



# Journal First: Interacto: A Modern User Interaction Processing Model

Arnaud Blouin, Jean-Marc Jézéquel

## ► To cite this version:

Arnaud Blouin, Jean-Marc Jézéquel. Journal First: Interacto: A Modern User Interaction Processing Model. ICSE 2022 - 44th International Conference on Software Engineering, May 2022, Pittsburgh / Virtual, United States. pp.1-2. hal-03613422

**HAL Id: hal-03613422**

**<https://inria.hal.science/hal-03613422>**

Submitted on 18 Mar 2022

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

# Journa First: Interacto: A Modern User Interaction Processing Model

Arnaud Blouin and Jean-Marc Jézéquel  
firstname.lastname@irisa.fr  
Univ Rennes, Inria, CNRS, IRISA, Rennes, France

## ACM Reference Format:

Arnaud Blouin and Jean-Marc Jézéquel. 2022. Journa First: Interacto: A Modern User Interaction Processing Model. In *Proceedings of The 44th International Conference on Software Engineering (ICSE 2022)*. , 2 pages.

## 1 EXTENDED ABSTRACT

"Anytime you turn on a computer, you're dealing with a user interface" [12]. User interfaces (UIs), and the user interactions they supply, pervade our daily lives by enabling users to interact with software systems. The user interactions provided by a UI form a dialect between a system and its users [9]: a given user interaction can be viewed as a sentence composed of predefined words, i.e. low-level UI events, such as mouse pressure or mouse move. For example, we can view the execution of a drag-and-drop interaction as a sentence emitted by a user to the system. This sentence is usually composed of the words *pressure*, *move*, and *release*, that are UI events assembled in this specific order. The Human-Computer Interaction (HCI) community designs novel and complex user interactions. As explained by [2], "*HCI researchers have created a variety of novel [user] interaction techniques and shown their effectiveness in the lab [...]. Software developers need models, methods, and tools that allow them to transfer these techniques to commercial applications.*" Currently, to use such novel user interactions in software systems developers must complete two software engineering tasks: (i) They must assemble low-level UI events to build the expected user interaction. For example, a developer must manually assemble the events *pressure*, *move*, and *release* to build a drag-and-drop. (ii) They have to code how to process such UI events when triggered by users.

To do so, developers still use a technique proposed with SmallTalk and the *Model-View-Controller* (MVC) pattern in the 80's [10]: the UI event processing model, currently implemented using callback methods or reactive programming [1] libraries. This model considers low-level UI events as the first-class concept developers can use for coding and using increasingly complex user interactions not supported off-the-shelf by graphical toolkits. The reason is that interacting with classical widgets (e.g., buttons, lists, menus) is usually one-shot: a single UI event, such as a mouse pressure on a button or menu, has to be processed. For more complex user interactions such as the drag-and-drop, the current event processing model exhibits critical software engineering flaws that hinder code reuse and testability, and negatively affect separation of concerns and code complexity:

- the concept of user interaction does not exist, so developers have to re-define user interactions for each UI by re-coding them using UI events;

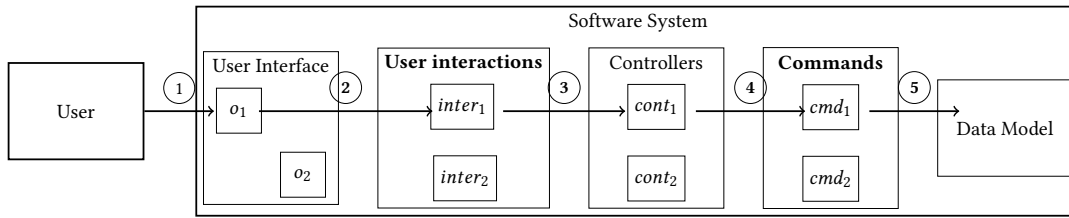
- the model does not natively support advanced features, such as cancellation (undo/redo), event throttling, or logging;
- developers mix in the same code the assembly of UI events and their processing, leading to a lack of separation of concerns;
- the use of callbacks to process UI events (1) can lead to "spaghetti" code [13, 14]; (2) is based on the *Observer* pattern that has several major drawbacks [6, 11, 16, 17]; (3) can be affected by design smells [5];

