Implementation of an Algorithm for Peer-to-Peer Collaborative Editing

Damien Aymon
École Polytechnique Fédérale de Lausanne, Switzerland

June 20, 2017

Outline I

Outline II

Introduction

Real-time collaborative editing tools (Google Docs, ...) are widely used nowadays.

Requirements:

- simultaneous editing of a shared document
- convergence
- undo of operations

Introduction

- Major issue : dependence on a central server implies loss of control over the data
- Solution is ABTU : a decentralized p2p algorithm for collaborative editing
- ABTU has been proven to converge

A Bigger Project

Implementation of ABTU is part of a bigger project: a user friendly software for collaborative editing.

Three main parts in the design

- Frontend: user interface and database (JavaScript)
- ABTU Instance: actual ABTU implementation (Go)
- Management: Links frontend and ABTU Instance (Go)

Goals

- Implement p2p communication between two sites
- Implement ABTU algorithm and design interface with management and frontend

If enough time is left

- Test the implementation
- Evaluate the performance of the algorithm
- Link ABTU implementation with frontend

Document and Operations

A document is a string of characters

- Each site has its own copy of the document
- Operations (INS, DEL) are executed on the local copy: $OP(siteId, type, position, character)$

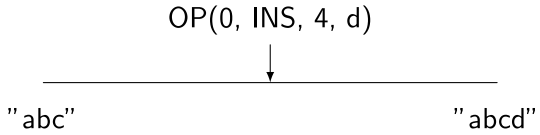

Figure: Execution of an Operation on a String. Own Illustration

Operations are concurrent

Local operation must applied at all other sites.

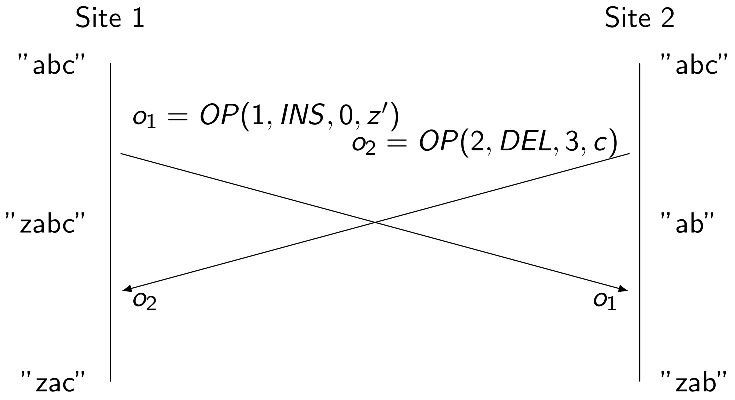

Figure: Generation of two Concurrent Operations. Own Illustration.

Operational transformation

Before a remote operation is executed, is should be transformed:
Operational transformation.

Time

Vector clocks are used to keep track of time:

- When an operation is generated, time increases.
- Operations are timestamped.

Character Order

There is a relation between characters in the system (**effects relation**), even between two characters not present in the same document state. Conceptually, $a \prec b$ if a precedes b in the document.

Undo of Operations

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

The undo of an operation is achieved applying its inverse.
Some operations cannot be undone:

- Define $o_1 = OP(1, INS, 0, a)$ and $o_2 = OP(1, DEL, 0, a)$
- o_2 depends on o_1

Timestamps keep track of the dependence between operations.

Causality and Admissibility Preservation

Two important principles which enforce the convergence of ABTU:

- Causality preservation: if $o_1 \rightarrow o_2$, then o_1 must be invoked before o_2.
- Admissibility preservation: execution of any operation respects effects relation $\prec$.

History and Receiving Buffer

To preserve causality, remote operations are treated once causally ready.

- Receiving buffer *RB* stores untreated remote operations

Before being executed, remote operation must be transformed against executed operation.

- History buffer *H* stores all operations locally executed, in effects relation order.

Local Thread

After local operation has been executed locally:

- Time is incremented
- Operation is timestamped
- Operation is distributed

If the operation is an undo:

- The original operation is recovered.
- Its inverse generated and applied.
- The steps for normal operations executed.

Remote Thread

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

When local thread is not busy, causally ready operation o from RB is treated:

- o is transformed against concurrent operations in H.
- Local time is incremented
- o' is executed.
- o' is integrated in H.

If o is an undo, original operation is marked as undone.

ABTU Instance as a "plug-and-play" module

An instance of the algorithm is represented by a Go structure ABTUInstance:

- Uses 4 Go channels for communication.
- Can be plugged in any frontend that respects the interface.

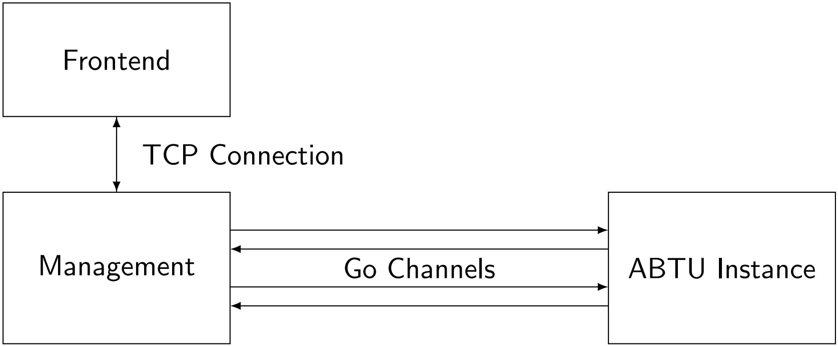

Figure: General Organization of the Software. Own Illustration.

