MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Real-time collaboration in Komodo

MASTER'S THESIS

Bc. Matúš Makový

Brno, Spring 2015

Declaration

Hereby I declare, that this paper is my original authorial work, which I have worked out by my own. All sources, references and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Bc. Matúš Makový

Advisor: RNDr. Filip Nguyen

Acknowledgement

Abstract

Keywords

Contents

1 Introduction

Many software solutions enable people to create new things in a better and faster way. In most cases the resulting product should be so complex that one person is not enuogh for the successful and fast creation. Creators try to collaborate to achive a common goal. Among other opportunities and possibilities are collaboration and sharing the greatest benefits of the Internet.

We can identify 2 types of collaboration over the Internet, non real-time collaboration and real-time collaboration. At the beginning, as the Internet didn't have such capacity, people tried to use it just for sharing their drafts of work and sending them to each other, this type of collaboration is called non real-time. In non real-time collaboration users work on separate copies of a project and then need to merge their changes into one final project. In other words, they had to find differences between their drafts and reflect them to each other's version. It doesn't offer such flexibility as real-time collaboraion and also it had many limitations, for example 2 people could not edit the same file in a project without having to resolve confilicts manualy when they tried to merge their work with collaborator's version

Examples of non real-time collaboration could be Revision control (Git, SVN). With the development of the Internet came a reasonable solution called real-time collaboration. Using this principle, author can see what his collaborator is doing in real-time and the manual synchronization or manual confilict resolution is not necessary. Information technologies take care of this synchronization and conflict resolution for the users.

In the theoretical part of this thesis, author deals with principles of real-time collaboration and describes some of the techniques used to implement real-time collaboration in software over the Internet. There is also a comparison of these techniques from various aspects.

The practical part of this thesis deals with JBoss Data Virtualization and Komodo software as a new version of Teiid Designer developed by Red Hat that should use real-time collaboration in its upcomming release. Author recommends best technique for this authoring software regarding the requirements of data models and finds a suitable implementation in Java programming language.

2 Real-time collaboration

This chapter covers the basic overview of collaboration in general, description of real-time collaboration and description of difficulties with its implementation over the Internet. Last sections of this chapter describe three techniques used for implementation over the Internet and its properties. Techniques described in this chapter are: Operational Transformation, Differential Synchronization and Commutative replicated data types.

2.1 Basic overview

Collaboration, as defined by English dictionary, is an act of working with another or others on a joint project. Authoring Systems in IT that support collaboration are called Groupware.

Groupware systems are computer-based systems that support two or more users engaged in a common task, and that provide an interface to a shared environment. These systems frequently require fine-granularity sharing of data and fast response times.[1]

There are many techniques used in groupware systems that are not suitable for real-time collaboration, for example, Locking or Single Active Participant technique. Fundamental principle of Locking and Single Active Participant is to lock data when it is being modified by someone. This principle is adopted for example by Microsoft, when users try to edit shared document.

When using real-time collaboration, authorig software creates an illusion that users are working on one common copy of a document online. There is no requirement to commit changes to some kind of shared repository and no need for a user to resolve conflicts. Changes should be reflected and saved immediately. Examples of real-time collaborative editors that we recognize today are Google Docs, Ether-

pad and already terminated Google Wave.

Software engineer has different options for implementetation of real-time collaboration in software solution.

Requirements for a good technique are:

- speed
- latency tolerantion
- low data transfer
- consistency maintanance
- good conflict resolution

The speed requirement means that changes made on one side of the collaboration process need to be reflected to the other sides as soon as possible and vice versa. Changes should be sent to other collaborators as soon as they are done. If a collaboration technique is fast enough it is much easier to satisfy other requirements on this technique. Fast enough technique is able to maintain good consistency. The system's response time is the time necessary for the actions of one user to be reflected by their own interface; the notification time is the time necessary for one user's actions to be propagated to the remaining users' interfaces.[1] Response time and notification time should be as short as possible.

