Simultaneous Editing of JSON Objects via Operational Transformation

Tim Jungnickel
TU Berlin
Germany
tim.jungnickel@tu-berlin.de

Tobias Herb
TU Berlin
Germany
tobias.herb@tu-berlin.de

ABSTRACT

Real-time collaboration among geographically distributed users is becoming increasingly important in our private and professional lives. Operational Transformation (OT) is a well established technique for consistency maintenance in the domain of collaborative text editing. In this paper we expand the scope of OT to support simultaneous editing of shared JSON objects. The presented approach can be used in almost any modern web framework to implement collaboration tools that are, in contrast to popular real-time collaboration tools like Google Docs, no longer restricted to text documents. Alongside with a detailed presentation of the underlying data structures and mechanics we discuss how web applications can be designed to use our extension.

CCS Concepts

•Human-centered computing → Collaborative and social computing systems and tools; •Information systems → *Web applications;*

Keywords

Operational Transformation; Consistency Maintenance; Collaboration; JSON; Web Applications

1. INTRODUCTION

Real-time collaboration among geographically distributed users has been revolutionized by the introduction of web based document editing tools. One of the first widely accepted products for collaborative text editing was Google Docs. Within a Google Docs text document multiple users can simultaneously edit text. The changes to the document are synchronized in the background and the displayed text of every connected user will be updated automatically. Meanwhile the scope of the application has been extended e.g., to collaborative spreadsheets and other companies introduced their products for collaborative document editing.

SAC 2016,April 04 - 08, 2016, Pisa, Italy

ACM 978-1-4503-3739-7/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2851613.2852003

For the users of this web based collaboration systems the synchronization process is completely transparent. The updates of other users simply result in appearing or disappearing characters. In addition a consistent result is guaranteed i.e., after the synchronization has finished the displayed document is identical for all users. The underlying technique for synchronization and consistency maintenance is Operational Transformation (OT). It has been introduced by Ellis and Gibbs in [3].

In this paper we present a novel approach that expands the scope of operational transformation to generic objects formatted in the JavaScript Object Notation (JSON). Since JSON is the de facto standard data interchange format for web applications, many applications can be made collaborative with our approach and benefit from a Google Docs-like way to synchronize changes. Instead of focusing on one specific implementation, we introduce our extension in a generic but programming-language-near notation, so that our approach can be adopted and implemented easily in various programming languages. Moreover we have figured a formal proof and show that our approach is correct and that consistency is guaranteed.

Operational Transformation.

An OT system provides a replicated framework for shared documents to ensure good responsiveness in high latency environments like the Internet [6]. Edit-operations are immediately applied to local sites, thus the effects instantly become apparent, and are then propagated to remote sites. This causes, in the case of simultaneous editing, different sites to apply operations in different orders. Since operations are typically not commutative, different orders lead to different states. OT tackles this problem by applying a transformation function on pairs of operations (local, remote) to achieve a pair of operations where the effects are mutually included. The transformation ensures that consistency is achieved across all sites despite different orders of operations.

To give some intuition of the mechanism, we demonstrate OT with a simple text editing scenario. The sites S1, S2

replicate the character sequence `XYZ`. S1 inserts character `A` at position 0, resulting in `AXYZ`. S2 simultaneously deletes the character `Y` at position 1, resulting in `XZ`. The operations are exchanged between the sites. If the remote operations are applied naively, then we get inconsistent replicas: S1 results in `AYZ` and S2 results in `AXZ`. Therefore, a transformation of the remote operation is performed, before the operation is applied. At S1, the position of the remote delete operation is incremented with respect to the local insert operation. At S2, the remote insert operation does not need to be transformed since the local delete operation has no effect on the remote insert operation. The transformations of the position of simultaneous operations leads to consistent replicas.

In general OT systems are divided into a control algorithm and a transformation function [9]. The control algorithm determines the operations to be transformed and the transformation order. Known representatives of such algorithms are Jupiter [6] and the Wave Protocol [1]. The transformation function determines how the operations are transformed to include the effects of previous operations.

