way limited to one specific piece of hardware. It provides tools and comprehensive API for the development of multi-touch applications that run, however, only on *Surface* (later rebranded *PixelSense*) devices. The SDK optimises traditional Windows Presentation Foundation (WPF) UI elements for touch, thus leverages on a large community of developers and users. The SDK supports simultaneous user input by associating each input action (or gesture) with concrete UI elements. It comes with several tools to ease the development of *SIS* applications such as a simulator and the visual editor Microsoft Expression Blend. It allows developers to rotate all WPF widgets to any angle. However, users cannot enter text simultaneously using the default soft keyboard, since only one instance of the default soft keyboard can be created at a time. While the Surface SDK fulfils many of our requirements it only runs on Windows systems with native multi-touch support.

jQMultiTouch [34] is a JavaScript (JS) library for developing multi-touch web applications for touch devices from smartphones to interactive walls. Though, it has no built-in support for SIS systems that send touch inputs via TUIO. As a JS library, jQMultiTouch can be used with touch-enabled traditional HTML widgets that are well-supported on most popular touch screens. These standard HTML widgets, however, are not optimised for multi-user interactions or free orientation (e.g. rotating a <select> element does not rotate its options). Additionally, jQMultiTouch lacks support for concurrent text entry. Yet, by leveraging on web technologies like HTML and JS, with a large community of web developers and extensive documentation material, it has a very low entrance barrier.

Table 1. Overview of related work and our appraisal whether the requirements are: fully supported (●), partially supported (○), not supported (–), or unclear if supported (?).

	Diamond-Spin [42]	**Surface SDK [27]**	**MT4j [24]**	**TISCH [7]**	**Kivy [19]**	**jQMulti-Touch [34]**	**IntuiFace [15]**
Platform Independence	○	-	●	●	●	○	○
Touch-Enabled Widgets	●	●	○	○	●	●	○
Optimised Widgets for Tabletops	-	●	●	●	●	-	●
Simultaneous User Actions	○	●	●	●	●	●	●
Reducing Multi-User Conflicts	○	○	○	?	?	-	○
Free Orientation	○	●	●	●	●	○	?
Providing Possibility for Text Entry	○	●	●	●	●	-	●
Facilities for Reusability and Extensibility	●	●	○	-	○	○	○
Tool Support Visual Editor	-	●	-	-	-	●	●
Community and Documentation	-	●	-	-	○	●	○

IntuiFace [15] is a software for authoring multi-touch applications deployable on almost any touch devices (Windows, macOS, TUIO trackers, iOS, Android, Chrome OS, etc.). It offers many specialised touch-enabled widgets for exhibitions and presentations with a few standard widgets: label, text field, normal and toggle/radio buttons, but e.g. no checkbox or slider. It solely supports the visual development of user interfaces through its authoring tool IntuiFace Composer, thus limiting the capabilities for extending and creating custom widgets. Also, the ability to present multiple virtual keyboard instances for concurrent text entry is not officially supported.

MT4j [24] is Java-based toolkit with good cross-platform support. Similar to Surface SDK, it supports concurrent user actions by associating touches to their target UI component. However, the amount and functionality of standard widgets and components is very limited. Hence, developers must deal with re-implementing components such as basic UI elements for laying out components or advanced controls. Besides being used in a few research projects, MT4j has no active community of developers and only sparse documentation. With the last activity in 2011, the toolkit seems to be no longer maintained.

TISCH (formerly libTisch) [7] is a cross-platform framework for developing multi-touch user interfaces written in C++ (with wrappers for C#, Java, Python). The *TISCH* toolkit shares similar drawbacks as MT4j. It only provides a few traditional widgets or visual components, no extensive community of developers and accordingly scarce documentation and no tool support like visual editors. Further, *TISCH* requires low-level OpenGL drawing instructions for creating new widgets. Thus, programmers have to be familiar with the rather complex OpenGL API.

Kivy [19] (formerly *PyMT* [12]) is a python-based cross-platform toolkit. It provides a series of traditional widgets optimised for touch and it allows simultaneous text entry using the soft keyboard. Kivy comes with a well-documented, but custom, API with several examples and it has an active developer community. However, there is only limited documentation that explains how defining the appearance of existing widgets works. Also, it seems that for creating and styling new widgets low-level drawing functions are often required.

While there are some more toolkits for *SIS*—some with very specialised focus like e.g. the hardware abstraction layer *Squidy* [21]—many of them share at least one of the drawbacks.

4 TUIOFX TOOLKIT—CONCEPTS AND IMPLEMENTATION

In the following we provide technical details of the architecture and the implementation of TUIOFX [8]. We thereby explain how we achieved the previous established requirements and realised the design goals.

The TUIOFX toolkit is composed of two parts: *TUIOFX–Core* and *TUIOFX–WidgetToolkit*. While the first is concerned with transforming raw touch events into meaningful JavaFX Gesture- and Touch-Events, the second is concerned with providing multi-user, multi-touch widgets. Fig. 3 shows the main components of the toolkit in an overview of the architecture.

4.1 The TUIOFX–Core

TUIOFX–Core provides a set of gesture recognizers that translate raw TUIO messages into the corresponding standard JavaFX `TouchEvent` and `GestureEvent` types of JavaFX's event handling infrastructure. The TUIOFX-Core mainly consists of four sub-components: `TuioInputService`, `TouchHandler`, `GestureHandler` and a set of `GestureRecognizer`s.

