An Analysis on Operational Transforms

CHRISTOFFER HIRSIMAA
and MARTIN NYCANDER

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2010

An Analysis on Operational Transforms

C H R I S T O F F E R H I R S I M A A
a n d M A R T I N N Y C A N D E R

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Mads Dam
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
hirsimaa_christoffer_OCH_nycander_martin_K10036.pdf

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Abstract

Google Wave attempts to redefine how we develop communication platforms by introducing *operational transforms* to the public. The concept of operational transforms was born about two decades ago in an attempt to expand the possibilities for live collaboration in documents. The subject has since been continuously researched and is now an established area of research.

This document summarizes, compares and describes the key algorithms in the development of OT systems. The systems documented are dOPT, adOPTed, GOT, GOTO, COT, Jupiter systems and Google. Finally it was discussed whether Google chose the most appropriate algorithm.

To determine if they chose the right approach we compared the best defined and promising COT algorithm to Google Wave's algorithm. The conclusion of this comparison is imperfect, but a basis for making a more complete evaluation has been documented.

Referat

En analys om “Operational Transforms”

Genom att introducera *“operational transforms”* för allmänheten utökade Google Wave sättet vi utvecklar kommunikationsplatformar på. Begreppet “operational transforms” föddes för omkring 20 år sedan i ett försök att utöka möjligheterna för samtida redigering av dokument. Området har sedan dess kontinuerligt forskats vidare på och är idag ett etablerat forskningsområde.

I det här dokumentet sammanfattas, jämförs och beskrivs de viktigaste algoritmerna i utvecklingen av OT-systemen. De system som dokumenterats är dOPT, adOPTed, GOT, GOTO, COT, Jupiter system och Google Wave. Mot slutet diskuteras huruvida Google Wave använder sig av rätt algoritm.

För att avgöra om Google Wave valde rätt strategi jämfördes den med COT-algoritmen, vilken såg ut att vara den mest lämpade. Slutsatsen av denna jämförelse är ofullständig, men underlag för att göra en mer komplett jämförelse finns dokumenterad.

Preface

This document was made in three overlapping states of production: research, analysis and documentation.

The research was equally shared among us both in search and reading papers with some joint discussion whether a essay or source could be used in our paper.

The analysis was made among the sources we found interesting which both of us read so that we could discuss them properly.

Documentation was distributed in the background chapter as such; Martin wrote dOPT, adOPTed and Google Wave while Christoffer wrote GOT(O), COT and the Jupiter System. In the rest of the document we mostly, but not entirely limited to, held the same break down as in the background. Since it was convenient that the one who wrote the background part of that algorithm should continue on the same domain.

Contents

1 Introduction **1**
- 1.1 Problem statement . . . 2
- 1.2 Definitions . . . 2

2 Background **5**
- 2.1 dOPT - distributed OPerational Transformation . . . 5
  - 2.1.1 Basic concepts . . . 5
  - 2.1.2 Problems . . . 7
  - 2.1.3 Strengths . . . 8
- 2.2 adOPTed . . . 8
  - 2.2.1 Changes in transformation . . . 9
  - 2.2.2 The algorithm . . . 9
  - 2.2.3 Problems . . . 9
  - 2.2.4 Strengths . . . 9
- 2.3 GOT & GOTO - Generic Operational Transformation . . . 10
  - 2.3.1 Introducing inclusion transformations counterpart . . . 10
  - 2.3.2 Problems . . . 11
  - 2.3.3 Strengths . . . 11
  - 2.3.4 GOT(O): GOT optimized . . . 11
  - 2.3.5 Remaining problems . . . 12
- 2.4 COT - Context-based OT . . . 12
  - 2.4.1 Context-based transformations . . . 12
  - 2.4.2 Problems . . . 14
  - 2.4.3 Strengths . . . 14
- 2.5 The Jupiter System . . . 14
  - 2.5.1 Introducing two-way communication . . . 14
  - 2.5.2 Main algorithm . . . 15
  - 2.5.3 Problems . . . 15
  - 2.5.4 Strengths . . . 16
- 2.6 Google Wave OT . . . 16
  - 2.6.1 Introducing the ACK-command . . . 16
  - 2.6.2 Bulking operations . . . 17

2.6.3 Practical properties 17
2.6.4 Problems 18

3 Analysis and Discussion **19**
3.1 What could be improved? 19
3.1.1 True intention preservation 19
3.1.2 Why Google Wave might fail to preserve intention 20
3.1.3 Perceived latency 21
3.2 Possible approaches for improvement 22
3.3 A solution: Incorporating COT into Google Wave? 22
3.3.1 Why COT might succeed in preserving intention 22
3.3.2 Efficiency 23
3.3.3 Is it worth it? 23

4 Conclusion **25**
4.1 Future work 26

References **27**

Chapter 1

Introduction

Google Wave is an attempt to renew the way we communicate today. The idea is to merge the many communication forms we use such as email, instant messaging, documents, blogs, forums and many more into a unified solution. Even more importantly; it makes the communication instant and collaborative, but with these communication features there is a problem.

You can edit the same object or word at the same time from more than one location; but how does Google Wave control what will happen? The content needs to be editable in a way that will not introduce bugs and it is kept up to date at all locations. In other words concurrency control, the mechanics of resolving conflicts. Over the years there have been a few general approaches to implementing conflict resolution and here follows a brief overview.

1. *Simple locking*, you may not modify a resource if someone else is editing it simultaneously.

