Thesis Defense:
Developing Real-Time Collaborative Editing Using Formal Methods

Lars Tveito

September 9th, 2016

Department of Informatics, University of Oslo

Outline

Introduction

Introduction

- A real-time collaborative editor enables multiple users to work on the same document *simultaneously*.

Introduction

- A real-time collaborative editor enables multiple users to work on the same document *simultaneously*.
- This thesis is on developing a protocol for enabling real-time collaboration in existing editors.

Introduction

- A real-time collaborative editor enables multiple users to work on the same document *simultaneously*.
- This thesis is on developing a protocol for enabling real-time collaboration in existing editors.
- The main problem is handling *concurrent* edits.

Demo

The Naïve Algorithm

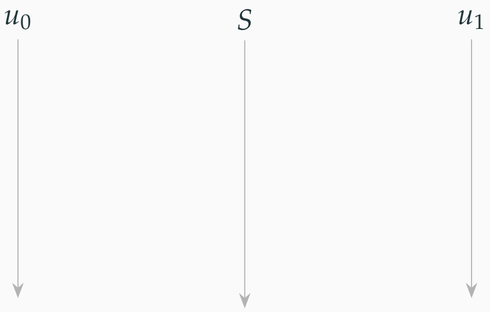


Figure 1: A minimal conflict with two clients.

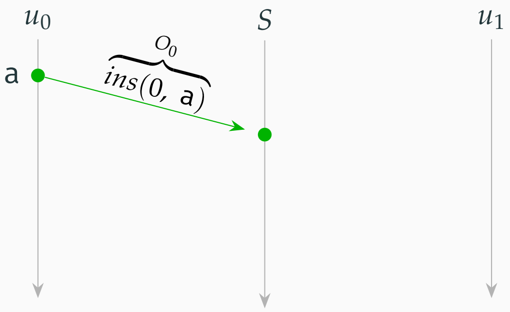


Figure 1: A minimal conflict with two clients.

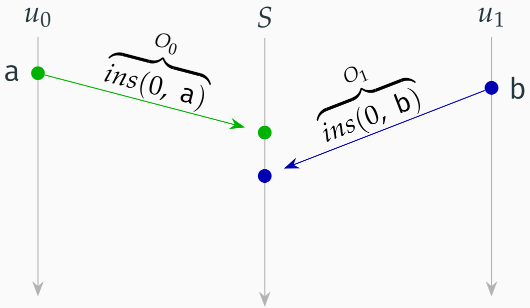


Figure 1: A minimal conflict with two clients.

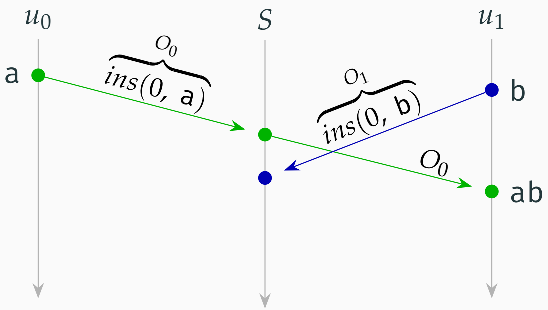


Figure 1: A minimal conflict with two clients.

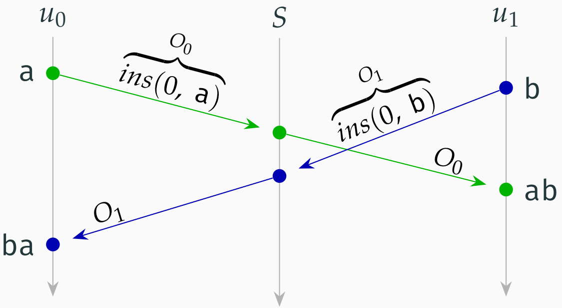
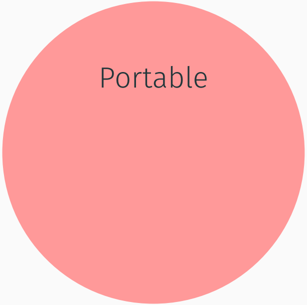


