Sketchable Interaction: Drawing User Interfaces with Interactive Regions

Jürgen Hahn
University of Regensburg
Germany
juergen.hahn@ur.de

Raphael Wimmer
University of Regensburg
Germany
raphael.wimmer@ur.de

ABSTRACT

Sketchable Interaction (SI) describes a concept and environment where end-users create regions by drawing on a canvas. These regions apply *effects* to each other on *collision*. Attributes of regions, e.g. position, can be *linked* to each other so that they change together once modified, e.g. moved on the canvas. Within Sketchable Interaction, all entities - mouse pointer, desktop icons, or windows - are implemented as interactive regions. End-users customize this environment by drawing new regions that apply certain actions e.g. tagging files, deleting other regions or automating processes.

CCS CONCEPTS

• **Human-centered computing → Graphical user interfaces**; **Interaction design theory, concepts and paradigms**; **User interface programming**; • **Software and its engineering → Integrated and visual development environments**.

KEYWORDS

Sketchable Interaction; End-User Customization; Desktop Environment;

ACM Reference Format:
Jürgen Hahn and Raphael Wimmer. 2021. Sketchable Interaction: Drawing User Interfaces with Interactive Regions. In *CHI Conference on Human Factors in Computing Systems Extended Abstracts (CHI '21 Extended Abstracts), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3411763.3451567

1 INTRODUCTION

Over the past 75 years, computers have evolved from powerful calculators to even more powerful generic tools, communication media, and companions. From early on, researchers such as Seymour Papert, Douglas Engelbart, or Alan Kay [7] saw computers and applications running on them as malleable artifacts and tools for human expression. It was assumed that people would build and adapt their own digital tools in the future[3, 4]. However, such a future has not yet become present. Most users are stuck with *one-size-fits-all*, generic user interfaces for working on everyday tasks, defining workflows, or completing repetitive tasks. With *Sketchable Interaction* (SI), we propose a concept for a generic, customizable digital environment based on sketchable interactive regions.

Sketchable Interaction is a significant extension of the basic idea and interaction concept presented as *Sketchable Workspaces and Workflows* [10] by Wimmer and Hahn. This initial concept in turn was inspired by Isenberg et al.'s *Buffer Framework* [5, 6]. Also, earlier work on information substrates [2] and the **strates* series by Klokmose, Rädle, et al. such as Webstrates [8], Codestrates [9], or Vistrates [1] inspired the SI concept.

In this paper we present our reference implementation of this concept, the *Sketchable Interaction General Runtime* (SIGRun).

Within SIGRun, we implemented a flexible computer desktop that is an ecosystem of interactive regions. This ecosystem can be extended and customized to support a variety of workflows. We demonstrate its versatility via three different application examples - tagging/sorting, image editing, and automation.

2 SKETCHABLE INTERACTION

Sketchable Interaction (SI) is a generic user-interface concept where end-users create interactive sketches by drawing *interactive regions* (short: *regions*) within a canvas (*SI context*). The regions apply *effects* to other regions on collision, i.e. once they touch or overlap. Additionally, properties of a region can be linked to properties of other regions (Figure 1).

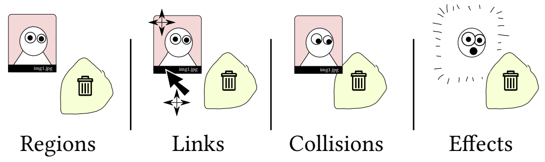


Figure 1: Sketchable Interaction encompasses four core concepts - regions, links, collisions, and effects.

2.1 Interactive Regions

Interactive regions are areas within an SI context with an arbitrary shape and position. Regions are created either automatically by SIGRun or by drawing an outline with the mouse or any other input device. In an SI context, everything is represented as a region. This includes e.g., the drawing surface (canvas), the mouse cursor, digital files, or windows of external applications. All interaction revolves around creating, moving, colliding, and linking such regions.