2. *Optimistic locking*, you may always modify a resource. The system will tell you to resolve any occurring conflicts yourself.

3. ***Operational Transformation***, you can always modify a resource. The system will resolve any conflicts for you. This is what Google Wave uses.

Operational transformation (OT) was originally born from the need of consistency maintenance in collaborative text editors. In the span of over two decades OTs have gained new capabilities (such as undo operations and group undo) and have been applied to different applications ranging from HTML/XML editing, office tools and even 3D digital media design tools.

At its heart an OT system is the collection of algorithms to update every operation into a *version* that can be applied to another document in such a way that it can be *regarded* as *the original version of itself.*

Since OTs have many different applications it is crucial that the OTs themselves have a solid foundation. With this paper we aim to further investigate the subject with the following problem statement.

1.1 Problem statement

1. Which models for operational transforms exists? How do they work?
2. What is Google Wave's approach? How does it work?
3. How does Google Wave's approach compare to other techniques?

1.2 Definitions

Operation, O_x
: The event of adding, deleting or modifying an entity. An entity could be any kind of resource or object.

Transformation, $T(O_x)$
: The action of mapping a set of entities onto a different set of entities.

OT
: Acronym for Operation Transformation.

User
: A person who uses a software application.

Site
: A participant system, usually corresponds to a machine (however some machines may run multiple sites). There is one user per site.

Request
: A request is denoted as a request between sites of operation execution.

Groupware systems
: Computer-based systems which provides an interface to a shared environment wherein several users are performing a common task.[1] In this document groupware systems will refer to *real time* groupware systems.

Precedence property
A consistency property which ensures that the execution order is the same as the operations natural cause-effect order. In other words if an operation O_a causally precedes another operation O_b, then at every site O_a will be executed before O_b.

Causality violation
Violation of the *precedence property*.

Convergence property
A consistency property which ensures that copies of a shared resource are identical at all sites in state of rest (i.e. ensuring that all operations are executed).

Divergence
Term which defines the violation of the *convergence property*.

Intention preservation
A consistency property which ensures that the intention of an operation is preserved after a transformation. The intention of an operation, O, is the effect of applying O on the resource where the operation was generated.

Intention violation
Violation of the *intention preservation* property.

History Buffer (HB)
A log of previously executed operations on a site.

Inclusion Transformation (IT)
An OT function, $IT(O_a, O_b)$, which transforms O_a on O_b in such a way that O_b's impact is included.

Exclusion Transformation (ET)
An OT function, $ET(O_a, O_b)$, which transforms O_a on O_b in such a way that O_b's impact is excluded. In other words the inverse function of IT, $IT^{-1}(O_a, O_b) = ET(O_a, O_b)$.

Document State (DS)
Describes the state of the document at the defined site. The state is described by which operations that have been executed.

Context, $C(O_x)$
Describes in what context an operation is executed in, that is defined by which operations that precedes it. $C(O_x)$ corresponds to "context of O_x".

Widget
An interactive virtual tool that provides a single-purpose service to the user.

ACK
: Short for acknowledgement.

Chapter 2

Background

In the problem statement it was asked what established approaches exists to operational transformations. An attempt to answer this question has been documented in this section.

As far as we have seen there has been two main branches which differs greatly; the distributed systems (beginning with dOPT) and the centralized systems (beginning with Jupiter system). We will walk through seven of the major algorithms[1] that have been well defined over the years, in chronological order[2].

2.1 dOPT - distributed OPerational Transformation

2.1.1 Basic concepts

The dOPT algorithm was one of the first approaches to Operational Transforms. The basic idea is to take an operation executed in some past state and transform it so it can be applied in the current state. To do this it keeps a *linear history buffer*, a *transformation matrix* and associates every operation with a *priority* and a *state vector*.[1]

The data structures

A history buffer, or log, is a list of all previously executed operations. An operation is stored in the form $< i, s, o, p >$ where i is the originating site's identifier, s is

[1] Please note that not all approaches has been taken into considerations. We have merely listed those which we believe are the most significant approaches.

[2] Except for the Jupiter system, which makes more sense preceding Google Wave OT.

the originating site's *state vector*, o is the operation and p is the priority associated with o.

A transformation matrix, denoted as T, is what solves conflicting operations. T is a $m \times m$ matrix, where m is the number of supported operations in the groupware system. Each entry in the matrix is a function which transforms operations into other operations and to ensure convergence dOPT requires the Transformation Property 1, defined below.

Definition 1. Transformation Property 1 (TP1).[5;6;1]
For two independent operations O_a and O_b suppose that $O'_a = T(O_a, O_b)$ and $O'_b = T(O_b, O_a)$, the following statement must hold.

$$O_a \circ O'_b \equiv O_b \circ O'_a \tag{2.1}$$

where "$\equiv$" means *equivalence* in the sense that they will produce the same output state from a given input state.

The state vector is a way to time stamp operations. It is usually stored in the form: $v = (x_1, x_2, \ldots, x_n)$ where x_i is the number of requests by site i and n is number of sites in the system.

The priority value is used to determine which operation to transform onto which when they are operating on the same position. How it is calculated might vary, but[1] suggested to define it as a list composed of its largest predecessor's priority, concatenated with its own site ID.

The algorithm

The algorithm is divided into two parts: generating and receiving a request. The implementation of these parts are rather straightforward and documented below.