One of the problems of real-time collabarative editing could be the latency of network. Implemented technique should be able to tolerate the latency of internet connections, because collaborators could be in very distinct parts of the world. It should be able to reconstruct the right order of operations because data sent over the Internet don't necessary come in right order and the order of the operations is ery important to maintain consistency.

Low data transfer requirement is also very important. Different sides of the collaboration should send as few data as possible. Transfered data should only describe change that has been done on one side of collaboration, it is not necessary to transfer the whole project. The less data is needed to transfer the faster the whole protocol can be.

Consistency maintanance is necesssary for the success, it has to be ensured, that users on both sides are looking and the same version of a document regardless of number and coplexity of operations done on both sides of the collaboration. Lack of consistency could cause other problems and chaos in the document versions. According to [7] there are three problems encountered when trying to achive consistency maintenance and they correspond to the properties of CCI consistency model proposed in [6]

1. **Casuality Preservation** - operation O_1 causually precedes operation O_2 if O_1 occured localy before O_2. The problem is to execute operations in right order on all sites.

2. **User Intension Preservation** - technique must preserve user's intension in the context of state, in which the operation was executed. This problem is in strog relation with confilict resolution requirement and occures when it is not posible to determine if O_x casually precedes O_y or O_y casually precedes O_x.

3. **Convergence** - when same operations have been applied on every site, the documents are identical.

Because the real-time collaboration is asychnronious there rises a problem of concurency. This means, that changes can happen at the same time and in the same sections of project. Good conflict resolution requirement is present because of concurency. Implementation techniques have to be able to identify and resolve a conflict when users are editing the same part of the project.

The concurrency control algorithms are separeted in two classes: pessimistic and optimistic.

Pessimistic algorithms require communication with other sites or with a central coordinator before making a change to data.[**?**] One example of pessimistic algorithm can be already mentioned Locking or Single Active Participant technique.

Optimistic concurrency control, on the other hand, requires no communication before applying changes locally. The party making a change applies it immediately, then informs the other parties of the action. If more than one participant makes a change at the same time, a conflict resolution algorithm creates compensating changes to move everyone to the same final state.[4]

This implies that optimistic algorithms are better solution for Internet, because the latency can not be guaranteed.

2.2 Operational Transformation

Operational transformation (OT) is optimistic concurrency control algorithm used for real-time collaborative editing over the Internet.

It was first introduced in paper Concurency Control in Groupware Systems[1] in 1989, together with The Distrubeted Optional Transformation (dOPT) Algorithm. As the paper describes it, algorithm has a number of properties which make it suitable for groupware. First, operations are performed immediately on their originating site, thus responsiveness is good. Secondly, locks are not necessary so all data remains accessible to group members. Finally, the algorithm is fully distributed, and resilient to site failure.

This first algorithm was able to process only plain text. Later some problems with correctnes were discovered and resolved in following works. Over the years many other algorithms implementing

OT have been published. Algorithms used today are able to process also XML and other formats.

OT was used also in Jupiter Collaboration System in 1995 [4]. Jupiter's two-way algorithm is derived from the dOPT algorithm used by Grove in [1]. This algorithm is one of the imporovments of dOPT.

Today is Google one of the most common implementators of this protocol. The main application was in Google Wave project, which is now terminated , although Operational Transformation found application also in Google Docs project. According to [5] Google is one of the bigest inovators of this approach an it's biggest contribution is the idea of operation compositon, which is described later in this section.

The approches differ also in the architecture, dOPT doesn't involve server in its design, but the recent algorithms use a central server that maintains its version of the document, coordinates the communication and broadcasts operations.

This section will explain basic principles of Operational transformation using Google version of the control algorithm, introduce some improvements added by Google and then point out the differences between the presented version and other versions like dOTP, Jupiter and TIPS.

