Programmer Experience When Using CRDTs to Build Collaborative Webapps: Initial Insights

Yicheng Zhang[1], Matthew Weidner[1] and Heather Miller[1]

[1] *Carnegie Mellon University, USA*

Abstract

Conflict-free Replicated Data Types (CRDTs) are data structures that are supposed to make multi-user applications easier to develop. In this paper, we describe our initial insights from a qualitative user study that seeks to answer the question: how hard is it in practice to add real-time collaboration to a web app using CRDTs? We performed coding interviews with 15 participants using three different CRDT libraries; *Automerge*, *Collabs*, and *Yjs*. We observed participants' confusion between local and collaborative data, their struggle to understand replica initialization, and the impact of different library designs on how quickly or slowly participants got started implementing their solutions, and how buggy or not, their solutions tended to be. We hope our work highlights directions for CRDT library developers to further improve.

Keywords: CRDTs. Replicated data. User study. Distributed systems. Concurrency.

1 Introduction

Real-time collaboration is a prominent aspect of popular web applications like Google Docs, Figma, and Notion. Given the success of these high-profile apps where collaboration is a core feature, web application developers are increasingly interested in adding real-time collaboration to their own apps [1]. However, real-time collaboration is difficult for programmers to implement—in particular, keeping data consistent in the face of concurrent edits by different users is challenging for most developers.

Conflict-free Replicated Data Types (CRDTs) provide one solution to this challenge [2][3]. Essentially, CRDTs are data structures that "look like" ordinary data structures (maps, sets, text strings, etc.), except that they are collaborative: when one user updates their copy of a CRDT, their changes automatically show up for everyone else. CRDTs guarantee consistency using expert-designed algorithms. By building an app on top of CRDTs provided by an external library, it is *in principle* possible to make the app collaborative with only modest programmer effort [1][4].

In this paper, we describe informal observations obtained while conducting a qualitative user study that seeks to answer the question: how hard is it *in practice* to add real-time collaboration to a web app using CRDTs? Specifically, we investigate this question for three Javascript/TypeScript CRDT libraries published on npm[1]: *Yjs* [5][6], *Automerge* [7][8], and *Collabs* [9][10]. We challenged 15 participants (5 per library) to add collaboration to a partially-completed web app and asked them to describe what they are trying to do under the think-aloud protocol[11][12]. Ours is the first user study examining how programmers reason about adding collaboration (using CRDTs) that we are aware of.

While we have not completed the data analysis, we share some initial observations and hope it will benefit the PLATEAU community. We discuss our preliminary results in section 3.

2 Methods

In this section, we provide a brief summary of our study design, and introduce three libraries that we chose.

2.1 Participants

We conducted our study with 15 participants (no specific preference over gender identity and information about it is not collected). We presented the same challenge to 5 participants for each of the 3 libraries: (*Automerge*, *Collabs*, *Yjs*). All participants were over 18 and had various levels of education and experience, including undergraduate, graduate, Ph.D., or several years of industry experience. All participants had experience with JavaScript or TypeScript and represent target developers for those libraries, which are mainly novices in Distributed Systems.

