formic: Building Collaborative Applications with Operational Transformation (Work in Progress)

Tim Jungnickel(✉) and Ronny Bräunlich

TU Berlin, Berlin, Germany
tim.jungnickel@tu-berlin.de, r.braeunlich@campus.tu-berlin.de

Abstract. As part of the ongoing revolution of the way people work in distributed teams, the need of applications for real-time collaboration is increasing. Commercial products like Google Docs set the landmark for modern web-based collaboration. In this work we provide a library that utilizes the underlying technology, namely Operational Transformation, to simplify the development of collaborative web applications. Our library *formic* features a novel transformation function that enables simultaneous editing of JSON objects.

Keywords: Consistency control · Operational Transformation · Collaboration · Web development

1 Introduction

The Internet has changed the way we work together and collaboration is no longer restricted to face-to-face meetings. Hence, working in geographically distributed teams will be the predominant part of our future work life [18]. Moreover, due to the increasing number of internet devices everyone uses, we will easily become collaborators with ourselves.

The only way to technically realize usable collaborative applications in high latency networks, like the Internet, is to use a local replica of the application state on every collaborating device. Hence, users can directly access and update the local replica on the device without any noticeable latency. All changes are propagated in the background. Unfortunately, having multiple replicas of the application state raises fundamental questions in distributed systems research. Most important is the need of a consistency control mechanism to ensure convergence among replicas [3].

Successful collaborative applications like Google Docs or Etherpad use Operational Transformation (OT) [7] to allow simultaneous editing of shared documents. We illustrate the mechanics of OT with the following simple text editing scenario. Two users u_1 and u_2 maintain their own replica of the character sequence `abc`. Both users simultaneously invoke edit operations on their local replica. The user u_1 inserts an X at position 0, resulting in Xabc. The user u_2 deletes the character b at position 1, resulting in ac. A naïve interchange of the invoked edit operations would result in diverging replicas: u_1 results in Xbc, whereas u_2 results in Xac. In OT, remote operations are *transformed* based on previously executed local operations. Hence, u_1 needs to transform the position of the remote delete operation to respect the effect of the local insert operation, i.e. u_2's delete operation on position 1 needs to be transformed to a delete operation on position 2 to ensure convergence.

In modern web development, single-page applications based on JavaScript become more and more popular. In frameworks like AngularJS or React essential business logic is executed in the browser at client site, allowing fully responsive user interfaces, even if the network connection is unstable. The major challenge for building collaborative web applications is to combine the chosen web framework with a fitting consistency control mechanism like OT.

Contributions: In this work we contribute to close the gap between the conceptual and formal descriptions of OT systems and real world web development by providing the missing details of an OT extension that supports simultaneous editing of JSON objects. Since JSON is the de facto standard data interchange format of the web, we expect that a combination of JSON with OT encourages the design of more complex collaborative web applications. Ultimately we present and evaluate a programming library that utilizes provably correct transformations and simplifies the development of collaboration systems.

Related Work: OT has been introduced by Ellis and Gibbs in 1989 [7], followed by multiple decades of research around the mechanism and very valuable contributions from various groups. Prominent example applications are Google's document editing suite Google Docs and the free competitor Etherpad. In recent work, Dang and Ignat showed, that the performance of both systems with a larger number of collaborators is limited [6]. Hence, our library must be able to compete with such systems, which we evaluate in Sect. 4.

We have seen notable work outside the academic community that aims to use OT for the development of new collaborative applications. The JavaScript library ShareDB [9] offers the OT mechanism with support for various datatypes such as lists or JSON objects. However, especially the JSON datatype misses a verification of the necessary transformation property (namely TP1).

Apart from OT, other consistency control systems are interesting for collaborative applications, for example Conflict-Free Replicated Data Types [20]. Several benchmarks have been conducted to show the suitability of CRDTs for document editing [1,4]. In recent work, Nédelec et al. introduced a web-based collaborative editor CRATE that enables collaboration without the need of a central server [17]. Kleppmann and Beresford presented a very promising JSON CRDT [15] which has, to the best of our knowledge, not been demonstrated in a collaborative application. In this work, however, we focus on OT systems and compare the performance of Google Docs and ShareDB against our library.