The basic idea is that data are replicated on every machine and the only information that is sent over the Internet are the operations. Data replication ensures good responsivnes in high latency enviroments. Wave's addition to OT are also annotations. An annotation is some meta-data associated with an item range, i.e., a start position and an end position. This is particularly useful for describing text formatting and spelling suggestions, as it does not unecessarily complicate the underlying structured document format.[8]

The main building block of this approach, as the name suggests, is

an operation. Here are operations defined for use in Google Wave[8]:

- retain() - move cursor
- insertCharacters() - inserts texts
- insertElementStart() - inserts starting tag
- insertElementEnd() - inserts end tag
- deleteCharacters() - deletes text
- deleteElementStart() - deletes starting tag
- deleteElementEnd() - deletes ending tag
- replaceAttributes() - replaces attributes in tag
- updateAttributes() - updates attributes in tah
- annotationBoundary() - describes changes in annotations

The operation is executed locally and then sent over the network to the server or to other peers, to execute the operation on their version of the document. Every object that can be changed in the document has its index for identification. Google Wave uses XML as format for storing information so as an object is considered a character in text and also a starting and terminating tag in XML structure (characters of tags are not considered elements).

The second major part of this technique is a transformation function, that transforms operations. Operations which are sent over the network and recieved by server can not be executed directly on the server's version of the document, because the index of the desired element could be different in the local copy and in the server copy of the document and this can cause inconsistency and violation of User Intension Preservation.

See Figure 2.1 with example for better explanation. For the purpose of this example, we will assume that we have one client that sends operations done on his local copy and a server that recieves operations from all peers and maintains server version of the document.

Figure 2.1: Example Operational Transformation

Both of them start with text 'TAX' in their document. Pointer starts at index 0 and operation retain() shifts the pointer by number of indexes given as parameter. Client wants to add character 'E' so he executes operations:

```
retain(3);
insertCharacters('E');
```

In the meantime server recieved operation from other peer, that has inserted character 'S', so server executes operations:

```
retain(3);
insertCharacters('S');
```

Server and client exchange the operations. Different order and difference in indexes on both sides of the collaboration result in incosistent document state. String in first document is 'TAXSE' and string in sec-

ond document is 'TAXES'. In order to preserve user's intension and to reach correct result we have to transform this operations. The correct result after inserting letters 'E' and 'S' should be 'TAXES'. Server has correct version of the document, but client doesn't so we have to transform the operation sent by server to look like this:

```
retain(4);
insertCharacters('S');
```

This transformation was helpful, but it is not enough. Now the client operation retains 3 characters, inserts new character and leaves cursor in front of last letter and the server operation retains 4 characters inserts character and leaves cursor behind the whole word. Client and server should end up in the same state. So it is necessary to transform also the client operation:

```
retain(3);
insertCharacters('E');
retain(1);
```

After this change the result of applying operations on both sides will be correct.

More gernerally, if we have two operations O_1 and O_2 we need a transformation function *trasform*, such that:

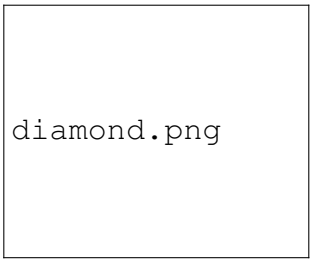
$$transform(O_1, O_2) = (O_1', O_2') \quad where \quad O_2' \circ O_1 \equiv O_1' \circ O_2$$

In other words, the function takes two operations and transforms them against each other and the output of the function are two transformed operations, such that, if we have same strings on both sides a we apply O_1 and O_2' on one side, and O_2 and O_1' on the other side, the resulting string is the same.