1 https://www.npmjs.com/

PLATEAU
13th Annual Workshop at the Intersection of PL and HCI

Organizers:
Sarah Chasins, Elena Glassman, and Joshua Sunshine

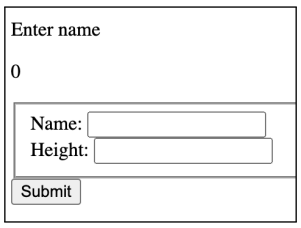

Figure 1. User Interface of Task 1

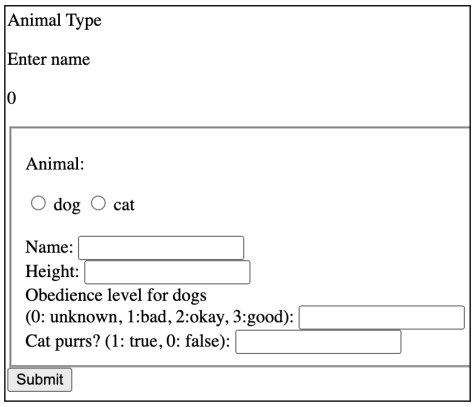

Figure 2. User Interface of Task 2

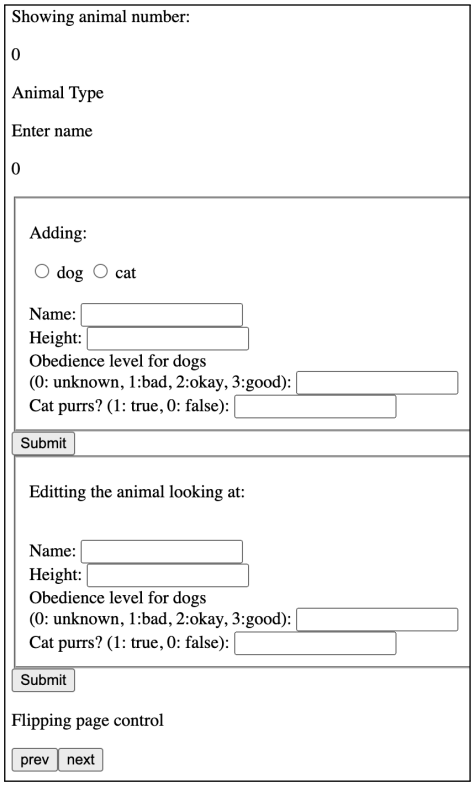

Figure 3. User Interface of Task 3

2.2 Tasks

Participants were asked to complete the provided partially-implemented collaborative Animal Shelter app which maintains a list of animals. Upon completion, users of the app should be able to collaboratively insert new animals and collaboratively change attributes of animals already stored in the app. Participants were asked to complete this task in a period of 90 minutes using TypeScript. We provided the code for a simple user interface that participants should bind to, as well as all P2P communication-related code. Participants were asked to follow the think-aloud protocol while writing code, and were encouraged to ask questions as they went.

All participants focusing on each of the three libraries were given the following tasks with the same general instructions and hints.

- Task 1 (figure 1): implement a data structure that has one animal with properties height and name.
- Task 2 (figure 2): implement a data structure that has one animal that can be either a dog or a cat. A dog has properties of height, name, and obedience level. A cat has properties of height, name, and whether or not it purrs.
- Task 3 (figure 3): implement a list of animals which allows an animal to be added or edited collaboratively. The app user should be able to iterate through the list of animals freely.

2.3 CRDT libraries

Our study considered three CRDT libraries. In particular, we were interested in observing the developer experience when developing collaborative applications across all three library designs. The libraries we focused on in this study are *Automerge*[8], *Collabs*[10], and *Yjs*[6]. All three libraries are comparable in the sense that they all support compositional data, they were created for collaborative web applications, and they are network-agnostic. Automerge and Yjs are by far the only libraries that are used in this field, and Collabs is a library developed by the researchers' group.

Automerge Automerge is a persistent CRDT library that provides tools for building collaborative applications in JavaScript. It is designed for and focused on JSON-formatted data, and is based on

OpSets[13]. It does not specify any network connection protocol but we configured it to use the same connection protocol as Collabs. In order to kick off communication between replicas of *automerge docs*, developers need to explicitly call the network sending and receiving functions. Automerge does not support creating their own new classes in Java style but only JaveScript-style types.

Collabs Collabs[10] [9] is a library created in the authors' research group. It provides CRDTs in the form of strongly-typed data structures, that are modeled on traditional (non-collaborative) collections libraries. It also allows creating custom CRDTs, including custom data models created through composition—e.g., a "CRDT object" composed of CRDT instance fields.

Yjs Yjs[6] is based on the *YATA* CRDT algorithm[5] and supports nested maps, arrays, rich text, and XML structured data. It has seen widespread uptake—it is used in the beta collaborative version of Jupyter Notebook[14]. However, Yjs does support modeling arbitrary data as classes, though its author believes app-specific data models are a good idea[15].

3 Observations and Discussion

In this section, we describe informal observations we made over the course of this study. While we have not yet completed the data analysis step yet of our study, and are currently in the process of qualitative coding, we believe that these initial observations are nonetheless useful to the PLATEAU community, and summarize them here.

3.1 Confusion between local and collaborative data

A point of confusion that we observed across all libraries was that participants sometimes mixed up local and shared data. But it seems easy for people to figure it out by themselves.

For example, in task 3, participants were asked to implement a list of animals that can be collaboratively added to and edited. As shown in Figure 3, the user interface displays an add/edit page to the end user, enabling them to either add a new animal or make changes to the currently selected animal. To navigate, end users are able to page backwards or forwards with the "prev" and "next" buttons.

To implement this, participants need to create a `list_index` to *locally* keep track of the index of which animal is shown. That is, `list_index` is local to the client.

Most participants correctly decided that `list_index` should be local and not collaborative. They did encounter bugs displaying updates to `list_index`, but we believe that these were an artifact of our starter code—in particular, our choice not to use a reactive GUI framework.