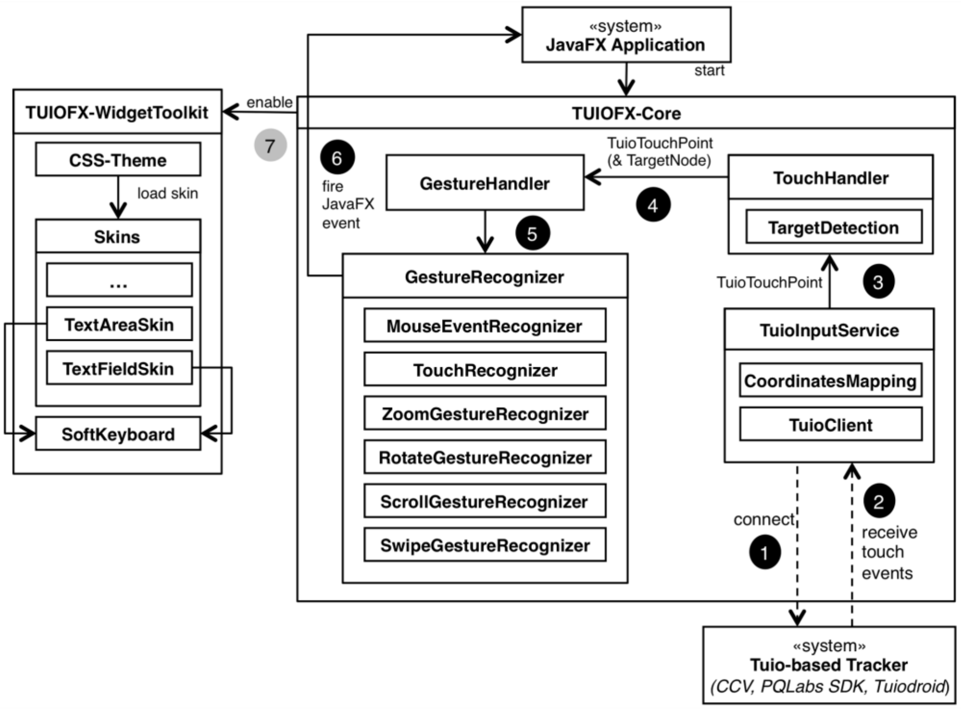


Fig. 3. Overview of the TUIOFX architecture, the main components and their interaction.

4.1.1 *TuioInputService*

When starting TUIOFX-Core, the `TuioInputService` establishes a connection to one or more TUIO trackers ((1) in Fig. 3) in order to listen to touch events sent from these trackers (2). The `TuioInputService` then translates the TUIO coordinates of the received touch event to the JavaFX coordinate space and creates a new object `TuioTouchPoint` that wraps the TUIO touch event object and is passed on to the `TouchHandler` (3).

4.1.2 *TouchHandler*

The `TouchHandler` sub-component is primarily responsible for the *target selection*, which is detecting the top-most JavaFX node (`TargetNode`) as the one the user wants to manipulate and thus should receive the touch event. The `TouchHandler` uses JavaFX's hit testing method `impl_pickNode()` of the scene's root `Node` to find the target node. The method delivers the front-most—in respect to the z-orientation—child node at the scene coordinates of the touch point. Special cases are controls that are based on a `PopUpWindow` (e.g., the list part of a drop-down menu), as the target nodes inside the `PopUpWindow` are not part of the root `Node` hierarchy. Therefore, the `TouchHandler` also iterates though eventual `PopUpWindows` at the touch location for finding the correct node. In each case, the information about the detected `TargetNode` is stored in the `TuioTouchPoint` object and handed to the *`GestureHandler`* (4).

4.1.3 *GestureHandler*

The first task of the *`GestureHandler`* is to group incoming `TuioTouchPoint` objects (i.e. touch events) according to their respective `TargetNode`. This is a necessary step to detect multi-touch gestures invoked on one single UI element [6]. Therefore, multiple logical touch event streams are formed and grouped for each `TargetNode`. Further, the *`GestureHandler`* initiates a set of `GestureRecognizers` whereby each gesture recognizer processes all touch-event streams associated with one `TargetNode` independently and in parallel.

4.1.4 *GestureRecognizer*

A `GestureRecognizer` (5) detects a gesture by analysing a stream of touch events (`TuioTouchPoint` objects) of a specific target node. When the user touches a node for the first time, *TUIOFX-Core* creates a set of gesture recognizers and attaches all of them to this node. That is, each gesture recognizer instance is associated with one target node, whereas a target node has multiple instances of different gesture recognizers. This allows users to perform combined gestures on a node: like dragging and rotating a node simultaneously.

Building on existing technology, TUIOFX toolkit is designed to reuse the JavaFX's multi-touch event handling infrastructure in order to send events about the recognised gesture to the associated target node. Therefore, *TUIOFX-Core* provides gesture recognizers to detect each `TouchEvent` and `GestureEvent` types supported by the JavaFX's event handling infrastructure (i.e. `TouchEvent`, `RotateEvent`, `ScrollEvent`, `SwipeEvent` and `Zoom Event`). Our custom gesture recogniser algorithms are informed by research concepts (e.g., RNT [23]) and real world implementations like Android's *GestureDetector class and robust* towards adding and removing fingers to a gesture. If an event is detected, a standard JavaFX event is fired on the respective JavaFX-Node, including a synthesised `MouseEvent`. Developers can simply use JavaFX's well-documented event handling API, to react to these events in their code. *TUIOFX-Core* can be used standalone with JavaFX and already allows a wide range of possible applications (e.g., multi-touch applications or games that rely on custom 2D or 3D graphics).

4.2 The TUIOFX–WidgetToolkit

To further support multi-user interaction on large touch screens, the *TUIOFX–WidgetToolkit* optimises the look-and-feel of the standard JavaFX widgets through extensive skinning. This includes very basic adjustments, like increasing the size of UI elements to improve the target selection to complex modifications, like profoundly changing the behaviour of some widgets to support multi-user interaction. To achieve this, *TUIOFX–WidgetToolkit* provides a CSS theme that overrides the JavaFX default CSS theme to adapt the appearance and behaviour of built-in controls with custom `Skin` implementations.