An alternative but noteworthy mechanism is Differential Synchronization by Fraser [8], which is a state based synchronization mechanism based on diffing and patching. An implementation of Jan Monschke demonstrates the applicability to JSON documents [16]. So far we have not seen much academic attention to it.

In earlier research, we presented the conceptual extension of OT to support operations on JSON objects [14]. In this paper, we deliver the missing implementation and evaluation.

2 Preliminaries

In general, OT is an operation based consistency control system, i.e. edit operations on the local replicas of the collaborators are propagated through the network and applied at remote sites. An OT system is composed of a *control algorithm* and a *transformation function* [21]. The control algorithm determines the operations to be transformed and the transformation order. The transformation function determines how the operations are transformed to include the effects of previous operations.

We demonstrate an extract of a transformation function for operations on lists in Listing 1.1, which was initially introduced by Ellis and Gibbs in [2] and improved by Ressel et al. in [8]. We recall the example from the introduction where u_1 inserts a character at position 0 and u_2 simultaneously deletes a character at position 1. According to line number 2 of Listing 1.1, the position of the delete operation must be increased by one. In this example, the *transformation* of the delete operation ensures convergence among replicas.

Listing 1.1. Pseudo code of the transformation function

```
1 function XFORM(insert(i, k1), delete(k2)):
2   if k1 < k2: return(insert(i, k1), delete(k2 + 1))
3   if k1 > k2: return(insert(i, k1 - 1), delete(k2))
4   if k1 == k2: return(insert(i, k1), delete(k2 + 1))
```

One essential and necessary property of a transformation function is the *Transformation Property 1* (TP1) [19]. In essence, TP1 describes that the transformation function needs to repair the inconsistencies that occur if two operation instances are applied in different orders, loosely formalized as:

$$\forall O_1, O_2.\, \mathrm{XFORM}(O_1, O_2) = (O_1', O_2') \Rightarrow (O_2' \circ O_1 = O_1' \circ O_2)$$

OT Systems are not restricted to operations on linear data structures such as lists. However, more complex data structures result in more complex transformation functions, which makes it difficult to prove that TP1 is satisfied. For the rest of this paper, we focus on operations on JSON objects, because we identified them to be most relevant for modern web applications.

JSON (JavaScript Object Notation) is a hierarchically structured data interchange format [10]. A JSON object is an unordered set of key/value pairs. Values can be of primitive type (such as string, number, or boolean) or complex structures, such as arrays or other objects. An array is an ordered list of values.

3 *formic*'s JSON Transformation

Our goal is to provide the necessary mechanics to build collaborative web applications easier and based on solid formal guarantees. Therefore, we implemented *formic* [5], a free software library that features collaboration on list structures, ordered n-ary trees, and JSON. In this section we report on *formic*'s architecture and provide the missing details of the transformation of operations on JSON objects.

Architecture: Our library *formic* utilizes the Wave OT control algorithm, which has been introduced by Google [2]. The algorithm provably ensures convergence among replicas as long as the used transformation function satisfies TP1. Our library implements the client and the server part of the Wave algorithm. Updates among server and client are sent via WebSockets, i.e. a communication protocol for bidirectional communication over a single TCP connection. In case of disconnection, a client can continue to operate *offline* based on the state of the local replica. If the client reconnects, all operations are exchanged and the replicas converge. In contrast to ShareDB and Etherpad, *formic* is, to the best of our knowledge, the first free software OT library that supports an offline mode.