3.2 Understanding Replica Initialization

Initialization of replicas was also a point of confusion across all libraries studied. Participants frequently did not consider implementing checks to see if a specific CRDT replica was already present when adding new functionality. This in turn frequently caused what we refer to as *double initialization*, which typically results in data loss.

More concretely, in an application that is built with CRDTs, when one user has initialized the collaborative data structure, other users should not also initialize it. If this happens, it effectively overwrites the first-initialized replica. When this problem is repeated, every operation on a CRDT causes new CRDTs to be created and previous state to be lost, which is counter to the goal of CRDTs in the first place. This problem is particularly pronounced for Automerge, and less so for Yjs.

For example, in Automerge, according to its documentation [16], when adding an object to an *Automerge* doc, users need to first check whether the element already exists, before subsequently adding or modifying it. However, we observed participants tending to initialize it first as shown in Figure 4. This mistake is also hard to debug, as there are no warnings and participants do not know what is happening until a new peer joins. Figure 5 shows correct initialization in Automerge.

We observed that for Automerge, developer-provided documentation provided some coverage of concepts related to initialization of replicas, but it was typically not found by subjects, leading to

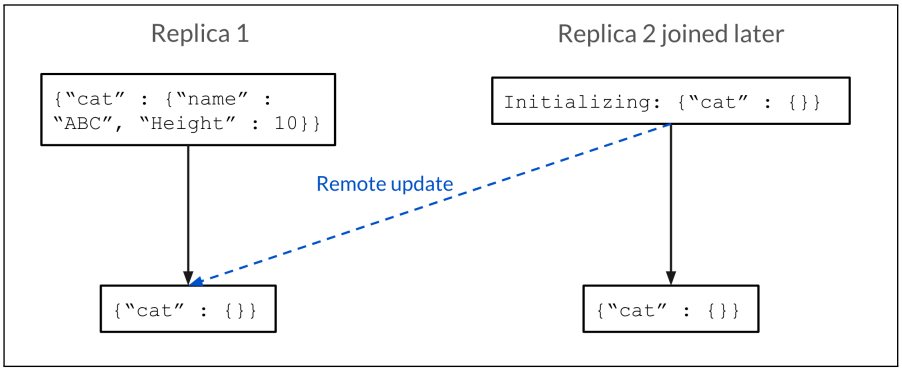

Figure 4. Double initialization logics

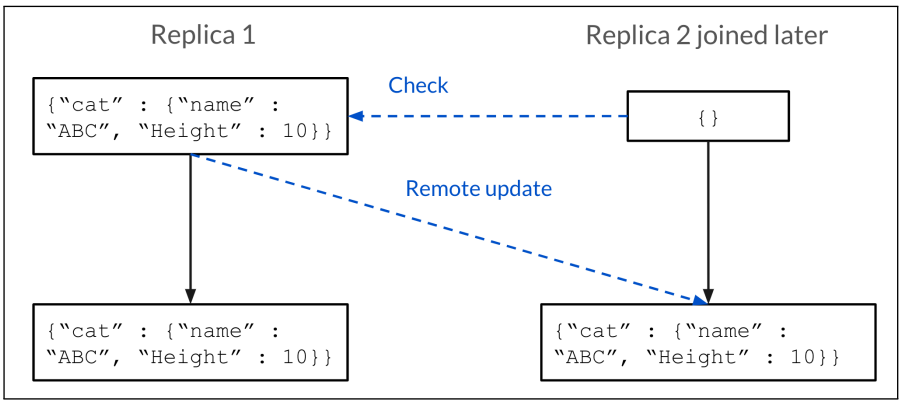

Figure 5. Correct initialization logic

significant confusion and substantial errors for novices attempting to build collaborative apps using Automerge in our study.

3.3 Differences between participants in different libraries

All libraries studied presented both strengths and weaknesses.

For the Automerge library, the JSON formatted files might be more familiar to JavaScript developers, and they may find easier to start, as shown in figure 6. However, it is easier to get into the double-initialization problem, mainly because Automerge is the only one out of the three tested libraries that is not declarative.

Participants working with Yjs to achieve their tasks to be quick and easy, as all data in Yjs is modeled as essentially nested maps or arrays, as shown in figure 7. For example, Task 1 could be solved by simply creating two Yjs maps. On the other hand, because there are only maps and more, it might be more difficult to maintain large, long-lived applications.

