Operational Transformation

Robert Krahn

Software Architecture Group
Hasso Plattner Institut
Potsdam, Germany
robert.krahn@student.hpi.uni-potsdam.de

Abstract. Multi-user applications allow participants to cooperatively author shared documents in real-time. The document state in these applications is distributed across several application instances and must be synchronized in order to ensure consistency for all participants. Since network latency can be high and human users demand immediate feedback, synchronization strategies must take these requirements into account. In this paper we describe the Operational Transformation framework. This framework synchronizes operations between application instances optimistically and addresses the requirements typically occurring in a multi-user application. Furthermore, we describe how a multi-user Web chat application can be implemented using the Operational Transformation approach.

1 Introduction

Multi-user applications are distributed systems in which two or more users are working in real-time on a shared document [1]. We call the distributed parts of the multi-user system *application instances*. Typically, there exists one application instance per user and, if a central-server exists, there runs at least one application instance on the server.

Generally, the system state can be changed at any time in all application instances. To allow users to work on the same state, the state of all application instances has to be synchronized mutually. Because of efficiency reasons, only the actions that lead to state changes (e.g. adding a character at a certain position to a text) are used for synchronization. We call those actions *operations*. This means, synchronization can be reduced to exchanging operations. If operations happen concurrently at at least two application instances then a conflict occurs.

Figure 1(a) shows non-conflicting operations on the text "hallo". At application instance A the fourth character is deleted. After the operation is applied locally it is send to application instance B where it is also applied, leading to a *consistent*[1] state in all application instances. Another operation is generated and applied at B and send to and applied at A. In this example the application instance states could be successfully synchronized by simply sending and applying operations.

[1] Consistency is defined in section 3.

Now consider figure 1(b). Here both deletions happen concurrently, i.e. the deletion at A is generated and applied before the deletion operation from B arrives and deletion at B is generated and applied before the deletion generated at A arrives. For synchronization the operations are now send to and applied at the application instances. The final text state at A and B differs and the users cannot work consistently on the same document any longer. A more sophisticated synchronization strategy is required.

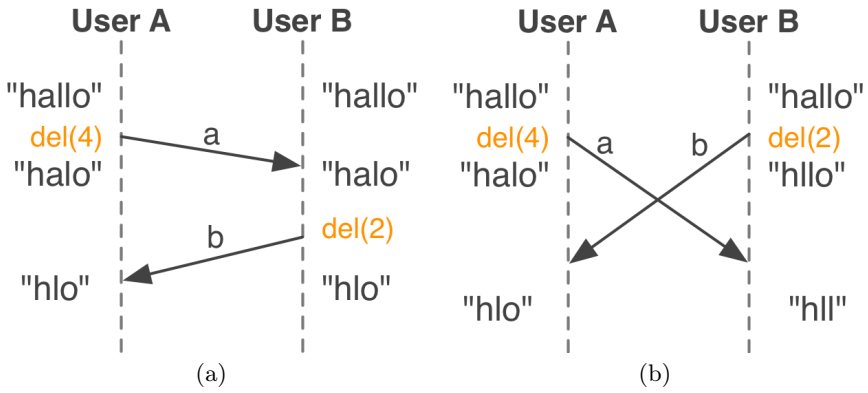

Fig. 1: Non-conflicting and conflicting text operations.

To fulfill the real-time requirement, multi-user applications require fine-grained operations. They also have to be responsive because usually human users interact with them.

Summing up, multi-user applications have to fulfill several requirements:

- Fine granularity of sharing,
- two or more participants,
- participants can concurrently modify data with a consistent outcome,
- the application should be highly interactive and response times should be minimized.

In section 2 we describe how systems that fulfill the presented requirements can be classified. Based on that classification we present the concurrency control approach of the Operational Transformation framework in section 3. In section 4 we introduce a Web chat application implemented using Operational transformations. Section 5 discusses the related work and in section 6 we give a summary and an outlook.

2 Concurrency Control Approaches

There exist several strategies to solve the presented problems. On the one hand, these approaches differ on whether they synchronize optimistically or pessimisti-

cally [2]. Pessimistic approaches require network communication with other instances. This means that the operation is send synchronous to other instances and only applied at the issuing instance after a response from other instances arrives. A multi-user system using this approach is Croquet [3]. In Croquet each distributed event is temporarily ordered. This approach allows to execute the events in the same order across all Croquet instances. The drawback of pessimistic algorithms is that a complete network round-trip has to be done. In low-latency networks this can result in a delayed user response, leading to a less interactive system.