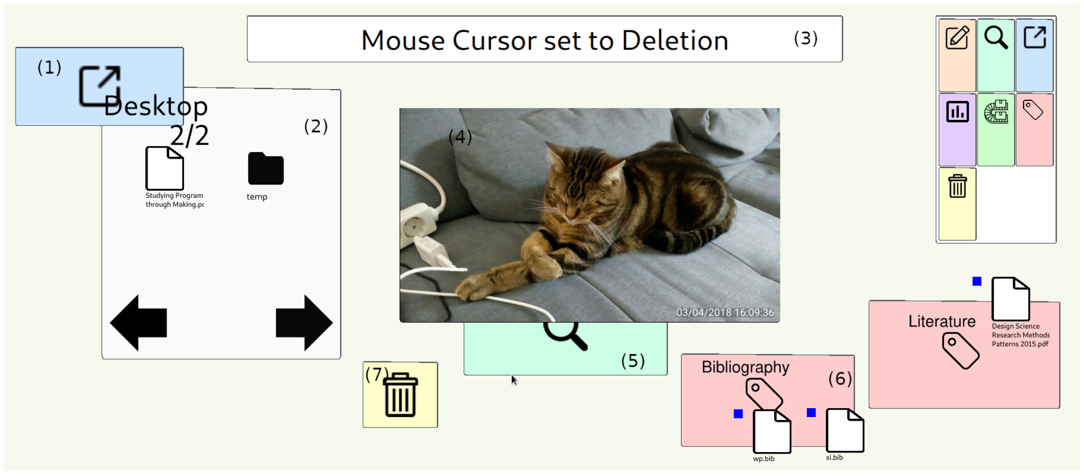

Mouse Cursor set to Deletion
(3)
(1)
Desktop
2/2
(2)
Studying Program through Making.p
temp
(4)
03/04/2018 16:09:36
(5)
(7)
Bibliography
(6)
wp.bib
si.bib
Literature
Design Science Research Methods Patterns 2015.pdf

Figure 2: A screenshot of our prototype - a basic *SI context* that allows for previewing, tagging, sorting, and deleting files. All elements in the screenshot - including the mouse cursor - are interactive regions that apply effects to other regions on collision. (1) The blue *Open* region opens folders and files whose icon collides with them. (2) A folder named "Desktop" has already been dragged onto it and is shown as a *Folder* region. For each file in the folder, a new interactive region is spawned representing the file. (3) A *Notification* region displays arbitrary status information. (4) One of the image files in the folder has been dragged onto a *Preview* region which instructs the file's region to display the contents. (5) A *Palette* region contains *Selector* regions which represent effects that are transferred to the mouse cursor when clicked. The user then can draw new regions that have the selected effect. (6) *Tag* regions contain editable text labels. These labels and a visual indicator (here: blue square) are applied to any files (or other suitable regions) that collide with the *Tag* region. (7) A *Delete* region removes regions that collide with it from the SI context. As with all other regions, a collision occurs if another region is dragged onto the Delete region or if the Delete region is used like an eraser and dragged across other regions.

2.2 Collision and Effects

Regions interact with each other on collision by applying effects to each other. These effects can modify properties of the other region (e.g., its size, color, or associated metadata). Effects emit *collision events* once two or more regions start overlapping (`on_enter`), continue to do so (`on_continuous`), or stop colliding for the first time (`on_leave`). Regions can register *capabilities* to accept only certain types of events. For example, unlike most other regions, the canvas and the mouse cursor should not accept *Delete* events. Effects are implemented as plugins which enables developers or power-users to extend the amount of available effects to draw as interactive regions.

2.3 Linking

Each region has properties, such as position, outline, color, or label. Regions of a certain type have further properties. For example, a region showing an image might have properties such as the path to the image file or the raw data. Properties can be linked so that e.g. a change in position of one region causes the same movement of another region. The effects of the linked regions have to handle translation between data types themselves, e.g., from a position to a color. These links are typically established when an effect is applied to a colliding region. Links use capabilities to determine which property of a region is linked to which property of another region. By creating and immediately removing a link, a region can set a property of another region.