In Collabs, users are encouraged to create data structures as class definitions (as in figure 8). Structuring programs in this way provides rich type information, which could lead to easier to reason about code and could lead to a more maintainable code base. However, participants faced a somewhat steeper learning curve and an increased upfront starting cost due to the need to read up on the Collabs library's base type definitions. In order to complete their tasks with Collabs, participants had to describe their data definitions as classes within the Collabs framework. Participants using Yjs on the other hand, lost no time upfront in describing data, because they simply inserted their data freeform into Yjs maps. For Task 1, this results in fewer lines of code for Yjs (∼45) whereas Collabs requires a bit more (∼75). We observed the trade-off here to be one that is similar to points commonly made in the (tired) debates of dynamic vs static languages: in one case, prototyping and getting to a functioning solution is faster, but making updates tends to be more error prone as more functionality is added. And in the other, initial data definitions require upfront effort and slow down initial development, but making updates later tends to be more straightforward and smoother. While

```
// Automerge
// We change the "doc" object by
// passing a lambda function
let newDoc =
  Automerge.change(currentDoc, (doc)⇒{

  // we change the variable as if it
  // was "just" a regular JSON object
  doc.animalName = "my_animal_name";
  doc.animalHeight = 5;
});

// Send the changes to peers explictly
// & update the DOM. Note that such
// sends are not declarative
wsNetwork.send(
    Automerge.getChanges(currentDoc, newDoc)
  );
currentDoc = newDoc;
refreshDisplay();
```

Figure 6. Automerge code example.

```
// Yjs
let doc = new Yjs.Doc();
const wsProvider =
  new WebsocketProvider(wsURL, "", doc);
// get a map, set a map, done
const animal = doc.getMap("animal");
animal.set('name', "my_animal_name");
animal.set('height', 5);
```

Figure 7. Yjs code example.

```
// Collabs
class Animal extends collabs.CObject {
  // Notice that we have two private
  // variables. Both are Collabs,
  // and not local variables.
  private readonly _animalName: CVar<string>;
  private readonly _height: CVar<number>;

  constructor(init: InitToken,
    initName: string, initHeight: number) {
    super(init);

    // Setup child Collabs.
    this._animalName = this.registerCollab(
      "animalName",
      (init) ⇒
        new Collabs.CVar(init, initName)
    );
    this._height = this.registerCollab(
      "height",
      (init) ⇒
        new Collabs.CVar(init, initHeight)
    );
  }

  // Convert our methods into child methods
  get animalName(): string {
    return this._animalName.value;
  }

  set animalName(animalName: string) {
    this.animalName._value = animalName;
  }

  get height(): number {
    return this._height.value;
  }

  set height(height: number) {
    this._height.value = height;
  }
}
```

Figure 8. Collabs code example.

it is possible to wrap Yjs maps in richly-typed classes [14], this requires the use of something like a database object-relational mapping (ORM). And while it's possible for the Yjs authors to include such a mapping as part of Yjs rather than by way of an external framework, it nonetheless would require end users of Yjs to actually provide that type information, which would increase the lines of code required of end users regardless of library.

In sum, we did not observe that any one library stands out in terms of ease of use as compared to the others. Of the three libraries studied, none enabled participants to implement collaborative functionality quickly and without bugs. The Collabs library makes headway on the intialization problem by requiring initial values when calling constructors, but requires more time upfront learning the framework and writing class definitions. Yjs tended to be quick to implement a sketch of a solution in, but initialization and correctly making updates to nested maps presented some obstacles for participants. Automerge, while general, requires initialization repeatedly throughout each object initialization, which participants notably struggled to get right. And finally, in all three libraries, participants struggled to differentiate local from collaborative data.

4 Conclusion

We performed a qualitative study aimed at understanding how novice JavaScript programmers make sense of converting a single-user app into a collaborative one, comparing approaches offered by three different libraries; Automerge, Yjs, and Collabs. Through our work, we observed a few noteworthy artifacts of the design of the various libraries. Notably; (1) the double initialization problem in

Automerge affected several subjects, and may be worth reconsidering in future designs. And (2), we note that both Automerge and Yjs required less code than Collabs for users to create and operate on collaborative data, but at the expense of rich type information and compile-time support for data access. This could be solved by adopting an ORM, but would thus require additional lines of code, as users must provide data definitions *somehow*. In sum, when considering all three libraries, we conclude that none in particular stood out from one another. However, we do suppose that there might be a useful taxonomy or classification that can be devised that can help programmers decide which library to use.

5 Future Work

We plan to complete qualitative coding of the coding interviews, and hope to surface additional insights in doing so. We also hope that authors of any of the frameworks we studied find these initial insights useful to drive usability design decisions for their libraries.

Acknowledgments

We thank our shepherd, Andrew Head, and the anonymous reviewers for their insightful comments. This work was funded by a Cylab Secure and Private IoT Initiative Award, and an NDSEG Fellowship sponsored by the US Office of Naval Research.