4.2.1 Skinning Widgets in JavaFX

JavaFX offers a great freedom in optimising the look-and-feel of controls (widgets) either via JavaFX CSS properties or by implementing custom `Skin` classes. A `Skin` class defines the visual representation and behaviour of a `Control`. While the basic styling of many aspects of a standard JavaFX `Control` can be adapted by pure CSS properties (comparable to CSS usage with HTML), by extending its `Skin` class, also its behaviour can be profoundly changed. TUIOFX heavenly relies on this skinning mechanism to overwrite all widgets.

For example, to adapt the appearance of JavaFX controls (e.g., dark colour scheme, size of fonts) in the *TUIOFX–WidgetToolkit*, we mainly use JavaFX CSS together with a few custom `Skin` classes for very specific visual styling purposes (e.g., adding the light bar at the bottom of the `ToggleButton` control (cf. On/Off toggle buttons in Fig. 2). Accordingly, the majority of the custom `Skin` classes in TUIOFX are for enhancing and tailoring the behaviour of JavaFX controls in order to support multi-user interaction and multi-touch.

```
.check-box{
   - fx.skin:"org.tuiofx.widgets.skin.MTCheckBoxSkin";
}
```

Listing 1. Snippet of the TUIOFX CSS theme that registers a custom Skin class "MTCheckBoxSkin" for the Checkbox control via the control's default selector ".check-box".

The *TUIOFX-WidgetToolkit* simply references all its custom `Skin` classes inside a CSS file (compare Listing 1). This way we ensure that creating new instances of these optimised controls work the same way as the normal JavaFX controls—that is, using exactly the same `Control` class regardless of the use of the *TUIOFX-WidgetToolkit*. For developers wanting to create a new `Checkbox` control, it is not necessary to learn the API and name of a new control (e.g., `MTCheckbox()`), but they can simply use the standard JavaFX `Checkbox()`, that will automatically load the needed adapted behaviour and styling of TUIOFX's `MTCheckboxSkin`.

In terms of visual styling, TUIOFX replaces JavaFX's default *Caspian* theme with a dark colour scheme to provide better readability and prevent eyestrain when working up close with brightly lit interactive surfaces. The font-size is increased to 16px for readability as are the size of the controls, for common tabletop sizes in the range of 40 to 50 inch with Full HD resolutions and average human finger size. By overwriting a few variables in the CSS-file, of course adequate settings for other configurations can be easily achieved.

4.2.2 Reducing Multi-User Interaction Conflicts

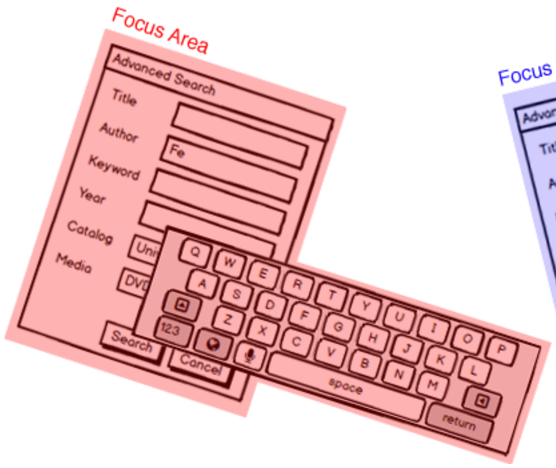
As already mentioned, to reduce multi-user interaction conflicts caused by the single-focus model, we have developed the concept of *task-based focus* [9]. The *task-based focus model* support concurrent user inputs without relying on user identification capabilities of the hardware. According to the single focus model anytime a new touch occurs on the interaction surface the graphical object under this new touch point will gain the focus, while the prior object that was in focus will lose it (i.e. a *blur* event). In JavaFX, the default behaviour of text entry controls and `PopupWindow`-based menus strongly depends on the single focus model, which makes it impossible for multiple users to interact simultaneously using these controls. For example, a `TextField` control requires to be first selected to exclusively gain the focus in order to receive and display key input events from the keyboard. Furthermore, without setting the focus, visual elements like the text cursor—used to provide feedback to the user—are not displayed. Consequently, multiple users cannot write simultaneously into different text input controls, at least not without battling for the focus or falling back into turn taking strategies. The same holds for JavaFX `PopupWindow`-based menu controls (e.g., ComboBox, ChoiceMenu, ContextMenu). By default, these PopupWindow-based menus disappear immediately after they lose the focus (i.e. as soon as a touch occurs outside the PopupWindow). With multiple users interacting on a tabletop, this may happen even before the user was able to make any selection.

To overcome these limitations we introduce the concept of a *focus area* to resolve conflicts caused by the *single-user focus* model. A *focus area* is a specific area of a user interface that contains one or more widgets where only one widget can be selected to gain the focus at a time. The selected widget remains in focus until any other user event occurs within the same focus area. A focus area is determined by simply defining a graphical object (Node) as the starting node. The focus area then extends from the starting node to all of its child nodes. Fig. 4 illustrates the concept of focus areas. As the root node of the Advanced Search dialogue (a simple Pane) was defined as a focus area, each child node (like the "Author" text field and the "Media" drop-down menu) shares the same focus. If a second instance of this dialogue is opened, it also holds its own focus. While every touch event outside of the red (respective blue area) does not interfere with the controls in the focus area, clicks inside the focus area will lead to a focus behaviour as it is expected from single user applications. As one can see, a node can be assigned to a focus area either explicitly by defining the object itself as starting node (the dialogue pane) or implicitly when being a nested node of a starting node (the text fields and drop down menus). More details can be found in [9].

Using this task-based focus model, one user can write inside a text field or interact with the PopupWindow while other users are simultaneously producing touch inputs outside the focus area. Thus, the PopupWindow or the text field only gets de-activated once the user selects another object within the same focus area. In general, each touch input within the same focus area is interpreted as a real focus shift and therefore de-activates the currently selected PopupWindow or text input control.

We call this conflict reduction strategy, *task-based focus*. It can be used to employ a task centric focus model by the developer, by grouping widgets that form a common task that is usually performed by one individual user.