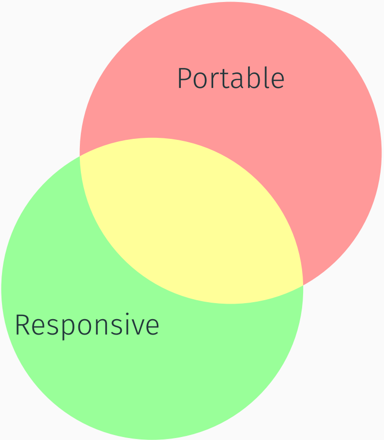
Figure 1: A minimal conflict with two clients.

Goals



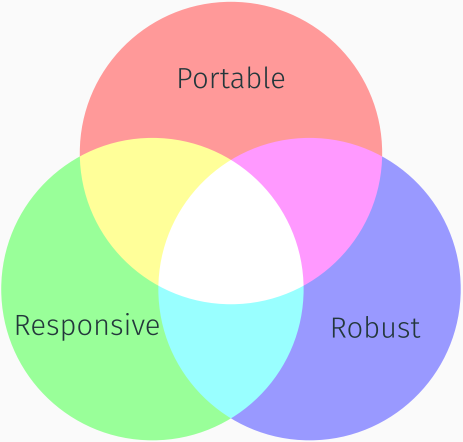
- Easy to implement in an *existing* plain text editor.

Goals



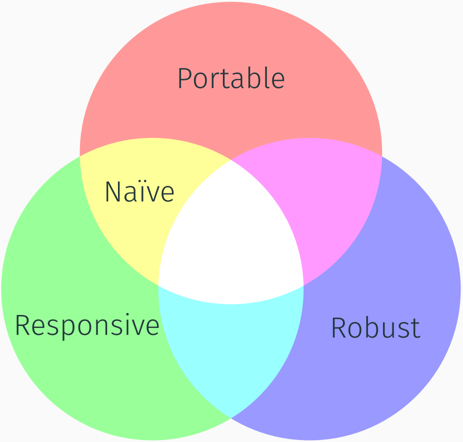
- Easy to implement in an *existing* plain text editor.
- Regular usage of the editor should not be degraded.

Goals



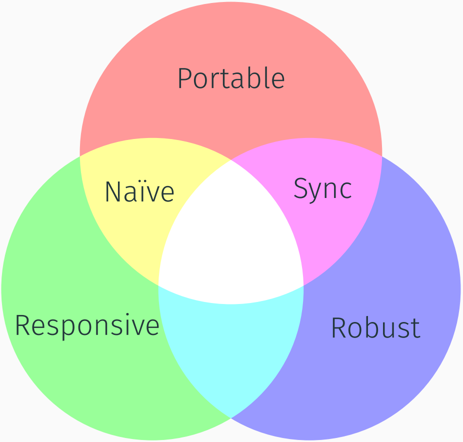
- Easy to implement in an *existing* plain text editor.
- Regular usage of the editor should not be degraded.
- Gracefully handle all conflicting editing operations.

Goals



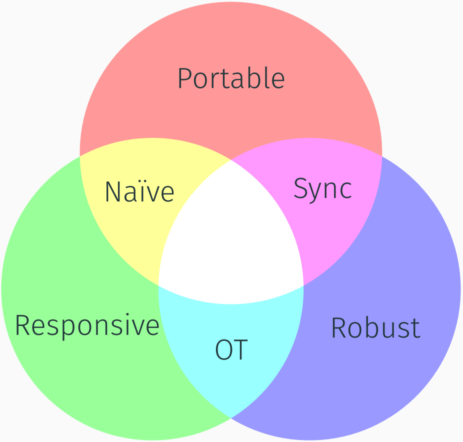
- The Naïve algorithm assumes no concurrent edits.