To mitigate these flaws, we propose *Interacto* as a high level user interaction processing model, depicted by Figure 1. *Interacto* reifies user interactions and UI commands as first-class concerns. *Interacto* makes it easy to design, implement and test modular and reusable advanced user interactions, and to connect them to commands with built-in undo/redo support.

To demonstrate its applicability and generality, we developed two implementations of *Interacto*: *Interacto-JavaFX* with Java and JavaFX [15], a mainstream Java graphical toolkit; *Interacto-Angular* with TypeScript [3] and Angular [8], a mainstream Web graphical toolkits. Both implementations take the form of a fluent API (*Application Programming Interface*) [7]. The following TypeScript code is an example of how *Interacto-Angular* works to use a multi-touch interaction for drawing a rectangle in an SVG canvas. *Interacto* provides a large set of predefined user interactions, such as the multi-touch interaction (with two touches) selected Line 1. Then, the developer specifies the UI command the execution of the user interaction may produce (Line 2), here the command *DrawRect* (coded by the developer). The developers then specifies the graphical component on which the user interaction will operate, here the SVG canvas (Line 3). While the user interaction is running (i.e., while the user is performing a multi-touch), the command is updated using the current interaction data (Line 5). The execution of the command is constrained by a predicate (Line 7).

```
1 this.bindings.multiTouchBinder(2)
2 .toProduce(() => new DrawRect(this.canvas))
3 .on(this.canvas)
4 .then((c, i) => {
5   c.setCoords(...);
6 })
7 .when(i => i.touches[0].src.target === this.canvas)
8 .bind();
```

Listing 1: Using a multi-touch interaction for drawing a rectangle in an SVG canvas with *Interacto-Angular*



**Figure 1: Behavior of the proposed user interaction processing model (in bold what differs from the standard UI processing model):**

- ① : A user interacts with an interactive object  $o_1$  of the user interface.**
- ② : The interactive object then triggers a UI event processed by a running user interaction  $inter_1$ .**
- ③ ④ : Controllers contain Interacto bindings that turns user interaction executions into UI commands.**
- ⑤ : The running Interacto binding executes the ongoing UI command to modify the state of the system.**

Using Interacto, a developers focuses on transforming the execution of a user interaction into UI command instances.

We evaluate *Interacto* interest:

- on a real world case study: the development of LaTeXDraw, an open-source highly interactive vector drawing editor for  $\text{\LaTeX}$ , downloaded 1.5 k times per month and available on more than 10 Linux distributions.
- with a controlled experiment with 44 master students, comparing it with traditional UI frameworks.

The long term experiment shows that the proposal scales for one very interactive and widely-used software system. The experiment conducted with students exhibited several pros and cons of the proposal. The use of *Interacto* is beneficial, in terms of time and correctness, for students to add undo/redo features to the application. The use of predefined user interactions is also beneficial in terms of time and correctness. However, an *entrance barrier* to use correctly *Interacto* may exist.

## 2 MAIN CONTRIBUTIONS INTRODUCED IN THE JOURNAL PAPER

Our work makes the following software engineering contributions:

- We propose a novel model, called *Interacto*, for processing UI events. This model overcomes the growing limits of the current UI event processing model.
- We provide Java and TypeScript implementations of this model to target the JavaFX and Angular UI toolkits. Both implementation are open-source and available on NPM and Maven repositories<sup>1</sup>. Source code is freely available on Github<sup>2</sup>.
- We detail a real world use case that consists of a the development of highly interactive open-source software system using *Interacto*.
- We conducted an experiment with 44 master students that exhibited several benefits to using *Interacto*.