Generating a request As soon as an operation has been generated on the local site, the site executes the operation immediately, thus providing immediate feedback to the user. A request is then generated and sent to all other participant sites.

Receiving a request When a request is received at a site, examine the meta-information. It can then be determined what to do by comparing the site vector s_i with the requests' state vector s_r. We then have three possibilities:

1. $s_r = s_i$
   The sites are synchronized, execute the incoming operation immediately.

2. $s_r > s_i$
   The incoming operation can not be executed since there are operations that have been executed at the originating site but have not been executed locally. Queue the operation for later execution.

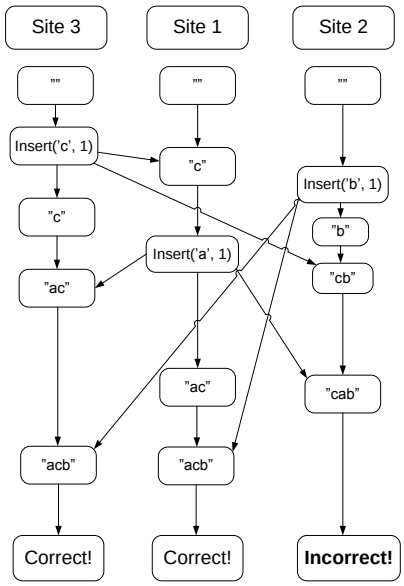
3. $s_r < s_i$
   Here an "old" request is received, which means that before the operation is executed, it must first be transformed. The history buffer is then examined for operations which has been executed locally but not at the originating site. Each such logged operation is then used to transform the incoming operation.

When an operation is executed it is added to the history buffer and the site's state vector is incremented.

2.1.2 Problems

The weakness of this algorithm, as shown in the example below, is the simple-minded priority calculation. It fails whenever an operation is concurrent with two or more dependent operations. [6]

Figure 2.1. An example illustrating the problem with the dOPT algorithm.



An example of this problem, known as the dOPT-puzzle can be seen in figure 2.1. In this specific example we can see that site 3 begins by insert 'c' to the document, it then receives `Insert('a', 1)` which it executes since it was executed after `Insert('c', 1)` at the sending site. To conclude, `Insert('b', 1)` is received and since it is a "past" operation it must be transformed against the two past operations. Site 1 performs similar and correct transformations.

The interesting part is how site 2 handles transformations. We can see that it begins by inserting 'b', 'c' then arrives and must be transformed but remains unmodified since 'c' has a higher priority, which simply is the same as the originating site ID. When 'a' arrives it will be transformed to `Insert('a', 2)` due to it's lower priority than 'b'. This results in divergence, breaking the *convergence property*.

2.1.3 Strengths

This algorithm has a few strengths, here follows a list of its features.

- Immediate feedback for local operations. [2]
- The algorithm is distributed and does not depend on a central server.
- Does not require locking of resources. [2]
- Relatively easy to implement due to its simple data structures.
- By fulfils the *precedence property*.
- Enforces the *Transformation property 1*.[2]

2.2 adOPTed

Because of the problems mentioned in section 2.1.2. A new algorithm was proposed which extends dOPT called the adOPTed algorithm. The most important changes are: a *multi-dimensional interaction model* for storing information and a *double recursive function* `Translate Request`.[6] It also redefines a few previously known terms.

The request r is redefined as (u, k, v, o) where u is the user generating the request, k is its *serial number*, v is the *state vector* and o is the operation itself. In comparison to dOPT we no longer need a priority value and in addition, the serial number is new. The serial number is simply initiated to one at the beginning of the session

[2] This is true for all OTs.

and then incremented by one for each request that the user makes. It is used along with the user ID to identify requests.[5]

2.2.1 Changes in transformation

The flaws in dOPT were in the way it transformed operations. adOPTed suggests a new transformation function, which also introduces a second transformation property. The second transformation property ensures that transformation of an operation along different paths will yield the same resulting operation.[5]

Definition 2. Transformation Property 2 (TP2).[5;6]
Given two operations O_a and O_b the following statement must hold, for any O:

$$T\left(T(O, O_a), T(O_b, O_a)\right) = T\left(T(O, O_b), T(O_a, O_b)\right) \tag{2.2}$$

If a transformation function in the transformation matrix holds for both TP1 and TP2 it should fulfil the *convergence property*.

2.2.2 The algorithm

The core algorithm works just the same as in dOPT (See section 2.1.1). The main difference is `Translate Request` function which takes a request r with a state vector v_r and transforms it recursively to a state vector v. It will only do this on the relevant requests. In practice, it is really just a more elegant rewrite of dOPT's implementation of handling requests.

2.2.3 Problems

The adOPTed algorithm is rather correct, except for the tricky *Intention Preservation* property. During this study no information could be found that this algorithm has been disproved.

2.2.4 Strengths

The algorithm inherits all the strengths of dOPT and adds a few more of its own.

- It is distributed and does not depend on a central server.
- Relatively easy to implement due to it's simple data structures.

- Fulfils the *convergence property*.
- Fulfils the *precedence property*.
- Enforces the *Transformation property 1*.
- Enforces the *Transformation property 2*.

2.3 GOT & GOTO - Generic Operational Transformation

2.3.1 Introducing inclusion transformations counterpart

While the adOPTed algorithm solved the dOPT puzzle by introducing an N-dimensional[3] interaction model (see section 2.2) the GOT algorithm introduces a whole new approach to the problem. GOT introduced the exclusion transformation (ET) as an inverse to the inclusion transformation (IT).