According to [3]. If an OT system is to support particular functionality, then it must be able to support certain transformation properties. For group editing and consistency maintenance, the system

must support a transformation function known as Inclusion Transformation. For group undo, where the effect of a previously executed operation is un-done at all sites, and all operations executed after it are all re-transformed, the system must support another transformation function known as Exclusion Transformation. Inclusion Transformation transforms operation O_A against another operation O_B in such a way that the impact of O_B is effectively included. Exclusion Transformation transforms operation O_A gainst another operation O_B in such a way that the impact of O_B s effectively excluded.[3] Problem described in the example could be also visualized as a "diamond problem". See Figure 2.2.

Figure 2.2: Diamond problem - 1



Operations done by peer take the document to the left side and operations done on server take the document to the right side. Each node in the diamond diagram represents the number of operations done by client and server in particular state.

As for the example, the correct scenario would be that peer and server exchange their operations, each one computes its transformation and applies the transformated operation on their versions of documents.

The problem presented by this example is acctually the only one that is Operational Transformation capable of solving. Complicated situation occurs when sides of the collaboration diverge by more

then one step.

Figure 2.3: Diamond problem - 2

This example is used in [5] On the server side, tranformation for operations a, c has to be computed, and server applies operation a' on its version of the document. The remaining operation c' has to be preserved for the next transformation. In the next step server must use b and c' as an input and computes b' that can be applied on its version of the document. Server's document is in the desired state.

On the client side, c has to be transformed against two operations to get to desired state. For two operations it is not a big problem, but as the number of operations could very quickly increase, because Operational Transformation is an optimistic technique, so it could potecially take long time to process. One of the advantages of Google Wave Operational Transformation is that it is able to compose the operations if they are compatible. This Composer is able to validate operations to make sure that they are compatible and compose them. One of the criteria for comatibility is that operations must span the same number of indexes. If operations a and b are composed into operation d, tranformation function can take operations d, c and produce operation c'' that can be applied to peer's document and the document is in desired state.

This example is still simple. It does not point out some necessary parts of Operational transformation. The importance of preservation of operation c' was obvious from the diagram, but without the diagram it is hard to determine what operation should server or client preserve. In order to make it easier, metadata about the partent state of the operation is added. Every operation has an identificator of state in which it was executed. Google Wave uses a scalar version number, but [5] suggests to use a hash of the documents contents.

Having information about parent state, the server is able to determine that parent state of operation b is not in server history and it has to derive a new operation, that would take the document to state after application of operations a and c. This operation is c'. This deriving of operation c' is called bridgeing.[5]

With this new feature comes a new problem. In order to derive the operation c' server has to preserve also operation a. This could be a scalability problem in an enviroment with many clients editing the same document, since server has potecially preserve this data for every client. This is resolved by trasnfering all the resposibility to client and buffering the operations on client side. Client can send only operations that come from state in server's history and also he can send only one operation to the server and he has to wait for the acknowledgment of the operation. All following operations are preserved in buffer on client side. One of the advantages of buffering operations is that if this operations are compatible, they can be composed using the Composer. When the server ackwnoledges first operation, client can send next operation. This way client can predict servers path and send only operations that are on the servers path. Also it is much more complicated for the server to track every client, but it generall is much easier for client to track only one server. This way the server solves only one step diamond problem presented in the first example.

Daniel Spiewak in article Understanding and Applying Operational Transformation[5] describes some other improvments of Operational Transformation by Google Wave. For example, when client recieves operation from server and has some operations in buffer, he can transform the whole buffer against this operation in order to accelerate the process on his site and also to ensure that operations in buffer have a parent state in servers history.

The main difference between Google control algorithm and the first dOPT algorithm is that dOPT doesn't use transformation function, it uses transformation matrix. If we have set of operations of cardinality m the matrix should be of size m x m. The entries of the matrix correspond to all possible pairs of operations and contain functions which transform operations to other operations.

Another important part of the algorithm is the state vector. Timestamps for each client are handled by a vector timestamp, where a state vector s_i for a client C_i will have at position j, the number of operations known to have been executed by client C_j.[3]

According to this vector it is decided whether the recieved operation is executed or put in the request queue.