3 SKETCHABLE INTERACTION AS A DESKTOP ENVIRONMENT

SI as a desktop environment (Figure 2) is a two-dimensional canvas containing file icons and other graphical objects that can be moved around - similar to a traditional computer desktop. It is implemented as an SI context within SIGRun and provides an ecosystem of interactive regions for common tasks that allows end-user users to customize the environment. Building on the SI concept offers three general advantages: (1) Sketching and direct manipulation are accessible and easy-to-learn methods for creating and modifying custom user interfaces. (2) As customization is embedded directly in the user interface, users can quickly change the UI in order to adapt to a changed workflow without having to switch to an IDE or separate editor. (3) Linking and collision-triggered events make it easy for intermediate developers to extend the ecosystem and provide a framework that inherently facilitates interoperability between effects due to implementing effects as plugins. SI extends the traditional desktop with new interaction techniques which we show in the following.

3.1 Sketching

The most important extension is to allow *Sketching* of user interface elements. This interaction technique is not part of SIGRun but implemented via plugins. Each SI context contains a default *Canvas* region which has the capability to receive `sketch` events and a *MouseCursor* region which emits `sketch` events when the left mouse button is pressed. Therefore, dragging the mouse cursor across the canvas triggers the `on_continuous()` handler of the canvas which records the path that the cursor takes. When releasing the mouse button, the cursor un-registers the `sketch` capability, which activates the `on_leave()` handler of the canvas. The canvas then creates a new region with the drawn shape.

3.2 Drag and Drop

Users can drag regions around using the mouse. This behavior is also implemented entirely using the core SI concepts described above. The mouse cursor is just another region whose position follows mouse movement.[1] Whenever the mouse cursor and another region collide, the cursor region checks whether the right mouse button is pressed, and then links the "position" attribute of the mouse cursor to the "position" attribute of the other region - provided the other region accepts `position` events.[2] This newly established link then triggers on every movement of the cursor region, which emits an update event containing absolute position and relative movement. Releasing the mouse button removes the link.

4 APPLICATION EXAMPLES

4.1 Sorting and Tagging

One of the most basic use cases for a desktop is organizing files. In order to support this use case within SI, we implemented three basic tools (i.e. SI effects) to support sorting of documents: *Tag*, *Delete*, and *Preview* (Figure 3) . Dragging a text file icon onto a *Preview* region instructs the text region to display a larger preview of its contents. *Tag* regions apply a custom text label and add a visual marker to a file. *Delete* regions remove objects on collision, except for the canvas, the mouse cursor, and files. An important property of all regions is that they can be dragged onto other regions *and* other regions can be dragged onto them. Therefore, users can choose the more efficient approach for a certain task or combine both.

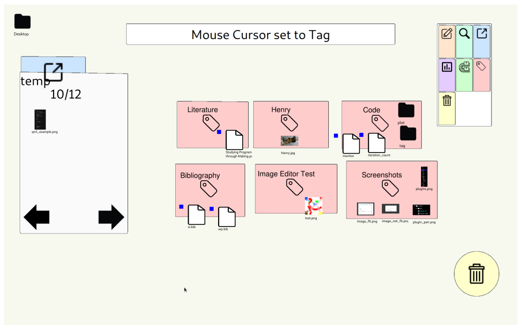
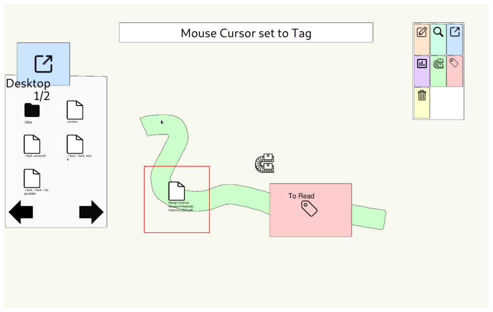


Figure 3: Example of an SI context designed for sorting digital files. This SI context incorporates Tagging regions which category can be typed in by users freely. Additionally, Tag regions outfit colliding TextFile regions with a small blue rectangle in the top left corner to visualize that tag.