IT, as we have seen before in dOPT and adOPTed, adds the impact of another operation to an arbitrary ready operation. As opposed to the IT function, ET effectively removes the impact of the other operation against the operation to be executed. These pairs gives the ability to effectively traverse backwards in such a way that the operation to be executed is executed against the resource's history. In other words, GOT consist of an advanced *undo*→*do*→*redo* scheme which achieves convergence.[7]

In figure 2.2, both sites begin with the same state 'abcd'. Site 2 does O_1 `Delete('c',3)` and Site 1 does in order O_2 `Delete('a',1)` and O_3 `Insert('x',3)`.

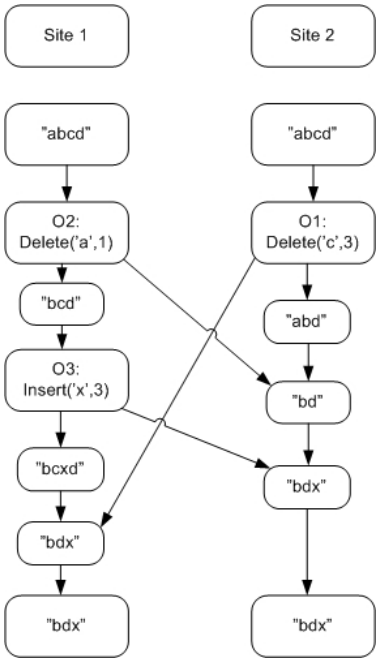
Here is what happens at Site 1 where the problem is [4] according to the GOT algorithm at:

Site 1:

1. O_2: is executed because it is a local operation.
2. O_3: is also executed directly for the same reason.
3. O_1: arrives and since O_2 and O_3 was executed already O_1 needs to be transformed. Foremost O_3 is applied with exclusion transformation against O_2 to get O_3'. O_3' is then exclusion transformed with O_1 to get O_1' in order to get the same execution state of this operation before applying inclusion transformation with O_2 and then O_3. Now all the operations have occurred in order backwards and forwards so that the execution states converges at all times.

[3] N denoted as number of sites.
[4] Further on how dOPT failed in section 2.5 for the same example.

Figure 2.2. A operational transform use case.



2.3.2 Problems

- Never theoretically proven correct for all scenarios. [2]

2.3.3 Strengths

- Fulfils the convergence property.
- Fulfils the precedence property.
- OT based undo, faster and more efficient undo based operations.
- Have been well tested as it has been implemented in several editing programs such as CoMaya, CoWord and more. [8]

2.3.4 GOT(O): GOT optimized

The GOT algorithm managed to solve all the problems however it was later optimized by adding the properties from adOPTed. TP1 and TP2 (see section 2.2) was included in order to make it even more effective by reducing the number of IT/ETs needed to execute every operation. [7]

2.3.5 Remaining problems

- Never theoretically proven correct for all scenarios.[2]

2.4 COT - Context-based OT

2.4.1 Context-based transformations

The COT algorithm no longer makes use of a history buffer (HB) when an operation is executed, instead it keeps track of the document state a operation have been executed in. So instead of a HB it has a context vector (see below for definition). This is the main difference since COT heavily relies on that every operation is to be executed on the same document state on all locations, if the operation is executed on the same document state at all sites the result can not diverge.

An operation is defined by a site identifier and a number given when it is generated. This operation in turn has a *context vector* that contains the operation context in terms of operations that precedes it. The operations inside the contexts vector in turn have their own site identifiers and defining numbers.

Definition 3. Context vector.[8]
Given an operation O, its context $C(O)$ can be represented by the following context vector $CV(O)$:

$$CV(O) = [(ns_0, ic_0), (ns_1, ic_1), ..., (ns_{N-1}, ic_{N-1})] \tag{2.3}$$

where for $0 \leq i \leq N-1$,

1. ns_i represents all original normal operations generated at site i, and
2. $ic_i = \left[(ns^0, is^0), (ns^1, is^1), ..., (ns^{k-1}, is^{k-1})\right]$ represents all inverse operations for undoing normal operations generated at site i, where (ns^j, is^j), $0 \leq j < k$, represents an inverse group with is^j inverses related to the normal operation with sequence number ns^j.

This definition, which have undergone three revisions so far, is very raw and have the potential to be optimized.[8;6] It will surely be revised further by future papers.

By changing to a context vector Chengzheng Sun et al.[8] introduced a new set of rules called *Context-based Conditions* (CCs) to capture the essential requirements for correct operation execution and transformation. The following is a short version

of the conditions and foundation for the COT algorithm, the reader is referred to his latest paper for a complete description of the here given conditions:

- **CC1**: Given a original *operation* O and a *document state* DS, where $O \notin DS$, O can be transformed for execution on DS only if $C(O) \subseteq DS$.
- **CC2**: Given an original operation O and a document state DS, where $O \notin DS$ and $C(O) \subseteq DS, DS - C(O)^2$ is the set of operations that O must be transformed against before being executed on DS.
- **CC3**: Given any operation O and a state DS, O can be executed on DS only if $C(O) = DS$.
- **CC4**: Given an original operation O_x and a operation O of any type, where $O_x \notin C(O)$, O_x can be transformed to the context of O only if $C(O_x) \subseteq C(O)$.
- **CC5**: Given an original operation O_x and a operation O of any type, where $O_x \notin C(O)$ and $C(O_x) \subseteq C(O)$, $C(O) - C(O_x)$ is the set of operations that O_x must be transformed against before being IT-transformed with O.
- **CC6**: Given two operations O_a and O_b, they can be IT-transformed with each other $IT(O_a, O_b)$ or $IT(O_b, O_a)$, only if $C(O_a) = C(O_b)$.