To be able to use the *task-based focus* (focus area) with PopupWindow-based controls (ComboBox, ChoiceBox), TUIOFX-WidgetToolkit extends the Skin implementation of these PopupWindow and text field controls to behave according to the task-based focus model.



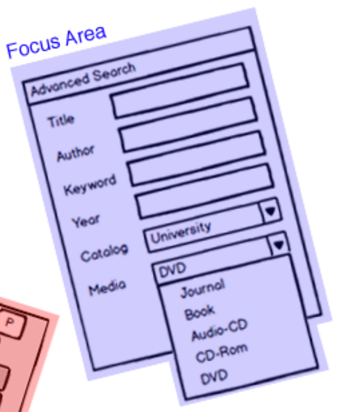


Fig. 4. Two instances of the same dialogue, each defining its own focus area (red left and blue right) respectively ignoring inputs outside these areas. That is, each focus area behaves internally according to the *single focus model*. For example, opening a drop-down list in the red area will remove the focus from the text field and accordingly hide the keyboard, but will not affect the opened drop-down list in the blue area.

That is, the PopupWindow should only get closed when a touch event occurs within the focus area associated to the PopupWindow. To detect whether a touch event occurs within the focus area or not the TUIOFX-WidgetToolkit adds a JavaFX event filter for each PopupWindow-based control. This event filter listens to all MOUSE_PRESSED events routed through the starting node of the PopupWindow and whenever a MOUSE_PRESSED occurs the filter close the PopupWindow.

Finally, TUIOFX-WidgetToolkit also changes the behaviour implementation of both JavaFX text input controls (TextField and TextArea) so that they can rely on the task-based focus model in order to support parallel writing. This includes extending the TextFieldSkin and TextAreadSkin to receive user text inputs from the attached soft keyboard even though they lose the focus as well as to only get deactivated depending on how the focus area of the particular text input control has been defined. Of course, this behaviour is also inherited by all other components that internally use TextField and TextArea, as for example editable cells in JavaFX's TableView.

4.2.3 *Supporting Text Entry*

While JavaFX comes with a soft keyboard implementation, this virtual keyboard is clearly aimed for single-user small display devices, like smart phones. It is one single keyboard, attached to the lower screen border and taking the full width of the screen. Thus, it is clearly not applicable for SIS. Therefore *TUIOFX-WidgetToolkit* provides its own soft keyboard implementation, which has been adopted from the fx-experience keyboard[1]. The soft keyboard comes with a QWERTY layout, but is extendable via XML to other layouts. By default, every text input control (i.e. TextArea and TextField) has an individual keyboard instance attached, so that touching a text input control opens the respective soft keyboard. However, by defining focus areas the developer easily can overwrite this standard behaviour, as within one focus area one keyboard instance is shared among multiple text-input controls.

4.2.4 *Providing Free Orientation*

In general, applying a rotation transformation to a JavaFX controls can change its orientation. Thus, in combination with an event handler for the RotateEvent, a node can be reoriented by users performing a rotate gesture. This allows participants around an interactive tabletop to orientate widgets towards them. In most cases, performing a rotation transformation on a control also rotates all graphical elements the control is composed of. For example, when rotating a Button control also the Text Node inside the Button gets rotated. However, this is not the case for JavaFX controls that are composed of a ContextMenu control (e.g., ComboBox, ChoiceBox). Therefore, the TUIOFX-WidgetToolkit provides a solution that rotates and repositions the ContextMenu to always match the rotation applied to the parent control. This is done via our custom Skin implementation of the ContextMenu. Further, also JavaFX ScrollEvents are fixed to the normal screen orientation. Therefore, for some controls that allow scrolling of their content, the ScrollEvent has to be transformed by taking into account the rotation of the control itself. This also happens inside some Skin implementations. This way, TUIOFX supports free orientation of all JavaFX controls.

5 UTILISING TUIOFX

In the following we highlight how easy it is to utilise TUIOFX for developers. The TUIOFX toolkit is distributed in form of two separate libraries: TUIOFX–Core and TUIOFX–WidgetToolkit. To migrate any existing JavaFX project into a multi-user, multi-touch TUIOFX project, both libraries can simply be added as JAR file to the project.

TUIOFX applications can be deployed on a variety of hardware and software combinations as long as they support the TUIO protocol and Java 8. We have primarily developed TUIOFX on our custom-made 42"

[1] http://github.com/comtel2000/fx-experience

tabletop built using a TUIO-enabled PQ Labs Multi-Touch Overlay as multi-touch sensing input device, but also successfully tested a variety of other setups and form factors.

5.1 The API Design of TUIOFX

One of TUIOFX strength is its minimal API and resulting high ease of learning. As TUIOFX is founding on JavaFX's flexible capabilities to overwrite the default look-and-feel of existing `Controls`, we were able to achieve the complex adaptations of UI elements without any changes for developers on how to use these components in their code. Accordingly, we were able to keep the API of TUIOFX very lightweight, allowing the toolkit to be minimally invasive to existing code. The `TuioFX` class of TUIOFX–Core with only eight methods is the main point of interaction with the API.

```
1 public class TuioFXApplication extends Application {
2
3     public static void main(String[] args) {
4         TuioFX.enableJavaFXTouchProperties();
5         launch(args);
6     }
7
8     public void start(Stage stage) throws Exception {
9         Pane root = new StackPane();
10        Scene scene = new Scene(root);
11        stage.setScene(scene);
12
13        // start TuioFX
14        TuioFX tuioFX = new TuioFX(stage, Configuration.debug());
15        tuioFX.enableMTWidgets(true);
16        tuioFX.start();
17
18        // display application window
19        stage.show();
20    }
21 }
```

Listing 2. Integration of TUIOFX-Core and TUIOFX–WidgetToolkit to a JavaFX Application.