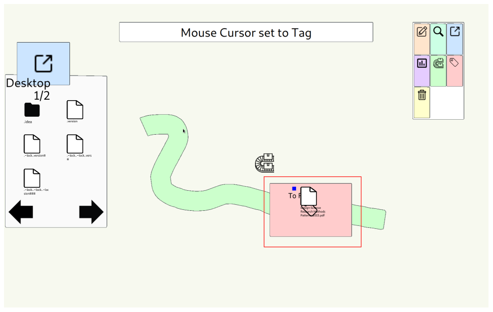
[1]This behavior is provided by SIGRun
[2]For example, the canvas can not be dragged around, and therefore does not register the capability "receive position".

4.2 Automation

In order to automate recurring workflows, users may add special conveyor belt regions (Figure 4) which can transport documents through multiple regions in succession. These conveyor belts also make manual sorting of documents easier as the user does not have to drag the document all the way to a tagging region but just to a conveyor belt leading to that region. Along the conveyor belt region's course, other regions can be placed so that transported documents receive and apply effects when passing through them[3]. In this way, end-users can visually build automated processing pipelines and observe the results of this automation in real-time which enables them to interfere at any point by dragging documents off the conveyor belt.



(a) A text file region (marked red) in transit on the conveyor belt region.



(b) A text file region (marked red) which is transported through a tagging region which applies its tag to the text file.

Figure 4: Simple example for automation with a text file being tagged along the conveyor belt's path.

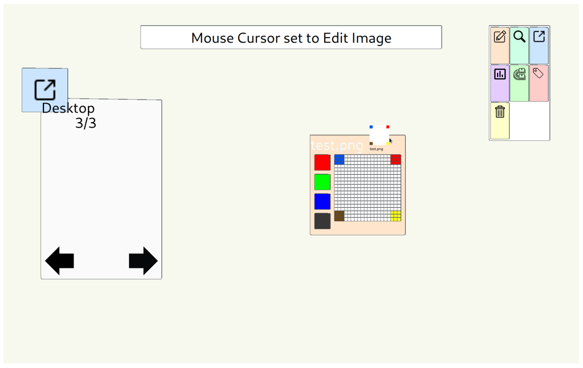
Conveyor belts can can be combined with regions serving as splitters and mergers. This feature allows implementing complex sorting tasks and adding simple processing logic according to the condition for splitting.

4.3 Extension: an Image Editor

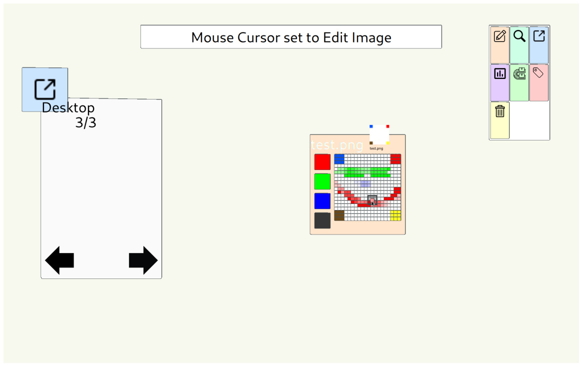
In order to demonstrate that SI can be used for more than simple desktop environments, we implemented an extensible image editor (Figure 5). Image documents can be dragged onto an *Image Editor*

[3]Only document regions accept *transport* events. Tools, such as the *Delete* region, are not transported by conveyor belts by default.

region which in turn displays this image as an array of individual *Pixel* regions. These pixels can then be painted with different brushes and colors. To this end, a brush region is attached to the mouse cursor which emits `color` events to all pixels it touches. A *Blur* tool is implemented via two linked regions: a larger region that samples the colors of all pixels it touches at the same time, and a smaller region which transfers the average color value to the pixel right under the mouse cursor. Dragging the original image file off the Image Editor region saves all modifications on disk and removes the individual pixel regions from the SI context.



(a) When dragging the image onto the editor region, its content gets displayed as individual *Pixel* regions.



(b) Various brushes - also implemented as regions - can be used to draw or apply filters to the image.