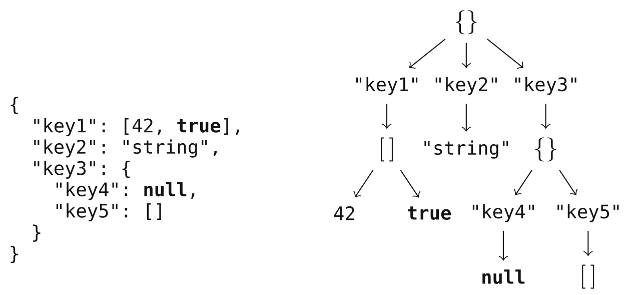
In *formic*, we implemented transformation functions for operations on lists and n-ary trees that are proven[1] to satisfy TP1. Moreover, we developed a transformation function for operations on arbitrary JSON objects.

JSON Transformation: In *formic*, we introduce a novel transformation function, which features insert, delete, and replace operations on arbitrary JSON objects. Since JSON is a hierarchical format, we derive essential parts from the transformation function on n-ary trees, which is already proven to be TP1-valid. The major difference is, that the position parameter, which indicates where an item should be edited, is no longer an *access path*, i.e. a vector with numeric parameters. Since a position inside a JSON is identified with a vector of keys and positions inside an array, the operations must be translated to operations on trees accordingly. For example, the position parameter of an operation on the JSON object in Fig. 1, that aims to insert an item at position `0` of the array with the key `"key5"`, would be translated in the following way:

$$[\texttt{"key3"}, \texttt{"key5"}, 0] \rightarrow [2, 0, 1, 0, 0]$$

The complete translation mechanism from a JSON object to an n-ary tree is documented in the *formic* repository [5]. The transformation function for replace operations can be found in the updated version of the technical report in [13].

[1] For list operations, we implemented a proof in the theorem prover Isabelle [11]. For operations on n-ary trees, we refer to our technical report [13].



```
{
  "key1": [42, true],
  "key2": "string",
  "key3": {
    "key4": null,
    "key5": []
  }
}
```

Fig. 1. Tree representation of a JSON object [14]

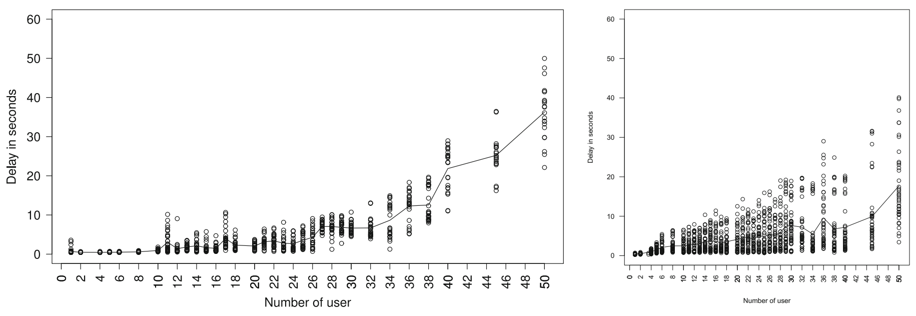
4 Evaluation

We evaluated the performance of our library by reusing the experiment of Dang and Ignat [6], which was initially used to explore the performance of Google Docs at large scale. In their experiment, real users have been simulated with Selenium, a widely accepted web-based testing tool. The simulated users are divided into one Writer, one Reader, and up to 50 DummyWriters. The DummyWriters write random strings to a shared document. The Writer writes a specific string to the document and the Reader waits until the specific string is present and reports the delay. Dang and Ignat measured the delay with different numbers of DummyWriters and various type speed (1–8 keystrokes per second). We used a similar setup to evaluate *formic* by installing the server and the Selenium users on several virtualized machines on our local cluster (16 servers with 2 x Intel Xeon X5355 (2×4 cores), 32 GB Memory).

Note that the original experiment design of Dang and Ignat is based on *simple* insertions of characters and strings to an empty document. Hence, no functionality of a rich text editor is utilized in this experiment. Therefore we decided to compare the performance measurement of Google Docs in [6] to the performance of the transformation of list operations in *formic*. We present the results in Fig. 2.