Systems using an optimistic synchronization strategy strategies on the other hand apply operations instantly at the issuing site. If a conflict occurs because another application instance had applied an operation B that was note present at the issuing instance at the time that operation A was created and applied then these operations have to be transformed in order to reach the same state across all instances. This approach requires that all operations must be transformable. User response times are not dependent of network delay. Therefore optimistic synchronization has an advantage in environments with high network latency.

Concurrency control can also be distinguished on whether they have a central server or a fully distributed peer-to-peer network. In peer-to-peer networks instances send and receive operations to and from an arbitrary number of other instances. This means that they have to be able to synchronize operations received from multiple sources.

Concurrency control systems using a central server, however, need only to synchronize the state between server and each instance. Only the server has to maintain multiple connections. Multi-user applications using a central server can still have an arbitrary number of participants. Additionally, the server can keep the application state when no client is connected, keeping the system always "alive".

3 Operational Transformation

Operational Transformation (OT) is a conceptual framework for concurrency control [1, 4, 5]. It is optimistic and can be implemented with a peer-to-peer or client-server architecture. The fundamental responsibilities of the OT framework can be separated into two tasks: How and when operations are applied to application instances and how operations are modified at the application instances to ensure the *consistency* of the shared state across all application instances. Consistency in this context means [1, 6]:

- Eventually the state of all instances must *converge*. If the same operations are applied to the application instances their state must be eventually identical.
- The *intention* of an operation must be maintained. Depending on the semantic of the operation, the modified operation should not confuse the application user. For example, if the shared state is a text and a conflict between two text operations occurs then a solution would be to delete the whole text.

As long as this is done at all application instances the convergence requirement would be satisfied. However, this solution would not incorporate the intention of the user.
- The *causality* of operations should be preserved. If an operation causally precedes another operation[2], then at each application instance the execution of the first operation (respectively its transformation) happens before the execution of second operation.

In the following we will explain what an operation transformation is, what properties is must have, and how operations are transformed across application instances to ensure consistency.

3.1 Transformations

We presented a conflicting text operation in the example shown in Figure 1(b). The conflict occurred because operations are applied optimistically at the generating instance and synchronization is done afterwards.

How is this conflict solved using transformations? Figure 2 shows the solution. When the deletion $del(2)$ generated at application instance B is received by application instance A, it is identified as a conflicting operation. The conflict exists between $del(2)$ and $del(4)$ because the state was synchronized before those operations were generated. For application instance A $del(2)$ is transformed, however, since $del(4)$ should have no impact on a deletion before that position the transformation result is the same $del(2)$. It is applied at application instance A. At application instance B we have to transform the received $del(4)$ operation so that the resulting state is as if $del(4)$ was applied before $del(2)$. This means application instance B transforms $del(4)$ and gets $del(3)$ as a result which is then applied. Both application instances now have the same state.

Generally a transformation function $transform(a, b)$ accepts two conflicting operations a, b as input and produces two transformed operations a', b' so that $a \circ b' = a' \circ b$. This property must be fulfilled by all transformation functions [1][3].

In the example both application instances execute $transform(del(4), del(2))$ and get $del(2), del(3)$. Only one result is applied, the other one is stored to resolve further conflicts as explained in section 3.2.

A general deletion transformation function could have the implementation shown in listing 1.1.

```
transform(del(i), del(j)):
  if i > j return del(i-1), del(j)
  if i < 1 return del(i), del(j-1)
  return no-op, no-op
```

Listing 1.1: Transformation function for deletion operations

[2] This means that the generation and execution of the first operation happened before the second. For a formal definition and detailed discussion see [6]. The *happened-before* relation is defined in [7].

[3] $x \circ y$ means first operation x and then operation y is applied.

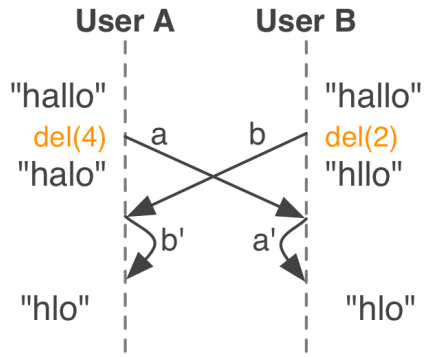

Fig. 2: Resolution of the conflict shown in figure 1(b).