References

[1] M. Kleppmann, A. Wiggins, P. van Hardenberg, and M. McGranaghan, "Local-first software: You own your data, in spite of the cloud," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, ser. Onward! 2019, Athens, Greece: Association for Computing Machinery, 2019, pp. 154–178, ISBN: 9781450369954. DOI: 10.1145/3359591.3359737. [Online]. Available: https://doi.org/10.1145/3359591.3359737.

[2] M. Shapiro, N. Preguiça, C. Baquero, and M. Zawirski, "A comprehensive study of Convergent and Commutative Replicated Data Types," Inria – Centre Paris-Rocquencourt ; INRIA, Research Report RR-7506, Jan. 2011, p. 50. [Online]. Available: https://hal.inria.fr/inria-00555588.

[3] N. Preguiça, C. Baquero, and M. Shapiro, "Conflict-free replicated data types crdts," in *Encyclopedia of Big Data Technologies*, S. Sakr and A. Zomaya, Eds. Cham: Springer International Publishing, 2018, pp. 1–10, ISBN: 978-3-319-63962-8. DOI: 10.1007/978-3-319-63962-8_185-1. [Online]. Available: https://doi.org/10.1007/978-3-319-63962-8_185-1.

[4] P. van Hardenberg and M. Kleppmann, "Pushpin: Towards production-quality peer-to-peer collaboration," in *Proceedings of the 7th Workshop on Principles and Practice of Consistency for Distributed Data*, ser. PaPoC '20, Heraklion, Greece: Association for Computing Machinery, 2020, ISBN: 9781450375245. DOI: 10.1145/3380787.3393683. [Online]. Available: https://doi.org/10.1145/3380787.3393683.

[5] P. Nicolaescu, K. Jahns, M. Derntl, and R. Klamma, "Near real-time peer-to-peer shared editing on extensible data types," in *Proceedings of the 19th International Conference on Supporting Group Work*, ser. GROUP '16, Sanibel Island, Florida, USA: Association for Computing Machinery, 2016, pp. 39–49, ISBN: 9781450342766. DOI: 10.1145/2957276.2957310. [Online]. Available: https://doi.org/10.1145/2957276.2957310.

[6] K. Johns, *Introduction*, 2018. [Online]. Available: https://docs.yjs.dev/.

[7] M. Kleppmann and A. R. Beresford, "Automerge: Real-time data sync between edge devices," in *1st UK Mobile, Wearable and Ubiquitous Systems Research Symposium (MobiUK 2018)*, 2018.

[8] A. Contributors, *Automerge crdt: Automerge crdt*, 2022. [Online]. Available: https://automerge.org/.

[9] M. Weidner, H. Miller, H. Qi, *et al.*, *Collabs: Composable collaborative data structures*, 2022. DOI: 10.48550/ARXIV.2212.02618. [Online]. Available: https://arxiv.org/abs/2212.02618.

[10] M. Weidner, H. Miller, H. Qi, *et al.*, *Collabs documentation*, 2022. [Online]. Available: https://collabs.readthedocs.io/en/latest/.

[11] E. Goodman, *Observing the user experience : a practitioner's guide to user research*, eng, 2nd ed. Waltham, MA: Morgan Kaufmann, 2012, ISBN: 0123848695.

[12] J. NIELSEN, "Chapter 6 - usability testing," in *Usability Engineering*, J. NIELSEN, Ed., San Diego: Morgan Kaufmann, 1993, pp. 165–206, ISBN: 978-0-12-518406-9. DOI: https://doi.org/10.1016/B978-0-08-052029-2.50009-7. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780080520292500097.

[13] M. Kleppmann, V. B. F. Gomes, D. P. Mulligan, and A. R. Beresford, *Opsets: Sequential specifications for replicated datatypes (extended version)*, 2018. DOI: 10.48550/ARXIV.1805.04263. [Online]. Available: https://arxiv.org/abs/1805.04263.

[14] JupyterLab, *Real time collaboration*, 2022. [Online]. Available: https://jupyterlab.readthedocs.io/en/stable/user/rtc.html.

[15] K. Jahns, *How we made jupyter notebooks collaborative with yjs*, Jun. 2021. [Online]. Available: https://blog.jupyter.org/how-we-made-jupyter-notebooks-collaborative-with-yjs-b8dff6a9d8af.

[16] A. Contributors, *Make a change: Automerge crdt*, 2021. [Online]. Available: https://automerge.org/docs/tutorial/make-a-change/.