In contrast to Google Docs, *formic* offers the OT mechanism in a way that web developers can enable simultaneous and collaborative editing of arbitrary objects, as long as the objects can be serialized into JSON. In order to evaluate the transformation of operations on JSON objects properly, we decided to compare the performance of *formic* to ShareDB in an collaborative JSON editing scenario. For this run, we modified the mentioned experiment design so that the DummyWriters are now invoking operations to a shared JSON object over a test website. To ensure comparability, we implemented an identical test website with *formic* and ShareDB and installed both systems on the same local cluster. We present the results in Fig. 3.

Results: In Fig. 2, we show a comparison between the results of Dang and Ignat and the performance of *formic* in an identical setup. We see that *formic* is able to compete with the performance of Google Docs and even shows a better



(a) Google Docs from [6] (b) *formic*

Fig. 2. The performance of collaborative editing with one keystroke per second.

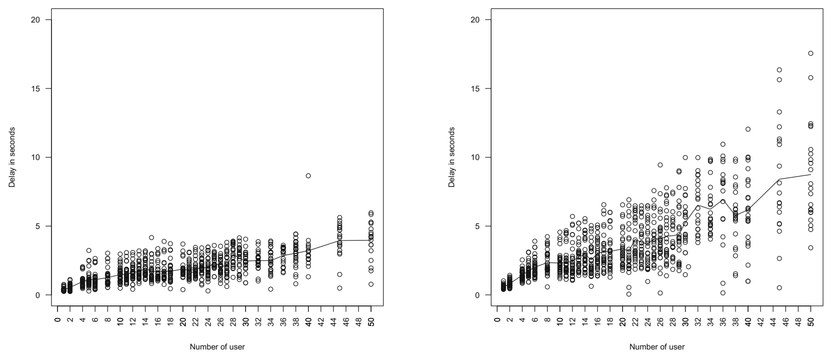
performance at large scale. In Fig. 3 we show the comparison between *formic* and ShareDB in a JSON editing scenario. We see that ShareDB performs slightly better. In contrast to the first experiment, the delay of both systems remains relatively low and does not exceed 10 s.

Discussion: With respect to text editing scenarios, we can confirm the finding of Dang and Ignat, that the performance of OT in collaborative web applications is limited at large scale. However, the performance of *formic* is comparable with Google Docs. We note that the used local cluster for the evaluation of *formic* is relatively old. Hence, we would expect even better results with modern hardware. Unfortunately, Google provides no insight into the used infrastructure and the underlying OT implementation and it is therefore difficult to reason about the performance results of Google Docs.

In the JSON editing scenario, our library performs slightly worse than the competitor ShareDB. We explain the difference in the performance by the used optimizations in ShareDB which are not implemented in *formic* yet. For example, multiple operations on the local replica can be combined before they are sent to the server. This reduces the amount of necessary communication and leads to faster response times. One major bottleneck in *formic* is the mapping of a JSON object to an ordered n-ary tree. The mapping enforces a total order in every layer of the tree, which is technically not necessary for every JSON component. For example, key/value pairs inside a JSON object do not require ordering, whereas elements inside an array must be ordered. This issue can be solved by introducing a more complex data model, which leads, as mentioned in Sect. 2, to more complex proofs. The most interesting solution would use a combination of different consistency control systems to best suit the JSON definition, e.g. a combination of the Observed-Remove Set CRDT [20] and OT.

Ultimately, *formic* is a considerable tool to develop collaborative web applications that are able to compete with established collaborative solutions. The very next step is to improve the accessibility of *formic* by providing examples and developer friendly integrations for commonly used web frameworks.

As a first step, we implemented a collaborative Battleship game based on a shared JSON object and published the source code along with the library [5]. Moreover, we plan to integrate the JSON transformation into a collaborative patient documentation system [12].



(a) ShareDB (b) *formic*

Fig. 3. JSON editing with one modification per second.

5 Conclusion

With *formic*, we presented an open-source library that simplifies the development of web-based collaborative applications by providing a fully working OT system with implemented transformation functions for operations on lists, trees and JSON objects. Moreover, *formic* is composed of tested and verified components. Especially the used transformation functions are proven to be TP1-valid, which makes *formic* an attractive tool to solve the consistency-related challenges in collaborative applications.