CC1 and CC4 defines what is needed to ensure the correct ordering of operation execution and transformation, CC2 and CC5 are required for determining the correct transformation target operations and CC3 and CC6 ensures correct operation execution and transformation.[8]

Due to these conditions and the context vector the COT algorithm is optimized to the extent that it does not even have a need for the ET transformations. This is made possible because of the context vector that enables a better use of ITs.

In order to give an example how the basic COT algorithm works consider the example in figure 2.2.

Site 2: [5]

1. O_1: After the generation of the operation it naturally has the same context as the local document state and is executed as is.
2. O_2: When O_2 arrives its context difference to the document state is O_1 and therefore is transformed by $O_2' = IT(O_2, O_1)$ and O_2' is then executed.

[5] Does not consider any optimisations.

3. O_3: Primary O_3 arrives with $C(O_3) = \{O_2\}$ the difference in contexts is O_1 compared to document state. Secondly O_3 is compared to O_1 and the resulting difference of contexts is O_2. Because $C(O_2) = C(O_1)$ the first transformation, $IT(O_1, O_2)$, will include O_2 in O_1 and the result O_1' is returned. Now $C(O_1') = C(O_3)$ so $O_3' = IT(O_3, O_1')$ includes O_1 (dOPT puzzle resolved here, see section 2.1.2). The resulting context of O_3' is $\{O_1, O_2\}$ and now $C(O_3') = DS$ and is therefore executed.

2.4.2 Problems

The COT algorithm may require more memory than earlier algorithms due to the nature of the context vectors as it is defined now.[2] Chengzheng Sun et al. have pointed out that many optimisations from previous algorithms can be applied to COT as well.[8]

2.4.3 Strengths

- Fulfils the convergence property.
- Fulfils the precedence property.
- Platform capable of more than text editing.
- Has been well tested as it has been implemented in several editing programs such as CoMaya, CoWord and more.[8]

2.5 The Jupiter System

Jupiter Collaboration System was developed as a multi-user, multimedia virtual platform with its foundation in the dOPT algorithm (see section 2.1). Jupiter was intended to be easy to develop new widgets for and stand as a evolving platform.

2.5.1 Introducing two-way communication

The Jupiter system forces all communication through a central server and thus bypasses the problems of the dOPT algorithm. This is because there is never more than two sites communicating.[3;7]

2.5.2 Main algorithm

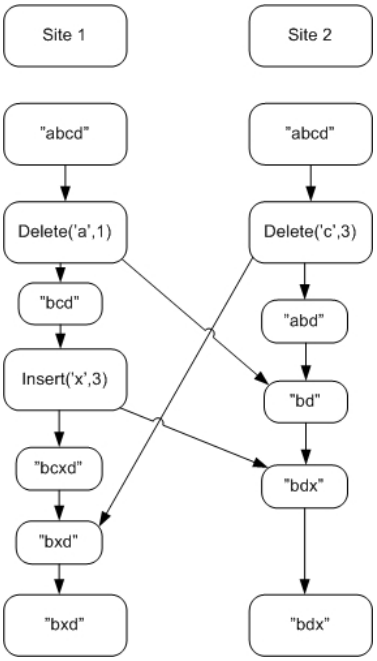
The dOPT algorithm seemed to work perfectly between two sites but when the number of sites increased it became apparent that the dOPT algorithm did not satisfy the convergence property. By making the communication pass through a server it enabled them to make the dOPT algorithm work for more sites. This is because there is never more than two sites communicating; every site is strictly limited to one server.[3]

For every operation executed on the server the operation is broadcasted to all the clients. Meanwhile a client only send the operation to the server.[3]

2.5.3 Problems

It was later realized the dOPT algorithm was not even sufficiently correct with only two users. Due to the nature of the dOPT algorithm it could not correctly handle the situation in figure 2.3 if the operations on site 1 are dependent on the operation on site 2. The resulting documents on both site will not be the same.

Figure 2.3. An example of when dOPT doesn't work correctly between two users.



Example in figure 2.3 Let the initial string be "abcd" across both sites. Site 2 deletes the character "c" at position 3, `Delete('c',1)`. Site 1 first deletes "a" at position 1, `Delete('a',1)`, and then inserts "x" at position 3, `Insert('x',3)`.

Further more by the dOPT algorithm, when site 2 receives `Delete('a',1)` no transformation is needed and neither when `Insert('x',3)` arrives. But when `Delete('c',3)` arrives at site 1 it will be first transformed into `Delete('c',2)` and after that no more transformation is needed according to dOPT.

Further problems are the same as in the dOPT algorithm in section 2.1.

2.5.4 Strengths

- Centralized solution that mended the need for a better algorithm.
- Platform capable of more than text editing, in theory but not proven in practice.

2.6 Google Wave OT

When Google Wave was developed it was based on the Jupiter system (see section 2.5).[9] Although Google adapted it to be more useful in practice than in theory since their concern was not of the academical faction, as many of the earlier OT techniques. The measures taken during development has been documented in this section.