Goals



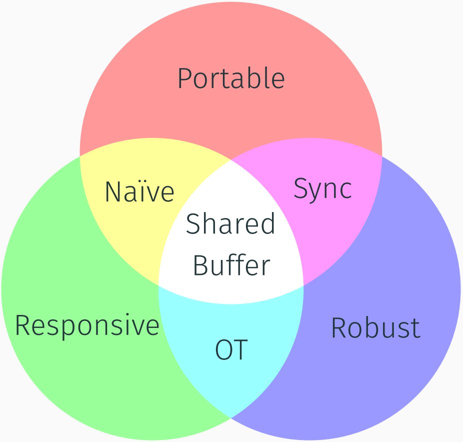
- The Naïve algorithm assumes no concurrent edits.
- We could dissallow concurrent edits by using distributed locks.

Goals



- The Naïve algorithm assumes no concurrent edits.
- We could dissallow concurrent edits by using distributed locks.
- *Operational Transformation* (OT) [1] is the most widely used solution.

Goals



- The Naïve algorithm assumes no concurrent edits.
- We could dissallow concurrent edits by using distributed locks.
- *Operational Transformation* (OT) [1] is the most widely used solution.
- Shared Buffer aims to find the intersection of the three characteristics.

Method

- Strategy: Specify and validate first. Implement after.

Method

- Strategy: Specify and validate first. Implement after.
- We use model checking for validating a *formal* specification.

Contributions

Main contribution:

- A new protocol for real-time collaborative editing.

Contributions

Main contribution:

- A new protocol for real-time collaborative editing.

In the process we have:

- formally specified the system in Maude,

Contributions

Main contribution:

- A new protocol for real-time collaborative editing.

In the process we have:

- formally specified the system in Maude,
- validated the system via model checking using the Maude LTL model checker,

Contributions

Main contribution:

- A new protocol for real-time collaborative editing.

In the process we have:

- formally specified the system in Maude,
- validated the system via model checking using the Maude LTL model checker,
- provided a client-side implementation as an extension for Emacs,

Contributions

Main contribution:

- A new protocol for real-time collaborative editing.

In the process we have:

- formally specified the system in Maude,
- validated the system via model checking using the Maude LTL model checker,
- provided a client-side implementation as an extension for Emacs,
- provided a prototype server-side implementation in Clojure.

Formal Semantics of Editing Operations

Operations and Buffers

- The operations we are concerned with is *insertions* and *deletions*, denoted $ins(i,\ \mathsf{c})$ and $del(i,\ \mathsf{c})$ respectively.

Operations and Buffers

- The operations we are concerned with is *insertions* and *deletions*, denoted $ins(i,\ \mathsf{c})$ and $del(i,\ \mathsf{c})$ respectively.
- All $nop, ins(i,\ \mathsf{c}), del(i,\ \mathsf{c}) \in \mathcal{O}$, and for any two $O_i, O_j \in \mathcal{O}$ then $O_j \circ O_i \in \mathcal{O}$.

Operations and Buffers

- The operations we are concerned with is *insertions* and *deletions*, denoted $ins(i,\ \mathsf{c})$ and $del(i,\ \mathsf{c})$ respectively.
- All $nop, ins(i,\ \mathsf{c}), del(i,\ \mathsf{c}) \in \mathcal{O}$, and for any two $O_i, O_j \in \mathcal{O}$ then $O_j \circ O_i \in \mathcal{O}$.
- A buffer is a 0-indexed string, and $\mathcal{B}$ constitutes the set of all buffers.

Operations and Buffers

- The operations we are concerned with is *insertions* and *deletions*, denoted $ins(i,\ \mathsf{c})$ and $del(i,\ \mathsf{c})$ respectively.
- All $nop, ins(i,\ \mathsf{c}), del(i,\ \mathsf{c}) \in \mathcal{O}$, and for any two $O_i, O_j \in \mathcal{O}$ then $O_j \circ O_i \in \mathcal{O}$.
- A buffer is a 0-indexed string, and $\mathcal{B}$ constitutes the set of all buffers.
- All $O \in \mathcal{O}$ are partial unary operations $O : \mathcal{B} \to \mathcal{B}$.