Depending on the application, transformation for other operations have to be implemented. For a text editor, for example. the *add* operation is necessary and if also rich text is supported then also operations for manipulating text attributes are required. Transformations might also be necessary between different kinds of operations. For a text editor not only $transform(add, add)$ must be implemented but also $transform(add, del)$ and $transform(del, add)$. Thus, in the worst case, an application with n operations requires n^2 transformations. However, in real applications the number of necessary transformations is lower because not all concurrent operations create a conflict [2].

As shown in the example above. the Operational Transformation approach is usually used to solve operation conflicts automatically. However, it also possible to implement a manual conflict resolution mechanism as it is done in the CoWord project [8, 5] for a few operations.

3.2 Merging Multiple Operations

The general control flow in OT is the following. An operation created by an application instance is first applied locally at that instance and then send to other application instances[4]. At the other instances the state change is transformed using a transformation function so that other changes that were created at that instance concurrently are taken into account. The transformed changes are then applied at the other application instances.

Until now we have only considered the case of two conflicting operations. How can the state made consistent if more then one operation is made concurrently? See figure 3 for an example. The state transitions of a text for two application instances A (orange) and B (blue) is shown. Both start with an empty text. The numbers below the text depict the number of operations generated at A and generated at B. We call them version tuple.

[4] Or it is send to the server. The server then transforms and broadcasts the change to other application instances.

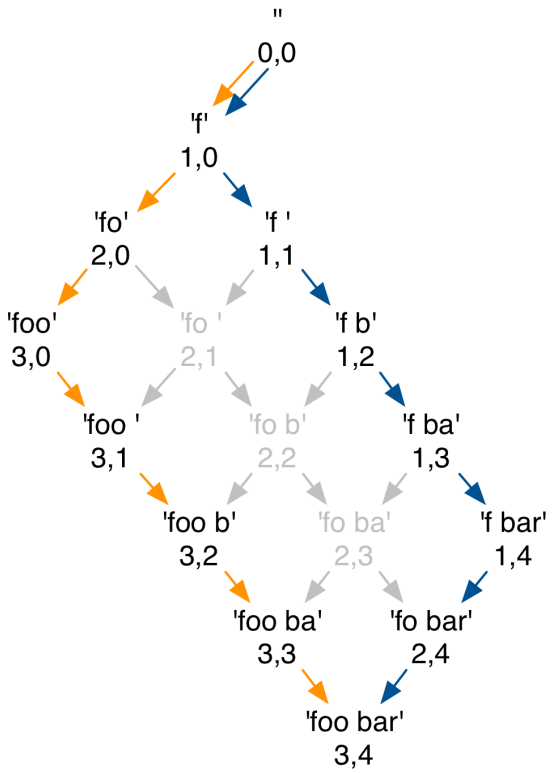

Fig. 3: The state space application instance A (orange) and B (blue) traverse when merging multiple operations.

First, A adds the character 'f'. This operation is submitted to B and applied there. Both are in the state $('f', 1, 0)$. Now A adds two more characters and ends up in the state $('foo', 3, 0)$. B does not receive these two new operations (for example because the network connection failed) and ends up in the state $('f\ bar', 1, 4)$. This means the state in A and B diverged by two, respectively 4, operations. To end up at a consistent state OT will merge all operations.

Assume that A receives the first operation of B $add(' ', 2)$, i.e. add space at position 2. The version tuple of the operation is $1, 1$. First, the conflicting operation is determined. It is $add('o', 2)$ because the last known common state had the version tuple $1, 0$. Then both operations are transformed: $transform(add('o', 2), add(' ', 2)) = add('o', 2), add(' ', 3)$. The result $add(' ', 3)$ is not applied at A but used as input for the next transformation: $transform(add('o', 3), add(' ', 3)) = add('o', 3), add(' ', 4)$. Now $add(' ', 4)$ can be applied at A to end up in state $('foo\ ', 3, 1)$.