The major challenge in this work is the hierarchical structure of JSON objects. A classic linear transformation, as presented in the example above, cannot be applied and the transformation needs to be extended to support the tree-like framework of JSON. Therefore base our approach on a transformation function for insert and delete operations on generic ordered n-ary trees and show how objects in JSON are mapped.

The major problems of applying linear OT on a hierarchical structures are simultaneous inserts of hierarchical nodes. This can be well exemplified by an HTML example. We reuse our document containing the sequence `XYZ`, replicated on the sites S1, S2. Site S1 decides that character `Y` should be in bold. This can be represented with two insert operations, producing the document state `X<b>Y</b>Z`. Simultaneously S2 decides that character `Y` should be in italic. If the server receives S2's edit after S1, it would need to transform the operations of S2 against the operations of S1. Usually the server would be configured to place the later edit behind the first edit. If the operations of S1 are applied before the transformed operations of S2 are applied, then the document will be syntactically incorrect: `X<b><i>Y</b></i>Z`.

Contributions.

The contributions of this paper are the following:

- We present a generic hierarchical approach that allows the edit of JSON objects based our verified transformation function for ordered n-ary trees.
- We introduce a mapping where our generic hierarchical approach is applied to hierarchical JSON documents.
- We show how changes to JSON objects can be synchronized in an elegant Google Docs like way.
- We discuss how the JSON synchronization should be implemented so that modern web applications can benefit from our approach.

2. PRELIMINARIES

In this section we define the necessary preliminaries. We give an intuition what operations we consider and introduce the OT specific preliminaries i.e., the definition of a transformation function and one important transformation property.

We consider an *operation* to be an identifier together with zero or more input values and well defined semantics. For example the insert_L operation has three input values: an item to be inserted, a position and a list. The semantics is given by the implementation. Sometimes we consider instances of an operation i.e., operations with given input values. For example $O_1 = \text{insert}_L(a, 2, [x, y, z])$. In this case we call the list $[x, y, z]$ the *context* of this instance since the result would be a new context for another instance. We denote the context of one instance O_1 as $C(O_1)$.

Definition 1 (Transformation Function). *A* Transformation Function *has a pair of operation instances* (O_1, O_2) *with* $C(O_1) = C(O_2)$ *as input parameter and returns a pair of transformed operation instances* (O_1', O_2') *where* O_1' *is the transformed version of* O_1 *with* $C(O_1') = O_2$ *and* O_2' *is the transformed version of* O_2 *with* $C(O_2') = O_1$.

Our definition of the transformation function is based on the introduction of the Jupiter OT system [6]. The input of a transformation function can be seen as two independent operation instances of two sites. Both sites apply the operation instance to a local replica of a context which may result in inconsistent replicas over both sites. In order to achieve consistent replicas, the transformed versions of the operation instances are exchanged and applied by both sites.

Definition 2 (Transformation Property 1). *Let* O_1 *and* O_2 *be two arbitrary operation instances with identical context.* Transformation Property 1 *(TP1) is satisfied by a transformation function* XFORM*, if the following holds for* $\text{XFORM}(O_1, O_2) = (O_1', O_2')$*:*

$$O_2' \circ O_1 = O_1' \circ O_2$$

One essential property of the transformation function to achieve consistent replicas is the *Transformation Property 1* (TP1) [7]. In essence TP1 describes that the transformation function needs to repair the inconsistencies that occur if two operation instances are applied in different orders.

In order to achieve a working OT system, a control algorithm like Jupiter [6] and a TP1-valid transformation function for operations on the desired data structure is necessary. In [5] we published a technical report that covers transformation functions for operations on lists and ordered n-ary trees with proofs for the TP1-validity.

3. JSON SYNCHRONIZATION

In this section we describe how a transformation function for ordered n-ary trees can be applied to operations on JSON objects. Later we discuss how our approach can be implemented in web applications.