The conducted experiment has demonstrated that our library is able to compete against Google Docs, the most successful collaborative application that utilizes OT. However, the underlying client-server architecture limits the performance at large scale.

Ultimately, the development of collaborative web applications will become more important in the future. The use of a single service for web-based collaboration, such as Google Docs, is highly questionable in terms of privacy and confidentiality, especially for sensitive data. Therefore, we expect to see more organizations, which use in-house applications for the collaborative work in distributed teams. The presented library simplifies the development and enables the design of collaborative applications that are not restricted to collaborative text editing, but rather fully flexible due to the JSON transformation. Future work includes the improvement of the accessibility and the development of more features, such as undo/redo or move operations.

References

1. Ahmed-Nacer, M., Ignat, C.-L., Oster, G., Roh, H.-G., Urso, P.: Evaluating crdts for real-time document editing. In: ACM Symposium on Document Engineering (2011)
2. Apache. Wave Protocol (2014). https://incubator.apache.org/wave
3. Brewer, E.: Towards robust distributed systems. In: Principles of Distributed Computing, PODC 2000 (2000). (Invited Talk)
4. Briot, L., Urso, P., Shapiro, M.: High responsiveness for group editing crdts. In: International Conference on Supporting Group Work, pp. 51–60 (2016)
5. Bräunlich, R.: formic (2017). https://github.com/rbraeunlich/formic
6. Dang, Q.V., Ignat, C.L.: Performance of real-time collaborative editors at large scale: user perspective. In: IFIP Networking Conference and Workshops, pp. 548–553 (2016)
7. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. SIGMOD Rec. **18**(2), 399–407 (1989)
8. Fraser, N.: Differential synchronization. In: ACM Symposium on Document Engineering, pp. 13–20 (2009)
9. Gentle, J., Smith, N.: ShareDB (2016). https://github.com/share/sharedb
10. JSON. JavaScript Object Notation (1999). http://json.org
11. Jungnickel, T.: A proof of tp1 for transformations of list operations (2015). https://gitlab.tubit.tu-berlin.de/jungnickel/isabelle
12. Jungnickel, T., Cabello, J., Raile, K.: Hotpi: open-source collaborative patient documentation. In: Companion of ACM Conference on Computer Supported Cooperative Work and Social Computing, pp. 219–222 (2017)
13. Jungnickel, T., Herb, T.: Tp1-valid transformation functions for operations on ordered n-ary trees (2015). http://arxiv.org/abs/1512.05949
14. Jungnickel, T., Herb, T.: Simultaneous editing of JSON objects via operational transformation. In: ACM Symposium on Applied Computing, pp. 812–815 (2016)
15. Kleppmann, M., Beresford, A.R.: A conflict-free replicated JSON datatype (2016). http://arxiv.org/abs/1608.03960
16. Monschke, J.: DiffSync (2015). https://github.com/janmonschke/diffsync
17. Nédelec, B., Molli, P., Mostefaoui, A.: Crate: writing stories together with our browsers. In: International Conference Companion on World Wide Web, pp. 231–234 (2016)
18. Powell, A., Piccoli, G., Ives, B.: Virtual teams: a review of current literature and directions for future research. SIGMIS Database **35**(1), 6–36 (2004)
19. Ressel, M., Nitsche-Ruhland, D., Gunzenhäuser, R.: An integrating, transformation-oriented approach to concurrency control and undo in group editors. In: ACM Conference on Computer Supported Cooperative Work, pp. 288–297 (1996)
20. Shapiro, M., Preguiça, N., Baquero, C., Zawirski, M.: Conflict-free replicated data types. In: Défago, X., Petit, F., Villain, V. (eds.) SSS 2011. LNCS, vol. 6976, pp. 386–400. Springer, Heidelberg (2011). doi:10.1007/978-3-642-24550-3_29
21. Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D.: Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Trans. Comput. Hum. Interact. **5**(1), 63–108 (1998)