During that process the transformations results not applied at A ($add('o', 2)$ and $add('o', 3)$) need to be stored. They are used to resolve the next conflict. The next operations A receives is $add('b', 3)$. The transformations $transform($

$add('o', 2), add('b', 3)) = add('o', 2), add('b', 4)$ and $transform(add('o', 3),$ $add('b', 4)) = add('o', 3), add('b', 5)$ are computed and $add('b', 5)$ is applied at A. The intermediate results $add('o', 2)$ and $add('o', 3)$ are stored again to resolve the next conflict. Generally when A is in state $(s_A, x + i, y)$ and last known state of B is (s_B, x, y), A needs to save i operations to successfully end up at a consistent state.

This process is done for every operation that A receives from B. B transforms the two operations from A ($add('o', 2)$ and $add('o', 3)$) analogously. Thus, both application instances end up in the state $('foo\ bar', 3, 4)$.

The complete description of the algorithm used to control the transformations is presented in [2].

Note that we have assumed that the operations received from remote application instance are ordered by their generation time. OT does not require such ordering since it is possible to order operations using their version tuple.

4 Using Operational Transformations in Web-based Multi-user Applications

We implemented a Web chat tool[5] that allows users to simultaneously modify a single text. Users can login with a Web browser and then immediately start interacting with other users online. The chat currently only supports basic text operations: Add, Delete, and Cursor move operations are supported. Extending the system with additional operations is possible and only requires implementing the operation classes and their transformations on client and server. The fundamental control flow of the application is the following. A user logs in and receives the current application state from the server. He can then make inputs (add and delete text). The inputs are events send to the text element in the browser. They are not only processed locally but each event is also converted to an operation object capturing. This operation is then send to the server for broadcasting it to other users. We are using a Comet-like communication scheme [9] with a basic protocol for broadcasting and receiving messages to and from channels. The broadcasting scheme was inspired by the Bayeux protocol [10].

How operations are interchanged and transformed is shown in figure 4. In the figure only two application instances A and B are connected to the server for simplicity, however, it is possible that an arbitrary number of application instances can be connected to the server. When A generates an operation it is applied locally and then send to the server. The server also keeps track of the application state of A in a separate state space called A'. The operation received from A is transformed and applied there using the approach described in section 3.2. The transformed operation is also send to the main server state (used for globally tracking the application state and for initially synchronizing new users) and to all state spaces from other connected clients. In the example there is only the state space B'. Again, the operation is transformed and applied there. The

[5] `http://lively-kernel.org/web-collab/webchatBox.xhtml`

resulting transformed operation is then send to B where it is transformed and applied once again. All transformation steps are necessary because concurrent operations can occur at any time in all application instances.

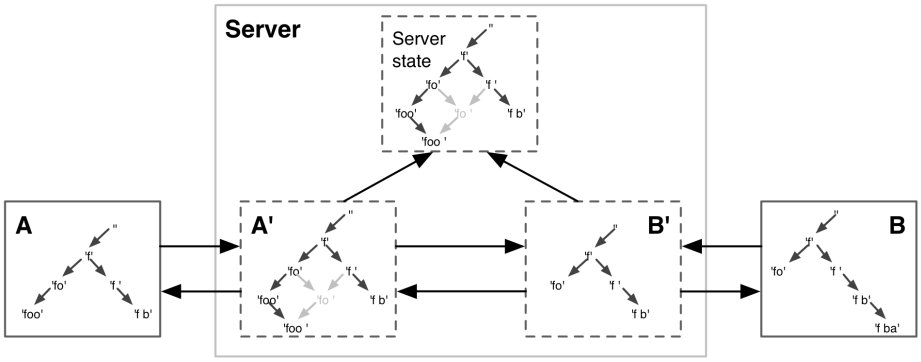

Fig. 4: Synchronization of operations.

5 Related Work

Operational Transformation is used in various editor applications. The Jupiter system [2] is a windowing system that uses OT for synchronizing text state but is also capable of synchronizing operations for graphical objects like a widget for drawing. The OT implementation used for the Web chat is closely related to the one presented in the Jupiter paper.

CoWord, CoPowerpoint, and CoMaya [5, 8] use OT to enable multi-user support in existing editor applications including synchronization support for operations on rich text, graphical widgets, and even 3D structures.

The text editor SubEthaEdit is a collaborative real-time editor designed for Mac OS X [11]. It uses OT for synchronizing text operations. SubEthaEdit supports only plain text modifications.