Integrating TUIOFX in a new or existing JavaFX application requires only four lines of code: First, the TUIOFX-Core needs to be initialised (see line 4 in Listing 2) before normally launching the JavaFX application in line 5. This enables several JavaFX system properties, which in turn activate features of the Java virtual machine reserved for touch devices. By adding the lines 14 and 16 `TuioFX tuioFX = new TuioFX(stage, Configuration. debug())` and `tuioFX.start()`to the code, any existing JavaFX application immediately is able to react to TUIO events from TUIO trackers over the standard port 3333. The line 15 `tuioFX.enableMTWidgets(true)` activates the TUIOFX-WidgetToolkit. All widgets receive the TUIOFX look-and-feel including widget functionality like text-input via the soft-keyboard, task-based focus, etc. With these four lines, most of TUIOFX's functionality for developing multi-user, multi-touch applications can be leveraged, in any JavaFX application. Beyond these lines, only a few more classes and custom properties are needed for further adapting and tuning the behaviour of TUIOFX. Two classes (`Configuration` and `Configuration.Builder`) allow to further tailor TUIOFX to specific hardware. While TUIOFX provides a few default configurations, the `Configuration` parameters allow to fine-tune aspects of the gesture recognisers, to configure a different TUIO port, or to toggle the debug output, etc. And finally a class `TuioTangible` and the respective `TangibleListener` provide possibilities to deal with

Tangibles. Together they allow the handling of relayed *TUIO Object* events in a way that resembles the general JavaFX event-handling infrastructure.

5.2 TUIOFX Custom Properties

Finally, TUIOFX behaviour can be further adapted by adding TUIOFX's custom properties directly to a JavaFX `Node`. A JavaFX `Node` property is set by calling its `getProperties().put("<propertyName>", "<value>")` method. Table 2 shows the two available TUIOFX properties. The property `focusArea` is used to define a node as focus area, as discussed in detail previously. The property `isTUIOTouchTransparent` is sometimes helpful to make transparent containers, only used for layouting controls (e.g. a `GridPane`), invisible for the gesture recognisers.

Table 2. TUIOFX Features activated using `Node` properties.

Property name	**Property description**
`focusArea`	If true, this `Node` is defined as a starting node of a focus area
`isTUIOTouch Transparent`	If true, this `Node` is ignored by the `TouchHandler` in the *target selection*

5.3 Building Applications With TUIOFX

In this section, we briefly highlight applications that have been built with TUIOFX, to assess its actual benefits. This includes a few small applications (1-4), which highlight different aspects of the toolkit, as well as a few bigger applications, that were built in student projects and theses (5 & 6), to informally evaluate the learnability as well as for judging the toolkits general robustness and stability.

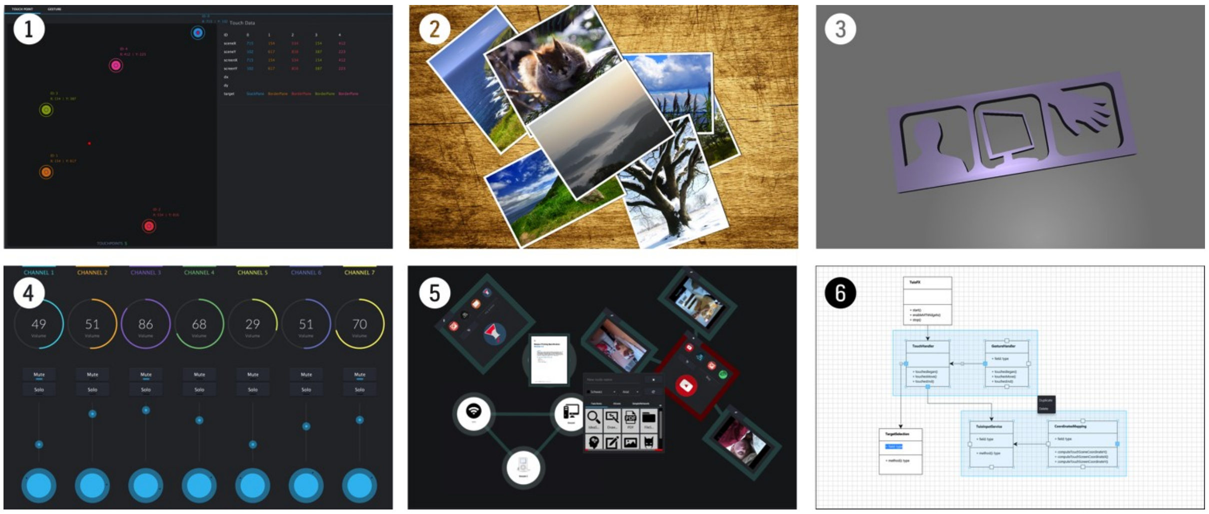


Fig. 5. Selection of TUIOFX applications, including Touch and Gesture Visualiser (1), Picture Browser (2), 3D Model Viewer (3), Mixing Console (4), IdeateTable Creativity Tool (5) and Collaborative UML-Editor (6).

In Fig. 5 we present a few of these applications as screenshots. The first example (1) is TUIOFX's Touch and Gesture Visualiser, a utility application that allows checking the basic functioning of TUIOFX-Core on new hardware and allows fine-tuning of Configuration parameters. The Picture Browser (2) rebuilds a common, classic multi-touch demo in ca. 100 lines of code (loc) and took ca. 20 minutes to implement and style via CSS. A version that allows playing several video files in parallel also exists, to evaluate the performance of the TUIOFX's touch handling during high CPU-load with ca. 10 loc more. Much similar, the 3D Browser (3) for manipulating the view of different 3D models with gestures and utilising JavaFX 3D Graphics (>500 loc) was used to successfully test the perceived latency of TUIOFX-Core. The Mixing Console UI mockup (4) was implemented for checking the out-of-the-box compatibility of the TUIOFX–WidgetToolkit with third party JavaFX UI libraries (i.e. the gauges and rotary controllers) in less than 250 loc. The IdeateTable ideation and mind mapping tool (5) and the Collaborative UML-Editor for class diagrams (6) are two more extensive (both 50+ classes) examples as results from a student project and a thesis. Both make heavy use of the Widget-Toolkit (although the components were visually restyled for the UML-Editor) and explore the applicability of *task-based focus* and *adhoc-focus-territories* [9]. For all, the TUIOFX specific code—i.e. the developer writing methods and classes from TUIOFX's API—never exceeds 20 lines of code.