In JSON, the data is structured in three components: an object (1), an array (2) and a value (3). An object (1) is a

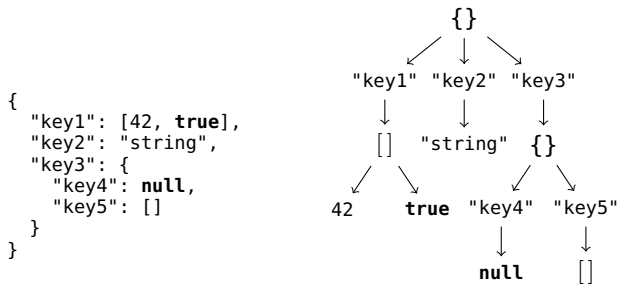

Figure 1: Tree representation of a JSON object

unordered set of key/value pairs. An array (2) is an *ordered* list of values and a value (3) is either an array, an object or a simple type like a string or a number. We see that JSON is hierarchical since a value can be an object and that the data is ordered with respect to the values inside an array. The transformation function we presented in [5] supports operations for a tree with ordered child nodes. Hence we handle both challenges in one step. To do so, we map the JSON structure to the tree structure in order to achieve an OT synchronization of JSON objects.

In Fig. 1 we see the tree representation of a JSON object. The tree representation has four components: an object node, a key node, a value node and an array node. An object node, denoted as {}, is the parent of arbitrary many key nodes. A key node has exactly one value node as child. A value node either is a simple type with no child nodes, an object node or an array node. An array node, denoted as [], is the parent of arbitrary many value nodes.

With the tree representation of a JSON object, we can use the operations insert and delete to modify the tree and the corresponding JSON object. We note that we need to extend the definition of a *valid* operation in [5] in order to ensure that the mentioned constrains from the previous paragraph are satisfied, but this can be done without much effort. The constrains are still easy to check and both server and client can reject invalid operations.

The insert and delete operations in [5] are working with a numeral positional parameter i.e., the access path to identify where an item should be inserted or deleted from. Except for the positions inside an array, JSON uses keys to identify values. In order to provide fully usable insert and delete operations we need to extend the definition of an access path to support keys as identifier. However this could be easily achieved by masquerading the internal position of an element by a key. We also note that we created a little overhead, since, in this representation, an object is an *ordered* list of key/value pairs. However this also can be masqueraded by providing a function that inserts a key/value pair at a random position inside an object. Ultimately we have a mapping from the JSON structure to a tree structure and achieved a way to simultaneously edit JSON objects via OT.

3.1 Implementation

Implementations of our approach should follow, but are not restricted to the design of Google Docs. Google has not published the source code yet, however the essential parts of Google Docs are derived from a former Google product named *Wave*, which is now an open source project of the Apache Software Foundation [1]. The underlying architecture is a client/server model where both sides perform transformations of operations.

The server implements the appropriate side of the Wave OT Control Algorithm, which is documented on Wave's project website [1], in a desired programming language. In most related work, the server is implemented in NodeJS [4]. With the documentation of the Wave Control Algorithm alongside with a proper transformation function, a working OT server for JSON can be implemented in any modern web framework and programming language.

The client implements the appropriate side of the Wave Control Algorithm in JavaScript and should use AJAX to exchange local and remote operation instances. With a working JavaScript implementation on the client side, lots of necessary transformations are transfered from the server to the client. This is cost efficient for the provider of the service since this reduces the necessary RAM and CPU utilization of the server.

The major strength of the OT technology together with current web technologies is, that for the user all own changes are instantly apparent. Even if the client has a bad connection with high latencies, the user has the feeling of a real-time application.

4. RELATED WORK

Davis et al [2] were, to our knowledge, the first that applied the OT approach on treelike structures. They extended operational transformation to support synchronous collaborative editing of documents written in dialects of SGML (Standard General Markup Language) such as XML and HTML. They introduced a set of structural operations with their associated transformation functions for SGML's abstract data model grove. Their approach is followed by [8, 10] showing improvements in XML editing and implementations in collaborative business software. Our work fills the gap that is needed to lift OT for trees to OT for JSON objects. Since JSON is the de facto standard data interchange format for the web, new forms of collaborative web application can benefit from our approach. We are, to our knowledge, the first that present an OT way to synchronize insert and delete operations of JSON objects.