2.6.1 Introducing the ACK-command

To reduce the complexity of server implementation the Google Wave team made the addition of requiring the server to acknowledge an operation before another one is submitted. This means that the client can only send one operation in between ACKs. This restriction has both benefits and drawbacks.

The server benefits from this since the client can perform a lot of the OT work while it waits for the server to acknowledge previous operations. Instead of `Insert('t', 1)` it can accumulate operations over time to `Insert('text', 1)` which saves the server a lot of work. The server is also able to store less data per client, more specifically it only needs to store a linear history buffer per client since the order of operations is guaranteed server side.

The drawback however is that the latency is increased since operations can not be streamed directly to all other participants, but have to be relayed through a server. In effect of this the interface will, at best, see accumulated batch operations arrive in intervals from the server.

2.6.2 Bulking operations

While a client is waiting for an acknowledgement from the server it bulks the local operations into one operation. It does this to lessen the strain on the server. The composition operation is designed to fulfil two requirements.

Definition 4. Composition requirement 1.[9]
The composition $O_b \circ O_a$ has the property of

$$(O_b \circ O_a)(d) = O_b(O_a(d)) \tag{2.4}$$

where d is the document.

Definition 5. Composition requirement 2.[9]

$$\begin{aligned} T(O_a, O_x) = (O'_a, O'_x),\ T(O_b, O'_x) = (O'_b, O''_x) \\ \Rightarrow T(O_b \circ O_a, O_x) = (O'_b \circ O'_a, O''_x) \end{aligned} \tag{2.5}$$

and that

$$\begin{aligned} T(O_x, O_a) = (O'_x, O'_a),\ T(O'_x, O_b) = (O'_x, O'_b) \\ \Rightarrow T(O_x, O_b \circ O_a) = (O''_x, O'_b \circ O'_a) \end{aligned} \tag{2.6}$$

The composition operations must fulfil these requirements to perform correctly. According to David Wang and Alex Mah[9] the performance of resolving accumulated concurrent unacknowledged operations is cut down to $O(n \log n + m \log m)$ from traditional OT's $O(m \cdot n)$ where n is the number of unsynchronized client operations and m is the total size of the server operations.

2.6.3 Practical properties

Since this system was to be deployed to the entire Internet community, it had to be failure resistant and uphold a certain quality of service. Therefore the ability to recover from a crash or communication failure was added. [6]

To prevent corruption in communication checksums of operations were also added to the mix. With all operations and acknowledgements a checksum of the entire document is added. Errors in the transformations can then be detected, and if there is an error a fresh copy can be copied from the server to the faulty client to resolve the problem. This ensures graceful error handling. If Google Wave's OT implementation would have flaws, they can be repaired live and recorded for later reparation. Thus not affecting the application in a production environment.

[6] Further details are buried in the source code of Google Wave.

2.6.4 Problems

The Google Wave approach to OT's performs correctly. However its correctness can not be taken for certain, since it is very hard to define what the users intention actually is. This problem will be further discussed in section 3.1.1.

Chapter 3

Analysis and Discussion

The goal of this chapter is to extend problem statement two and three by asking the questions "Is Google Wave right in using dOPT? Would it benefit from using a more modern algorithm?". An attempt to evaluate that thesis by brief analysis and discussion is documented in this chapter.

3.1 What could be improved?

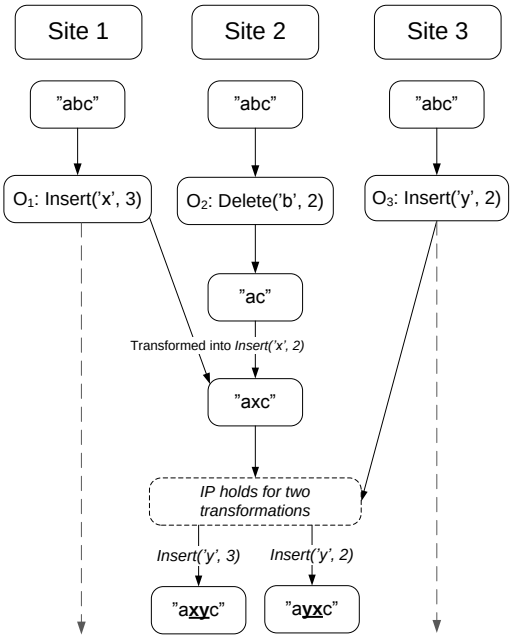
One could argue that Google Wave is fine as it is. In this section we will try to motivate why Wave might profit from changing algorithm.

3.1.1 True intention preservation

The definition of intention preservation (IP) is: *by executing an operation, O, on a remote document, D_r, it must achieve the same effect as the intention of O on the original document D_o.*[6] The ambiguity of this definition enables it to apply to many different types of media, but it also causes problems in verifying whether or not an algorithm really preserved the operational intentions.

Consider the example in figure 3.1. All sites begin by generating three operations on the same document, site 2 begins by removing 'b' between 'a' and 'c'. O_1 then arrives and is transformed into O_1' to preserve the intention of 'x' occurring after 'b' and before 'c', 'b' has been removed, but the intention of occurring before 'c' is upheld by transformation. O_3 then arrives to insert 'y' after 'a' and before 'c' and an inconsistency problem occurs. The problem is that however O_3 is transformed the result can be regarded as equally intention preserving, as 'y' will have been inserted after 'a' and before 'c' in both 'axyc' and 'ayxc'.[6;2]

Figure 3.1. An example illustrating the problem with the definition of intention preservation.