Accordingly, from our observations and the feedback we received within the student projects, we found that the utilisation of TUIOFX only accounted for a negligible number of the issues encountered by students in the project. Although the students experience with Java and JavaFX was quite diverse from only introductory courses to several years of active programming experience, not one single student had problems with the usage of TUIOFX per se. As a further result, we were able to identify and remove a few bugs and glitches that occurred through the sometimes unconventional, and therefore unforeseeable, programming solutions of the students in the first projects.

6 CONCLUSION & FUTURE WORK

In this paper we presented the TUIOFX toolkit, that aims to support developers of multi-user, multi-touch applications with an easy to learn, stable, and cross-platform toolkit. By only working beneath the surface of JavaFX, we are able to present a toolkit that affords almost zero adoption time for experienced Java programmers and is highly flexible. With our underlying concept of *task-based focus*, we also equipped the toolkit with a novel interaction paradigm—if this paradigm proves successful for users needs to be justified in future evaluations. So far, just from observing users interacting with demo applications that rely on task-based focus, we found that they interacted without even noticing the underlying mechanisms—which was the aim of the concept. However, in future work we aim to systematically evaluate *task-based focus* in a user study.

By making TUIOFX available to the community[1], we continuously gather further insights from programmers, researchers, and designers with a plethora of hardware setups and different application scenarios. This allows further improving the stability and robustness of the toolkit. As currently Java is transitioning from version 8 to 9, some effort also is going into observing and maintaining the compatibility with future versions of JavaFX.

ACKNOWLEDGMENTS

We thank the members of the Cooperative Media Lab of the University Bamberg, our students and everybody testing and providing feedback on TUIOFX. We also thank the reviewers for their valuable comments.

REFERENCES

[1] Hrvoje Benko, Shahram Izadi, Andrew D Wilson, Xiang Cao, Dan Rosenfeld and Ken Hinckley. Design and Evaluation of Interaction Models for Multi-touch Mice. In Proceedings of Graphics Interface 2010 - GI 2010 (May 31 - Jun 2, Ottawa, Canada). Canadian Information Processing Society, Toronto, Canada, 2010. pp. 253-260.

[2] Christophe Bortolaso, T. Nicholas C. Graham, Stacey D. Scott, Doug Brown and Liam Porter. Design of a Multi-Touch Tabletop for Simulation-Based Training. In Proceedings of International Command and Control Research and Technology Symposium - ICCRTS 2014 (Jun 16-19, Alexandria, VA, USA). International Command and Control Research Institute, Washington, DC, USA, 2014.

[1] http://www.tuiofx.org

[3] Tom Brinck and Louis M. Gomez. A Collaborative Medium for the Support of Conversational Props. In Proceedings of the 1992 ACM Conference on Computer-Supported Cooperative Work - CSCW 1992 (Oct. 31 - Nov. 4, Toronto, Ontario, Canada). ACM Press, New York, NY, USA 1992. pp. 171–178.

[4] Andrew Clayphan, Anthony Collins, Christopher Ackad, Bob Kummerfeld and Judy Kay. Firestorm: A Brainstorming Application for Collaborative Group Work at Tabletops. In Proceedings of the 2011 International Conference on Interactive Tabletops & Surfaces - ITS 2011 (Oct. 30 - Nov. 2, Kobe, Japan). ACM Press, New York, NY, USA, 2011. pp. 162–171.

[5] Paul Dietz and Darren Leigh. Diamondtouch: A Multi-User Touch Technology. In Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology - UIST 2001 (Nov. 11-14, Orlando, FL, USA). ACM Press, New York, NY, USA, 2001. pp. 219-226.

[6] Florian Echtler and Andreas Butz. GISpL: Gestures Made Easy. In Proceedings of the Sixth International Conference on Tangible Embedded Interaction - TEI 2012 (Feb.19-22, Kingston, Ontario, Canada). ACM Press, New York, NY, USA, 2012. pp. 233-240.

[7] Florian Echtler and Gudrun Klinker. A Multitouch Software Architecture. In Proceedings of the 5th Nordic Conference on Human-Computer Interaction - NordiCHI 2008 (Lund, Sweden). ACM Press, New York, NY, USA, 2008. pp. 463-466.

[8] Mirko Fetter and David Bimamisa. TUIOFX—Toolkit Support for the Development of JavaFX Applications for Interactive Tabletops. In Proceedings of the 15th IFIP TC.13 International Conference on Human-Computer Interaction - INTERACT 2015 (Sept. 14-18, Bamberg, Germany). Springer, Heidelberg, Germany, 2015. pp. 476-479.

[9] Mirko Fetter, David Bimamisa and Tom Gross. Task-Based Focus and AdHoc-Focus-Territory—Novel Concepts for Shared Interactive Surfaces. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2016 (May 7-12, San Jose, CA, USA). ACM Press, New York, NY, USA, 2016. pp. 1193-1200.

[10] Mirko Fetter, Tom Gross and Maxi Hucke. Supporting Social Protocols in Tabletop Interaction through Visual Cues. In Proceedings of the Thirteenth IFIP TC.13 International Conference on Human-Computer Interaction - INTERACT 2011 (Sept. 5-9, Lisbon, Portugal). Springer, Heidelberg, 2011. pp. 435-442.