OT also gets more and more attention outside the academic community. This becomes apparent through various open source projects. The most interesting representative for us is ShareJS [4]. An OT library based on JavaScript that allows an easy integration of live concurrent editing in web applications. The library additionally supports concurrent editing of hierarchical JSON objects. As opposed to our generic approach, ShareJS only offers support for JSON-based hierarchical structures. Further exists, to our knowledge, no verification of the used transformation functions and no publication or abstracted documentation so that a reimplementation is rather difficult. In contrast, our work enables the programming language independent synchronization of operations on JSON objects and our mapping can be imple-

mented in any suited scenario. We give the missing intuition that is necessary to rebuild ShareJS based on verified transformations.

Another interesting Open Source library is JOT (JSON Operational Transformation) [11] that also supports the concurrent modification of JSON objects. This approach does not represent the hierarchical document structure of an ordered tree. It is not apparent how nested element access is done. However they introduce an interesting rebase transformation, which is illustrated by the following example: The state of the replicated document is "key1": "Hello world!". User A makes the following change "title": "Hello world!" by applying the operation A = rename("key1","title"). Simultaneously user B makes the following change "key1": "My Program" by applying the operation B = set("key1","My Program"). In order to keep the replicas consistent, the remote operation must be rebased against the incoming operation. This kind of transformation is, unlike almost all other OT approaches, not based on positional addressing, but on a structural property of the JSON document. However a verification and an abstracted documentation is missing.

5. CONCLUSION

With the Internet we have connected people all over the world. Real-time collaboration is no longer restricted to one location. Unfortunately high latency environments like the Internet can slow down or freeze a collaboration, if one site is blocked as long as another site performs changes. OT solves this problem by replicating a shared source in an optimistic way i.e., all sites can perform changes to local replicas and the synchronization happens in the background.

The impressive breakthrough of web tools for real-time collaboration like Google Docs is the result of continuous improvement of the underlying OT technology. With this paper we contribute to the series of improvements of OT by presenting our approach to extend this technology to synchronize edit operations on JSON objects.

The data interchange format JSON is de facto standard in almost all modern web frameworks. With our presented approach, web applications can be extended to support real-time collaboration on generic data, as long as the data can be represented in JSON. Hence compared to products like Google Docs, the elegant OT way of synchronize changes is no longer restricted to just text documents.

Future work includes the implementation of programming libraries to provide an OT synchronization of edit operations of JSON objects in various programming languages that are used in current web frameworks. In addition we plan to contribute our transformation function and a JSON implementation to the well used and accepted JavaScript OT libraries ShareJS [4] and JOT [11].

6. REFERENCES

[1] Apache. Apache wave. https://incubator.apache.org/wave/, 2012.

[2] A. H. Davis, C. Sun, and J. Lu. Generalizing operational transformation to the standard general markup language. In *Computer Supported Cooperative Work*, CSCW '02, pages 58–67, 2002.

[3] C. A. Ellis and S. J. Gibbs. Concurrency control in groupware systems. *SIGMOD Rec.*, 18(2):399–407, 1989.

[4] D. Govett and J. Gentle. Json ot type. https://github.com/share/sharejs, 2015.

[5] T. Jungnickel and T. Herb. Tp1-valid transformation functions for operations on ordered n-ary trees. http://arxiv.org/abs/1512.05949, 2015.

[6] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Symposium on User Interface and Software Technology*, UIST '95, pages 111–120, 1995.

[7] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Computer Supported Cooperative Work*, CSCW '96, pages 288–297, 1996.

[8] H. Skaf-Molli, P. Molli, C. Rahhal, and H. Naja-Jazzar. Collaborative writing of xml documents. In *Information and Communication Technologies: From Theory to Applications*, ICTTA '08, pages 1–6, 2008.

[9] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1):63–108, 1998.

[10] C. Sun, S. Xia, D. Sun, D. Chen, H. Shen, and W. Cai. Transparent adaptation of single-user applications for multi-user real-time collaboration. *ACM Trans. Comput.-Hum. Interact.*, 13(4):531–582, 2006.

[11] J. Tauberer. Json operational transformation (jot). https://github.com/JoshData/jot, 2013.