Figure 5: An image editor implemented with SI concepts.

5 SUMMARY AND RELEVANCE STATEMENT

With Sketchable Interaction (SI), we offer a new perspective on customizable digital desktop environments where the lines between desktop, file manager, and applications start to blur. This concept allows end-users to build, modify, and extend *their* user interfaces on-the-fly as they see fit. Our reference implementation offers a playground for exploring the concept and a workbench for applying the concept to concrete use cases. Our three example use cases illustrate how SI might be used for iteratively implementing ad-hoc workspaces, building processing pipelines, or developing graphical applications. In all cases, interactive regions act as data representations, GUI elements, tools, and algorithmic building blocks. Therefore, SI is not only a generic toolkit for implementing existing and novel use cases but also an artifact that may foster new perspectives on the architecture of interactive systems.

ACKNOWLEDGMENTS

This project is funded by the Bavarian State Ministry of Science and the Arts and coordinated by the Bavarian Research Institute for Digital Transformation (bidt). From 2017 to 2020, the project was funded in the context of the Centre Digitisation.Bavaria (ZD.B).

REFERENCES

[1] Sriram Karthik Badam, Andreas Mathisen, Roman Radle, Clemens N. Klokmose, and Niklas Elmqvist. 2019. Vistrates: A Component Model for Ubiquitous Analytics. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (Jan. 2019), 586–596. https://doi.org/10.1109/TVCG.2018.2865144

[2] Michel Beaudouin-Lafon. 2017. Towards Unified Principles of Interaction. In *Proceedings of the 12th Biannual Conference on Italian SIGCHI Chapter (CHItaly '17)*. ACM, New York, NY, USA, 1:1–1:2. https://doi.org/10.1145/3125571.3125602

[3] Adele Goldberg and David Robson. 1983. *Smalltalk-80: The Language and Its Implementation.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[4] Dan Ingalls, Ted Kaehler, John Maloney, Scott Wallace, and Alan Kay. 1997. Back to the future: the story of Squeak, a practical Smalltalk written in itself. In *Proceedings of the 12th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications - OOPSLA '97.* ACM Press, Atlanta, Georgia, United States, 318–326. https://doi.org/10.1145/263698.263754

[5] Tobias Isenberg, André Miede, and Sheelagh Carpendale. 2006. A buffer framework for supporting responsive interaction in information visualization interfaces. In *Creating, Connecting and Collaborating through Computing, 2006. C5'06. The Fourth International Conference on.* IEEE, 262–269.

[6] Tobias Isenberg, Simon Nix, Martin Schwarz, Andre Miede, Stacey D. Scott, and Sheelagh Carpendale. 2007. Mobile Spatial Tools for Fluid Interaction. (July 2007). https://doi.org/10.11575/PRISM/30508

[7] Alan C. Kay. 1972. A Personal Computer for Children of All Ages. In *Proceedings of the ACM Annual Conference - Volume 1 (ACM '72).* ACM, New York, NY, USA. https://doi.org/10.1145/800193.1971922 event-place: Boston, Massachusetts, USA.

[8] Clemens N. Klokmose, James R. Eagan, Siemen Baader, Wendy Mackay, and Michel Beaudouin-Lafon. 2015. *Webstrates*: Shareable Dynamic Media. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology - UIST '15.* ACM Press, Daegu, Kyungpook, Republic of Korea, 280–290. https://doi.org/10.1145/2807442.2807446

[9] Roman Rädle, Midas Nouwens, Kristian Antonsen, James R. Eagan, and Clemens N. Klokmose. 2017. Codestrates: Literate Computing with Webstrates. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology - UIST '17.* ACM Press, Qu&#233;bec City, QC, Canada, 715–725. https://doi.org/10.1145/3126594.3126642

[10] Raphael Wimmer and Jürgen Hahn. 2018. A Concept for Sketchable Workspaces and Workflows. In *"Workshop Rrethinking Interaction" in conjunction with CHI 2018.* Montreal, Canada. https://epub.uni-regensburg.de/36818/