[11] Mirko Fetter, Sascha Leicht, David Bimamisa and Tom Gross. Structuring Interaction in Group Decision Making on Tabletops. In Mensch & Computer - 13. Fachuebergreifende Konferenz fuer interaktive und kooperative Medien - M&C 2013 (Sept. 8-11, Bremen, Germany). Oldenbourg, Munich, Germany, 2013. pp. 277-280.

[12] Thomas E. Hansen, Juan Pablo Hourcade, Mathieu Virbel, Sharath Patali and Tiago Serra. PyMT: A post-WIMP Multi-touch User Interface Toolkit. In Proceedings of the International Conference on Interactive Tabletops and Surfaces - ITS 2009 (Nov. 23-25, Banff, Alberta, Canada). ACM Press, New York, NY, USA, 2009. pp. 17-24.

[13] Björn Hartmann, Meredith Ringel Morris, Hrvoje Benko and Andrew D. Wilson. Augmenting Interactive Tables with Mice & Keyboards. In Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology - UIST 2009 (Oct. 4-7, Victoria, BC, Canada). ACM Press, New York, NY, USA, 2009. pp. 149–152.

[14] Uta Hinrichs, Mark Hancock, Sheelagh Carpendale and Christopher Collins. Examination of Text-Entry Methods for Tabletop Displays. In Proceedings of the Second Annual IEEE International Workshop on Horizontal Interactive Human-Computer Systems - TABLETOP 2007 (Oct. 10-12, Newport, RI, USA). IEEE Computer Society, Los Alamitos, CA, USA, 2007. pp. 105–112.

[15] IntuiLab SA. IntuiFace. https://http://www.intuilab.com/, 2017. (Last accessed: 29/03//2017).

[16] Martin Kaltenbrunner, T Bovermann, Ross Bencina and E Costanza. TUIO - A Protocol for Table-Top Tangible User Interfaces. In Proceedings of the 6th International Workshop on Gesture in Human-Computer Interaction and Simulation - GW 2005 (May 18-20, Ile de Berder, France). 2005.

[17] Kenrick Kin, Björn Hartmann, Tony DeRose and Maneesh Agrawala. Proton++: A Customizable Declarative Multitouch Framework. In Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology - UIST 2012 (Oct. 7-10, Cambridge, MA, USA). ACM Press, New York, NY, USA, 2012. pp. 477–486.

[18] Ulrike Kister, Patrick Reipschlaeger, Fabrice Matulic and Raimund Dachselt. BodyLenses: Embodied Magic Lenses and Personal Territories for Wall Displays. In Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces - ITS 2015 (Nov. 15-18, Madeira, Portugal). ACM Press, New York, NY, USA, 2015. pp. 117-126.

[19] Kivy. Kivy: Cross-Platform Python Framework for NUI Development. http://kivy.org/, 2017. (Last accessed: 29/03//2017).

[20] Sungahn Ko, KyungTae Kim, Tejas Kulkarni and Niklas Elmqvist. Applying Mobile Device Soft Keyboards to Collaborative Multitouch Tabletop Displays. In Proceedings of the 2011 International Conference on Interactive Tabletops & Surfaces - ITS 2011 (Oct. 30 - Nov. 2, Kobe, Japan). ACM Press, New York, NY, USA, 2011. pp. 130–139.

[21] Werner A Koenig, Roman Raedle and Harald Reiterer. Squidy: A Zoomable Design Environment for Natural User Interfaces. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2009 (Apr. 4-9, Boston, MA, USA). ACM Press, New York, NY, USA, 2009. pp. 4561-4566.

[22] Russell Kruger, Sheelagh Carpendale, Stacey D. Scott and Saul Greenberg. Roles of Orientation in Tabletop Collaboration: Comprehension, Coordination and Communication. Computer Supported Cooperative Work (CSCW) 13, 5-6 (2004). pp. 501–537.

[23] Russell Kruger, Sheelagh Carpendale, Stacey D. Scott and Anthony Tang. Fluid Integration of Rotation and Translation. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 2005 (Apr. 2-7, Portland, USA). ACM Press, New York, NY, USA, 2005. pp. 601-610.

[24] Uwe Laufs, Christopher Ruff and Jan Zibuschka. MT4j – A Cross-platform Multi-touch Development Framework. CoRR - http://arxiv.org/abs/1012.0467, 2010. (Last accessed: 12/01/2017).

[25] I. Scott MacKenzie, Shawn X. Zhang and R. William Soukoreff. Text Entry Using Soft Keyboards. Behaviour & Information Technology 18, 4 (1999). pp. 235–244.

[26] Nicolai Marquardt, Johannes Kiemer, David Ledo, Sebastian Boring and Saul Greenberg. Designing User-, Hand-, and Handpart-aware Tabletop Interactions with the TouchID Toolkit. In Proceedings of the 2011 International Conference on Interactive Tabletops & Surfaces - ITS 2011 (Oct. 30 - Nov. 2, Kobe, Japan). ACM Press, New York, NY, USA, 2011. pp. 21-30.

[27] Microsoft Corporation. The Microsoft Surface 2.0 SDK. https://msdn.microsoft.com/en-us/library/ee692162(v=surface.10).aspx, 2014. (Last accessed: 29/03//2017).

[28] Meredith Ringel Morris, Anthony Cassanego, Andreas Paepcke, Terry Winograd, Anne Marie Piper and Anqi Huang. Mediating Group Dynamics through Tabletop Interface Design. IEEE Computer Graphics and Applications 26, 5 (2006). pp. 65–73.

[29] Meredith Ringel Morris, Jarrod Lombardo and Daniel Wigdor. WeSearch: Supporting Collaborative Search and Sensemaking on a Tabletop Display. In Proceedings of the ACM 2010 Conference on Computer-Supported Cooperative Work - CSCW 2010 (Feb. 6-10, Savannah, GA, USA). ACM Press, New York, NY, USA, 2010. pp. 401–410.