The Big Picture

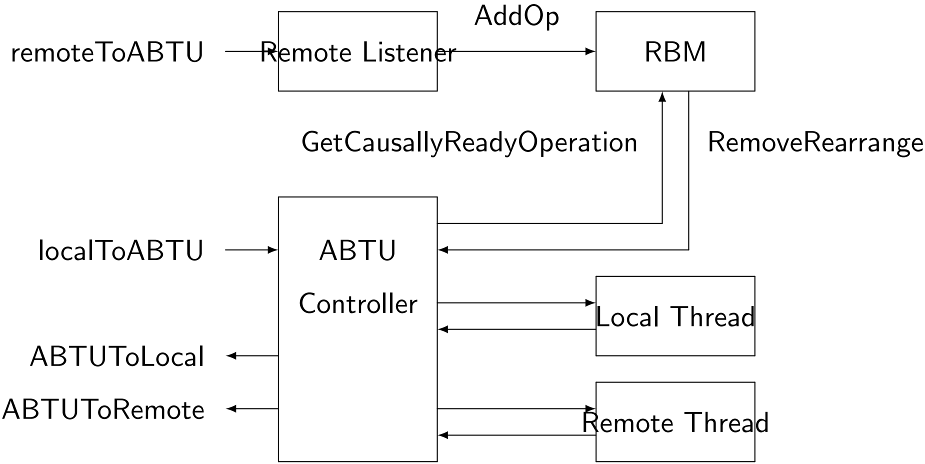

Figure: The Big Picture. Own Illustration

Receiving Buffer Manager

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Two tasks use the receiving buffer concurrently:

- Remote operations are put into *RB*
- Remote thread requests causally ready operation

There is a need for a concurrent datastructure, the receiving buffer manager *RBM*.

Frontend Controller

Life of a local operation:

1. Frontend generates and applies local operation o.
2. o is sent to ABTU Instance
3. o is integrated into H and distributed.

No remote operation can be integrated into H between steps 1 and 3.

Frontend Controller

Frontend implements a controller for the execution of operations:

- No remote operation can be executed as long as pending local operations have not been integrated.
- Wait for ackLocalOperation from ABTU before accepting remote operations

For a local undo:

- No operation can be executed nor generated before undo operation is received from ABTU.
- Wait for ackLocalUndo from ABTU.

ABTU Controller

ABTU instance must implement a controller to give priority to local operations:

ABTU instance must listen for local operations.

- Listen for local operation and handle it.
- Ask for causally ready operation o_r to RBM.
- Handle o_r and send result to frontend. Changes to H, SV and RB should not be applied but stored.
- If local operation is received before "ackRemoteOperation", discard changes. Apply changes otherwise.

The Big Picture

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

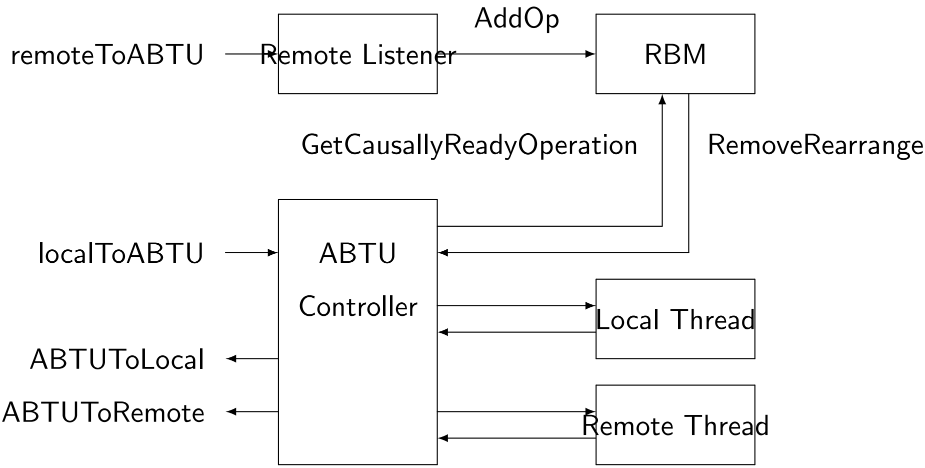

Figure: The Big Picture. Own Illustration

Peer-to-Peer Communication

The communication with other peers is done by using the go-libp2p library:

- A peer is represented by its ip/tcp address, siteId and peerId.
- A communicationService struct can be instanciated with a list of peers.
- The communicationService provides two Go channels for communication (sending/receiving).

The communication is part of the management. This is done to allow more than only ABTU operations to be shared over the network.

Results

The implementation has been tested in different ways. Let us execute one of them:

- Two ABTU instance communicate with each other over p2p.
- Local operations are sent to the first instance.
- The resulting input/output is printed out in the log for both instances.

Limitations and Future Work

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Some feature to be implemented:

- Stopping of an instance is not implemented.
- Communication protocol between peers must be improved ...
- ... to allow for features such as peer joining/leaving.
- Error handling: integrate it in communication with management.
- Secure communication.

Project continuation:

- Further in depth testing of the ABTU implementation
- Performance evaluation
- Linking with management and frontend.
- Feature implementation

Conclusion

- Algorithms for P2P collaborative editing are complex but nonetheless interesting.
- The implementation of ABTU requires a deep understanding of the ABTU framework.
- The perspective of a complete software with an intuitive graphical interface is exciting.