Operations and Buffers

- The operations we are concerned with is *insertions* and *deletions*, denoted $ins(i,\ \mathsf{c})$ and $del(i,\ \mathsf{c})$ respectively.
- All $nop, ins(i,\ \mathsf{c}), del(i,\ \mathsf{c}) \in \mathcal{O}$, and for any two $O_i, O_j \in \mathcal{O}$ then $O_j \circ O_i \in \mathcal{O}$.
- A buffer is a 0-indexed string, and $\mathcal{B}$ constitutes the set of all buffers.
- All $O \in \mathcal{O}$ are partial unary operations $O : \mathcal{B} \to \mathcal{B}$.
- The set of operations $\mathcal{O}$ is closed under composition and forms an algebraic structure $\langle \mathcal{O}, \circ \rangle$.

Operations and Buffers

- The operations we are concerned with is *insertions* and *deletions*, denoted $ins(i, \text{ c})$ and $del(i, \text{ c})$ respectively.
- All $nop, ins(i, \text{ c}), del(i, \text{ c}) \in \mathcal{O}$, and for any two $O_i, O_j \in \mathcal{O}$ then $O_j \circ O_i \in \mathcal{O}$.
- A buffer is a 0-indexed string, and $\mathcal{B}$ constitutes the set of all buffers.
- All $O \in \mathcal{O}$ are partial unary operations $O : \mathcal{B} \to \mathcal{B}$.
- The set of operations $\mathcal{O}$ is closed under composition and forms an algebraic structure $\langle \mathcal{O}, \circ \rangle$.
- The structure is a monoid, where all elements have an inverse (but is not strictly a group).

Related Work

Basics of Operational Transformation

- The idea of Operational Transformation is to *transform* concurrent operations.

Basics of Operational Transformation

- The idea of Operational Transformation is to *transform* concurrent operations.
- A function $T : \mathcal{O} \times \mathcal{O} \to \mathcal{O}$ is used to transform an operation wrt. another.

Basics of Operational Transformation

- The idea of Operational Transformation is to *transform* concurrent operations.
- A function $T : \mathcal{O} \times \mathcal{O} \rightarrow \mathcal{O}$ is used to transform an operation wrt. another.
- Given two operations $O_i, O_j \in \mathcal{O}$, then $T(O_i, O_j)$ can be understood as: "O_i *as if* O_j had already been applied".

Work on Operational Transformation

- Pioneering work by Ellis and Gibbs [1].

Work on Operational Transformation

- Pioneering work by Ellis and Gibbs [1].
- Work on proving correctness of transformation functions [4, 2, 7, 3].

Work on Operational Transformation

- Pioneering work by Ellis and Gibbs [1].
- Work on proving correctness of transformation functions [4, 2, 7, 3].
- The Jupiter System leverages a server-client architecture for OT [6].

Work on Operational Transformation

- Pioneering work by Ellis and Gibbs [1].
- Work on proving correctness of transformation functions [4, 2, 7, 3].
- The Jupiter System leverages a server-client architecture for OT [6].
- The GOT algorithm [9] introduces:

Work on Operational Transformation

- Pioneering work by Ellis and Gibbs [1].
- Work on proving correctness of transformation functions [4, 2, 7, 3].
- The Jupiter System leverages a server-client architecture for OT [6].
- The GOT algorithm [9] introduces:
    - *inclusion* and *exclusion* transformation functions,

Work on Operational Transformation

- Pioneering work by Ellis and Gibbs [1].
- Work on proving correctness of transformation functions [4, 2, 7, 3].
- The Jupiter System leverages a server-client architecture for OT [6].
- The GOT algorithm [9] introduces:
  - *inclusion* and *exclusion* transformation functions,
  - a undo/do/redo scheme.

Client-side Specification

Maude Specification

- Non-deterministic changes in the system is described in terms of rewrite rules.

Maude Specification

- Non-deterministic changes in the system is described in terms of rewrite rules.
  - A user may insert or delete characters at any time.