[30] Meredith Ringel Morris, Kathy Ryall, Chia Shen, Clifton Forlines and Frederic Vernier. Beyond "Social Protocols": Multi-User Coordination Policies for Co-located Groupware. In Proceedings of the 2004 ACM Conference on Computer-Supported Cooperative Work - CSCW 2004 (Nov. 6-10, Chicago, IL, USA). ACM Press, New York, NY, USA, 2004. pp. 262-265.

[31] Meredith Ringel Morris, Kathy Ryall, Chia Shen, Clifton Forlines and Frederic Vernier. Release, Relocate, Reorient, Resize. In Extended Abstracts of the Conference on Human Factors in Computing Systems - CHI 2004 (Apr. 24-29, Vienna, Austria). ACM Press, New York, NY, USA, 2004. pp. 1441-1444.

[32] MultiTouch Ltd. Cornerstone SDK. https://cornerstone.multitouch.fi/developer_guide, 2015. (Last accessed: 29/03//2017).

[33] Miguel A. Nacenta, David Pinelle, Dane Stuckel and Carl Gutwin. The Effects of Interaction Technique on Coordination in Tabletop Groupware. In Proceedings of Graphics Interface 2007 - GI 2007 (May 28 - 30, Montréal, Canada). Canadian Information Processing Society, Mississauga, Ontario, Canada, 2007. pp. 191–198.

[34] Michael Nebeling and Moira Norrie. jQMultiTouch: Lightweight Toolkit and Development Framework for Multi-touch/Multi-device Web Interfaces. In Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS 2012 (Jun. 25 - 28, Copenhagen, Denmark). ACM Press, New York, NY, USA, 2012. pp. 61-70.

[35] David Pinelle, Miguel Nacenta, Carl Gutwin and Tadeusz Stach. The Effects of Co-Present Embodiments on Awareness and Collaboration in Tabletop Groupware. In Proceedings of Graphics Interface 2008 - GI 2008 (May 28-30, Windsor, Ontario, Canada). Canadian Information Processing Society, Mississauga, Ontario, Canada, 2008. pp. 1–8.

[36] Mark Roseman and Saul Greenberg. Building Real Time Groupware with GroupKit, A Groupware Toolkit. ACM Transactions on Computer-Human Interaction (TOCHI) 3, 1 (March 1996). pp. 66-106.

[37] Volker Roth, Philipp Schmidt and Benjamin Gueldenring. The IR Ring: Authenticating Users' Touches on a Multi-touch Display. In Proceedings of the 23rd Annual ACM Symposium on User Interface Software and Technology - UIST 2010 (Oct. 3-6, New York, NY, USA). ACM Press, New York, NY, USA, 2010. pp. 259-262.

[38] Kathy Ryall, A. Esenther, C. Forlines, C. Shen, S. Shipman, Meredith Ringel Morris, K. Everitt and F. D. Vernier. Identity-Differentiating Widgets for Multiuser Interactive Surfaces. IEEE Computer Graphics and Applications 26, 5 (2006). pp. 56–64.

[39] Dominik Schmidt, Ming Ki Chong and Hans W Gellersen. IdLenses: Dynamic Personal Areas on Shared Surfaces. In Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces - ITS 2010 (Nov. 7-10, Saarbrücken, Germany). ACM Press, New York, NY, USA, 2010. pp. 131-134.

[40] Stacey D. Scott, M. Sheelagh T Carpendale and Kori M Inkpen. Territoriality in Collaborative Tabletop Workspaces. In Proceedings of the 2004 ACM Conference on Computer-Supported Cooperative Work - CSCW 2004 (Nov. 6-10, Chicago, IL, USA). ACM Press, New York, NY, USA, 2004. pp. 294-303.

[41] Stacey D. Scott, Karen D. Grant and Regan L. Mandryk. System Guidelines for Co-located, Collaborative Work on a Tabletop Display. In Proceedings of the 8th European Conference on Computer-Supported Cooperative Work - ECSCW 2003 (Sept. 14-18, Helsinki, Finland). Kluwer Academic Publishers, Netherlands, 2003. pp. 159-178.

[42] Chia Shen, Frédéric D. Vernier, Clifton Forlines and Meredith Ringel. DiamondSpin: An Extensible Toolkit for Around-the-Table Interaction. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 2004 (Apr. 24-29, Vienna, Austria). ACM Press, New York, NY, USA, 2004. pp. 167-174.

[43] Jason Stewart, Benjamin B. Bederson and Allison Druin. Single Display Groupware: A Model for Co-present Collaboration. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 1999 (May 15-20, Pittsburgh, PA, USA). ACM Press, New York, NY, USA, 1999. pp. 286-293.

[44] John C. Tang. Findings from Observational Studies of Collaborative Work. International Journal of Man-Machine Studies 34, 2 (1991). pp. 143–160.

[45] Edward Tse, Jonathan Histon, Stacey D. Scott and Saul Greenberg. Avoiding Interference: How People Use Spatial Separation and Partitioning in SDG Workspace. In Proceedings of the ACM 2004 Conference on Computer-Supported Cooperative Work - CSCW 2004 (Nov. 6-10, Chicago, IL, USA). ACM, New York, NY, USA, 2004. pp. 252-261.

[46] Malte Weiss, Julie Wagner, Yvonne Jansen, Roger Jennings, Ramsin Khoshabeh, James D. Hollan and Jan Borchers. SLAP Widgets: Bridging the Gap Between Virtual and Physical Controls on Tabletops. In Proceedings of the Conference on Human Factors in Computing Systems - CHI 2006 (Apr. 22-27, Montreal, Canada). ACM Press, New York, NY, USA, 2006. pp. 481–490.

[47] Daniel Wigdor and Ravin Balakrishnan. Empirical Investigation into the Effect of Orientation on Text Readability in Tabletop Displays. In Proceedings of the Ninth Conference on European Conference on Computer-Supported Cooperative Work - ECSCW 2005 (Paris, France). Springer-Verlag, Berlin/Heidelberg, Germany, 2005. pp. 205–224.