Google Wave is a Web-based communication tool with the goal to merge e-mail, instant messaging, wikis, and social network into one application [4]. In Google Wave users can create and modify documents called Wavelets. Wavelets contain rich text and other objects like embedded images and videos. Modifications of Wavelets are synchronized using OT. Google Wave extends the OT approach of the Jupiter system to make it more scalable. For example, multiple operations can be merged into one, making synchronization more efficient. Unlike other OT systems, application instances in Google Wave use confirmation messages from the server when the server has successfully processed operations send by the application instance. New operations are only send to the server

when a confirmation was received. This allows to simplify the server logic. Unlike the Web chat tool presented here, Google Wave only has to manage one state space at the server.

6 Summary

In this paper we presented the concept of Operational Transformation. OT is an approach to optimistically synchronize operations in a real-time multi-user system. OT transforms conflicting operations so that the execution of the transformed operations leads to a consistent state across all application instances. Application instances can be "offline" for an arbitrary amount of time. OT ensures that all changes happened at other application instances are iteratively transformed.

We furthermore presented an implementation of a Web chat tool that synchronizes its state using an OT implementation similar to the Jupiter system [2].

We plan to extend the system so that the chat-like features can be used to annotate worlds in the Lively Kernel [12]. This live documentation would work similar to wavelets presented in Google Wave [4]. We then plan to extend this system to allow sharing of Lively Kernel objects ("objects drawer"). In the end, we want multi-user support for modifying the full graphical environment of the Lively Kernel. For that we may base our implementation on the operations introduced by Webcards [13]. Webcards already allowed multi-user interaction based on these operations, however, no conflict resolution was implemented.

References

1. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. SIGMOD Rec. **18** (1989) 399–407
2. Nichols, D.A., Curtis, P., Dixon, M., Lamping, J.: High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System. In: UIST '95: Proceedings of the 8th annual ACM symposium on User interface and software technology, New York, NY, USA, ACM (1995) 111–120
3. Smith, D.A., Kay, A., Raab, A., Reed, D.P.: Croquet - a collaboration system architecture. Creating, Connecting and Collaborating through Computing, International Conference on **0** (2003) 2
4. Wang, D., Mah, A.: Google wave operational transformation. http://www.waveprotocol.org/whitepapers/operational-transform (2009)
5. Sun, C.: Designing Real-time Collaborative Editing Systems. Google TechTalk `http://www.youtube.com/watch?v=84zqbXUQIHc` (2008)
6. Sun, C., Ellis, C.: Operational transformation in real-time group rditors: Issues, algorithms, and achievements. In: CSCW '98: Proceedings of the 1998 ACM conference on Computer supported cooperative work, New York, NY, USA, ACM (1998) 59–68
7. Lamport, L.: Time, clocks, and the ordering of events in a distributed system. Commun. ACM **21** (1978) 558–565

8. Xia, S., Sun, D., Sun, C., Chen, D., Shen, H.: Leveraging single-user applications for multi-user collaboration: The coword approach. In: CSCW '04: Proceedings of the 2004 ACM conference on Computer supported cooperative work, New York, NY, USA, ACM (2004) 162–171
9. Bozdag, E., Mesbah, A., van Deursen, A.: A comparison of push and pull techniques for ajax. In: WSE '07: Proceedings of the 2007 9th IEEE International Workshop on Web Site Evolution, Washington, DC, USA, IEEE Computer Society (2007) 15–22
10. Russell, A., Wilkins, G., Davis, D.: Bayeux - a JSON Protocol for Publish/Subscribe Event Delivery. `http://svn.xantus.org/shortbus/trunk/bayeux/bayeux.html` (2007) As of Dec 29, 2009.
11. TheCodingMonkeys: SubEthaEdit. `http://www.subethaedit.net` (2003) As of Feb 10 2010.
12. Ingalls, D., Palacz, K., Uhler, S., Taivalsaari, A., Mikkonen, T.: The Lively Kernel A Self-supporting System on a Web Page. In Hirschfeld, R., Rose, K., eds.: S3. Volume 5146 of Lecture Notes in Computer Science., Springer (2008) 31–50
13. Dannert, J.: WebCards - Entwurf und Implementierung eines kollaborativen, graphischen Web-Entwicklungssystems für Endanwender. Master's thesis, Hasso Plattner Institut (2009)