<sup>1</sup><https://www.npmjs.com/package/interacto>

<https://search.maven.org/search?q=interacto>

<sup>2</sup><https://github.com/interacto/>

## 3 POINTER TO THE JOURNAL PAPER

The paper called "Interacto: A Modern User Interaction Processing Model" has been accepted for publication at the IEEE Transaction of Software Engineering Journal on May, the 20th 2021 [4]. It is now available at the following link:

<https://ieeexplore.ieee.org/document/9440800>

We also provide a preprint at the following address:

<https://hal.inria.fr/hal-03231669/>

## REFERENCES

- [1] Engineer Bainomugisha, Andoni Lombide Carreton, Tom van Cutsem, Stijn Mostinckx, and Wolfgang de Meuter. 2013. A survey on reactive programming. *ACM Computing Surveys (CSUR)* 45, 4 (2013), 52. <https://doi.org/10.1145/2501654.2501666>
- [2] Michel Beaudouin-Lafon. 2004. Designing interaction, not interfaces. In *Proc. of AVI '04*. ACM, 15–22.
- [3] Gavin Bierman, Martin Abadi, and Mads Torgersen. 2014. Understanding TypeScript. In *Proc. of ECOOP'14*. 257–281. [https://doi.org/10.1007/978-3-662-44202-9\\_11](https://doi.org/10.1007/978-3-662-44202-9_11)
- [4] Arnaud Blouin and Jean-Marc Jézéquel. 2021. Interacto: A Modern User Interaction Processing Model. *IEEE Transactions on Software Engineering* (2021), 1–20. <https://doi.org/10.1109/TSE.2021.3083321>
- [5] Arnaud Blouin, Valéria Lelli, Benoit Baudry, and Fabien Coulon. 2018. User Interface Design Smell: Automatic Detection and Refactoring of Blob Listeners. *Information and Software Technology* 102 (2018), 49–64.
- [6] Gabriel Foust, Jaakko Järvi, and Sean Parent. 2015. Generating Reactive Programs for Graphical User Interfaces from Multi-way Dataflow Constraint Systems. In *Proc. of GPCE'2015*. ACM, 121–130.
- [7] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.
- [8] Google. 2019. <https://angular.io>
- [9] Mark Green. 1986. A Survey of Three Dialogue Models. *ACM Trans. Graph.* 5, 3 (1986), 244–275. <https://doi.org/10.1145/24054.24057>
- [10] Glenn E Krasner, Stephen T Pope, et al. 1988. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. *Journal of object oriented programming* 1, 3 (1988), 26–49.
- [11] Ingo Maier and Martin Odersky. 2012. *Deprecating the Observer Pattern with Scala. React*. Technical Report, EPFL.
- [12] Z. Mijailovic and D. Milicev. 2013. A Retrospective on User Interface Development Technology. *IEEE Software* 30 (2013), 76–83. <https://doi.org/10.1109/MS.2013.45>
- [13] Brad A. Myers. 1991. Separating Application Code from Toolkits: Eliminating the Spaghetti of Call-backs. In *Proc. of UIST'91*. ACM, 211–220.
- [14] Stephen Oney, Brad Myers, and Joel Brandt. 2014. Interstate: Interaction-oriented language primitives for expressing GUI behavior. In *Proc. of UIST '14*. ACM, 10–1145. <https://doi.org/10.1145/2642918.2647358>
- [15] Oracle. 2018. <https://openjfx.io>
- [16] Guido Salvaneschi, Sven Amann, Sebastian Proksch, and Mira Mezini. 2014. An Empirical Study on Program Comprehension with Reactive Programming. In *Proc. of FSE 2014*. ACM, 564–575.
- [17] Guido Salvaneschi and Mira Mezini. 2014. Towards reactive programming for object-oriented applications. *Trans. on AOSD* 8400 (2014), 227–261.