The difference between Google algorithm and Jupiter is not so significant. Actually, Jupiter also uses tranformation fucntion, it is called xform, although it is very similar to dOPT transformation matrix. Only differences is that Jupiter doesn't use the operation composition. According to [4] this algorithm assumes use of transport layer that delivers data in right order, such as TCP.

The last major algorithm implementing this technique is TIPS, which is modern and based on the admissibility-based trasnformation (ABT) framework. It is created on top of HTTP protocol. Main difference is that clients can join and leave session at any time and the algorithm is more reliable when network failures can be expected.

2.3 Differential Synchronization

Differential synchronization is the second tchnique used for real-time collaboration. It was fully described in 2009 by Neil Fraser in paper Differential Synchronization[2]. This section will describe and explain this technique, point out its advatages and disadvatages. As the main source of this information will be used paper mentioned above.

DS also replicates data on all sites of collaboration and identical code runs on both server and client, so it is symmetrical. The main building block is the presence of diff & patch algorithms. Diff algorithm is able to compute a difference between to documents and save it in apropriate format for the patch algorithm, wich is able to apply these changes on the other copy. The patch algorithm must be fuzzy, this means that the changes may be applied even if the document has changed in the meentime, because Differential Sychronization is an optimistic technique. The final version presented by Neil Fraser is suitable for unreliable networks and networks with high-latency. This property will be described later in this section.

The key feature of DS is that it is simple and well suited for use in both novel and existing state-based applications without requireing application redesign.[2] Differential Synchronization is able to process variety of formats, not only plain text. As long as there is a diff and patch algorithm available for the desired format, DS is able to use it. It is implemented and used in MobWrite.

Basic priciple of this tehcnique can be described using data flow diagram in Figure 2.4 originaly presented in [2].

In the begining Client text, Server text and Common Shadow are the same. Client text and Server text represent 2 sites of the collaboration. The goal is to keep them updated. Client and Server are allowed to make changes in their documents at any time.

Figure 2.4: Differential synchronization

After specified time interval a snapshot of Client text is taken and using a diff algorithm difference between snapshot and common shadow is computed. Common shadow represets the last common document state before any edits have been made. This diff returns changes that have been performed by Client. The snapshot is copied over to the common shadow and the changes are patched on the Server text. Now the proces repeats symmetrically using the server text in order to apply changes made by server or other clients on the client text.

2.4 Commutative Replicated Data Types

3 Comparison of Techniques

3.1 Comparison Criteria

3.2 Comparison

4 Komodo

4.1 Overview

4.2 Komodo Requirements for Real-time Collaboration

4.3 Best Technique for Komodo

4.4 Java Implementation

5 Conclusion

Bibliography

[1] ELLIS, C. A., AND GIBBS, S. J. Concurrency control in groupware systems. *SIGMOD Rec. 18*, 2 (June 1989), 399–407.

[2] FRASER, N. Differential synchronization. In *Proceedings of the 9th ACM Symposium on Document Engineering* (New York, NY, USA, 2009), DocEng '09, ACM, pp. 13–20.

[3] LEUNG, C. Operational transformation in cooperative software systems. *McGill Science Undergraduate Research Journal*, 8 (2013).

[4] NICHOLS, D. A., CURTIS, P., DIXON, M., AND LAMPING, J. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology* (New York, NY, USA, 1995), UIST '95, ACM, pp. 111–120.

[5] SPIEWAK, D. Understanding and applying operational transformation, 2015.

[6] SUN, C., JIA, X., ZHANG, Y., YANG, Y., AND CHEN, D. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact. 5*, 1 (Mar. 1998), 63–108.

[7] VIDOT, N., CART, M., FERRIÉ, J., AND SULEIMAN, M. Copies convergence in a distributed real-time collaborative environment. 171–180.

[8] WANG D., MAH A., L. S. Wave protocol google code, 2010.