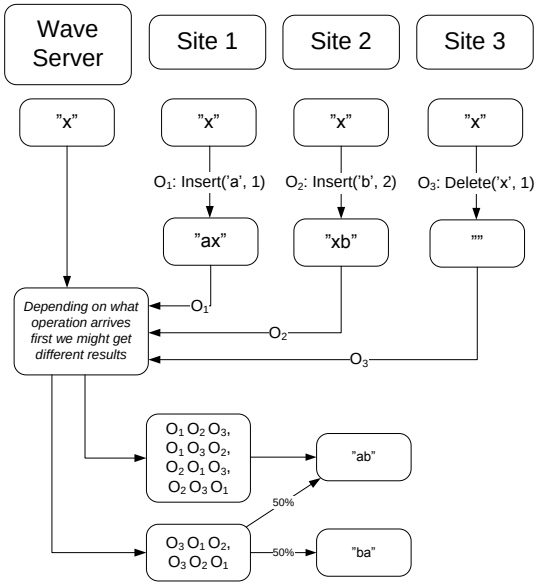


3.1.2 Why Google Wave might fail to preserve intention

Since Google Wave is based on the Jupiter system, which in turn is based on the dOPT algorithm, it inherits some flaws which have been fixed by two decades of evolution in the dOPT family tree. Google Wave solves most of these flaws by its ACK command and by strictly limiting the number of participants in communication to two. However, there is one flaw that might have slipped through, namely intention preservation.

Consider the example in figure 3.2. Three Wave clients concurrently execute different operations on the same document, depending on what operation arrives first at the wave server we get different results. If the server executes the delete operation first, the two insert operations will be transformed to insert different characters at the same position in the document. The dOPT algorithm looked at the site identifier to determine which operation should have priority over which, Google Wave simply uses the order they arrived. This means that Google Wave will not preserve the intention $\frac{2}{6} \cdot \frac{1}{2} = \frac{1}{6} \approx 17\%$ of the time in all concurrent scenarios with conflicts involving three participants.[4]

Figure 3.2. An example illustrating the problem with intention preservation in Google Wave.



One could argue that since the users were not aware of each others actions, and that the overall intention does not matter. But remember, 'a' could in theory be an entire chapter since Google Wave bulks operations together while it is waiting for the ACK command. This results in 17% of the time two chapters submitted to a Wave at the same time will occur in the wrong order, and each author should know what chapter is *intended* to precede and to follow the current chapter.

3.1.3 Perceived latency

Because of Wave's ACK command there is an increase in the users perceived latency. The network latency is probably improved by the ACK command, but since the user will see external operations arriving in bulk it will be perceived as sluggish.

There is room for improvement here. A quick fix could be to animate bulked operations in the user interface, but this has problems in itself. For example; What would happen if a user tampers with the operation in the middle of the animation?

3.2 Possible approaches for improvement

One could argue that Google could patch or rewrite the source code from the beginning of Google Wave in order to perfect it. In contrast, this study questions if they started with the right algorithm to begin with; and as such Google Wave will be compared to a more modern algorithm.

Naturally a good choice of algorithm to investigate is the adOPTed algorithm since it is a redefinition of dOPT that Google Wave uses. It is also a very good choice except that COT have a stronger algorithm itself that even can handle any undo compared to adOPTeds chronological undo. We chose COT over GOT and GOTO since COT is a more efficient algorithm[8].

It is because of COTs stronger potential that we have chosen to compare it with Google Waves own algorithm.

3.3 A solution: Incorporating COT into Google Wave?

Even though the COT algorithm is more complex at its heart and have theoretical proofs to back it up does not indicate it is better for the task at hand for Google's project. Wave puts great emphasis on being able to handle many different situations and to be highly scalable as it was meant to be a unified communication platform.

To begin with the COT algorithm works perfectly with the same centralized solution as Wave's and there are several reasons to keep it that way. One of the most important reasons is that you will need the centralized server for new users to be able to join Waves when all others are offline.

What COT might provide for Google Wave is better intention preservation, following is a brief explanation of why.

3.3.1 Why COT might succeed in preserving intention

Intention preservation was never the main focus of the COT algorithm from the start and was therefore not discussed in the reports about the algorithm itself.

However; considering how the algorithm works it, to the best of our knowledge, solves the intention preservation problem in the earlier example in figure 3.2. COT achieves this because the algorithm is not based on when the operation arrives at different places but in what context the original operation was executed in. Therefore at each site when O_x arrives a comparison will be made with O_x's context vector and the site's own document state.

3.3.2 Efficiency

One of the important things with this solution is that the OT algorithms must be very efficiently implemented, especially on the server-side. Whether or not COT is more efficient can be viewed from two main points, the amount of communication traffic needed and the amount of processing needed for the algorithm itself.

Networking latency

Considering communication efficiency, the COT algorithm should be less effective since every operation is sent directly and because of that the algorithm will generate more TCP packages than Google Wave's solution. This is due to the bulk of operations generated while waiting for ACK commands from the server in Google Wave. However the possible increase in traffic may provide a lower perceived latency between sites.

Processing resources

With smart caching of transformed operations the COT algorithm will not require much more processing than dOPT. Neither of the algorithms will have to transform more than one operation against another.

Although; the COT algorithm requires a greater amount of memory in order to achieve the significant advantage it has over other algorithms. Garbage collecting algorithms and optimisations have been proposed to reduce the amount of memory needed[8].

3.3.3 Is it worth it?

Google Wave might achieve better theoretical correctness by updating its core algorithm, may it not be COT but any modern OT algorithm. Wave will in practice be correct in almost every case. When it is incorrect, it will not be observed by the user as an error. Thus the theoretical correctness is not of Google's concern.

What might be of its concern is to achieve a lower perceived latency and in turn a more responsive user interface experience. COT might be able to help with that by removing the need for an ACK command. The effect of that is that fewer operations will be bulked together and other sites operations will be received in shorter intervals and provide a smoother user experience.

Chapter 4

Conclusion

We have summarized a large number of operational transform algorithms over a span of twenty years with their achievements, discoveries, adherent problems and contributions to this field of science. This technology is still in its infancy in the software development business with Google recently partaking with its own Google Wave. We set out to learn about operational transforms in order to review Google Wave and compare it to other algorithms. We concluded that the COT algorithm by Chengzheng Sun et al. was the most appropriate algorithm to be compared with Google Wave's. This is because COT is the latest that have been implemented in a wide spread of software and have been well defined in reports by its creator(s).

Google Wave's foundation lies in the early dOPT algorithm and the Jupiter system which explains some decisions made by Google. However, because of the problems inherited from the dOPT algorithm Google Wave seems forced to add the ACK command and use the centralized solution that also Jupiter used, in order to achieve convergence and to enforce the precedence property. It would also be interesting to compare how Google Wave implements undo compared to COT but the details of the implementation in Google Wave was not found. Also, in theory Google Wave actually fails to preserve the users real intention as defined in 3.1.2, however the consequences of it is not noticeable by its users.

In conclusion we managed to compare the two algorithms to some extent but in order to give a more complete answer, if Google chose the best approach to the problem, further testing is needed and more importantly an increased knowledge of Google Wave. The only apparent way to appreciate its efficiency is to fully understand Google Wave's source code and benchmark it against other algorithms. The documentation given for Google Wave was not enough to give a complete answer, however,d we managed to pinpoint more accurately what to consider in future testing and research as can be found in the following section "Future work".

4.1 Future work

The scope of this paper far from exhausts the narrow field of modern implementations of Operational Transforms. The time span for this paper was not enough to fully do so. Following is a list of possible future research projects, which could be performed as an extension to this project.

- **Benchmarking Google Wave and COT**
  Benchmarking must be performed to really know if COT is suitable for Google Wave. Suggested approaches would be to insert a COT implementation into the sample source code from the Wave Federation and then to run a number of benchmarking tests.
- **COT intention preservation testing**
  To — either by formal proof or by implementing and testing COT — investigate if COT has true intention preservation properties.
- **Google Wave's IP failure in practice**
  By using source code provided by the Wave Federation implementing a test to prove that Waves OT fails to preserve true intention in practice.
- **Research undo**
  The mystery of how undo is implemented in Google Wave has not been covered by this paper. It could be interesting to investigate how it relates to other OT approaches to undo, and to see if this makes a change of core algorithm in Wave more complicated or even more necessary.
- **More Google Wave alterations**
  By further analysis investigate if there are other OT algorithms which could be incorporated into Wave to achieve better performance and/or correctness. Perhaps look into the new admissibility-based operational transformation framework by Li and Li[2], which was not covered by this report.

References

[1] CA Ellis and SJ Gibbs. Concurrency control in groupware systems. *ACM SIGMOD Record*, 18(2):399–407, 1989.

[2] Du Li and Rui Li. An admissibility-based operational transformation framework for collaborative editing systems. *Comput. Supported Coop. Work*, 19(1):1–43, 2010. ISSN 0925-9724. doi: http://dx.doi.org/10.1007/s10606-009-9103-1.

[3] David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping. 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*, pages 111–120, New York, NY, USA, 1995. ACM. ISBN 0-89791-709-X. doi: http://doi.acm.org/10.1145/215585.215706.

[4] Daniel Paull. Google wave: Intention preservation, branching, merging and tp2. http://www.thinkbottomup.com.au/site/blog/Google_Wave_Intention_Preservation_Branching_Merging_and_TP2, March 2010. [Online; accessed 22 April 2010.].

[5] M. Ressel, D. Nitsche-Ruhland, and R. Gunzenhäuser. An integrating, transformation-oriented approach to concurrency control and undo in group editors. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, pages 288–297. ACM New York, NY, USA, 1996.

[6] C. Sun and C. Ellis. Operational transformation in real-time group editors: issues, algorithms, and achievements. In *Proceedings of the 1998 ACM conference on Computer supported cooperative work*, pages 59–68. ACM New York, NY, USA, 1998.

[7] Chengzheng Sun, Xiaohua Jia, Yanchun Zhang, Yun Yang, and David Chen. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, 5(1): 63–108, 1998. ISSN 1073-0516. doi: http://doi.acm.org/10.1145/274444.274447.

[8] David Sun and Chengzheng Sun. Context-based operational transformation in distributed collaborative editing systems. *IEEE Trans. Parallel Distrib. Syst.*, 20

(10):1454–1470, 2009. ISSN 1045-9219. doi: http://dx.doi.org/10.1109/TPDS.2008.240.

[9] D. Wang and A. Mah. Google wave operational transformation. http://www.waveprotocol.org/whitepapers/operational-transform. [Online; accessed 14 April 2010.].