Maude Specification

- Non-deterministic changes in the system is described in terms of rewrite rules.
  - A user may insert or delete characters at any time.
  - Messages from the server can arrive at the client at any time.

Maude Specification

- Non-deterministic changes in the system is described in terms of rewrite rules.
    - A user may insert or delete characters at any time.
    - Messages from the server can arrive at the client at any time.
- Equations describe the system's *reaction* to changes in the system. They express the deterministic behaviour of the system.

Maude Specification

- Non-deterministic changes in the system is described in terms of rewrite rules.
  - A user may insert or delete characters at any time.
  - Messages from the server can arrive at the client at any time.
- Equations describe the system's *reaction* to changes in the system. They express the deterministic behaviour of the system.
  - Applying operations to a local buffer, etc...

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
    - Apply the operation.

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
  - Apply the operation.
  - Send the operation, with current seqno and token.

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
  - Apply the operation.
  - Send the operation, with current seqno and token.
  - Increment sequence number.

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
  - Apply the operation.
  - Send the operation, with current seqno and token.
  - Increment sequence number.
- On reception of message:

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
  - Apply the operation.
  - Send the operation, with current seqno and token.
  - Increment sequence number.
- On reception of message:
  - If seqno of the message matches local seqno:

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
  - Apply the operation.
  - Send the operation, with current seqno and token.
  - Increment sequence number.
- On reception of message:
  - If seqno of the message matches local seqno:
    - Apply remote operation.

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
    - Apply the operation.
    - Send the operation, with current seqno and token.
    - Increment sequence number.
- On reception of message:
    - If seqno of the message matches local seqno:
        - Apply remote operation.
        - Set current token to the token of the message.

Client-side Algorithm

All clients keep a sequence number and a token.

- On generation of operation:
  - Apply the operation.
  - Send the operation, with current seqno and token.
  - Increment sequence number.
- On reception of message:
  - If seqno of the message matches local seqno:
    - Apply remote operation.
    - Set current token to the token of the message.
  - Always increment sequence number (regardless of seqno).

Server-side Specification

Maude Specification

- Only one rewrite rule:

Maude Specification

- Only one rewrite rule:
  - the reception of a message.

Maude Specification

- Only one rewrite rule:
    - the reception of a message.
- The equations of the specification expresses the server-side algorithm.

Core Idea

1. The server constructs a history dictating an order of which operations must be applied.

Core Idea

1. The server constructs a history dictating an order of which operations must be applied.
2. Make all clients conform to the constructed history.

Example

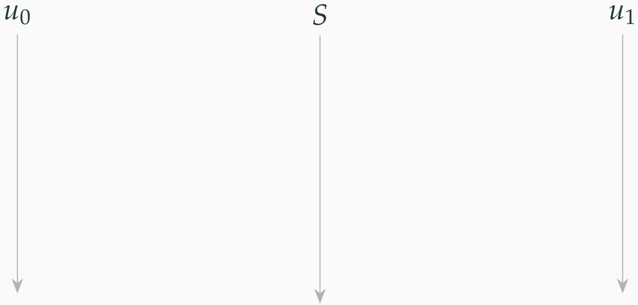


Figure 2: A minimal conflict resolved by Shared Buffer.

Example

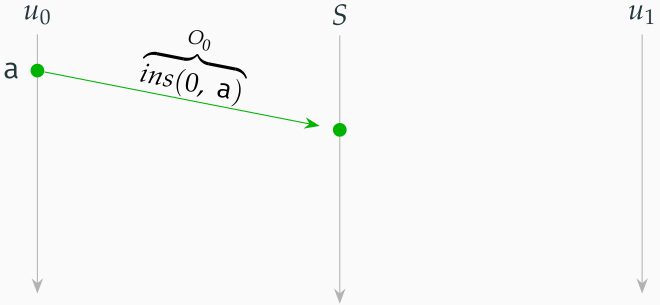


Figure 2: A minimal conflict resolved by Shared Buffer.

Example

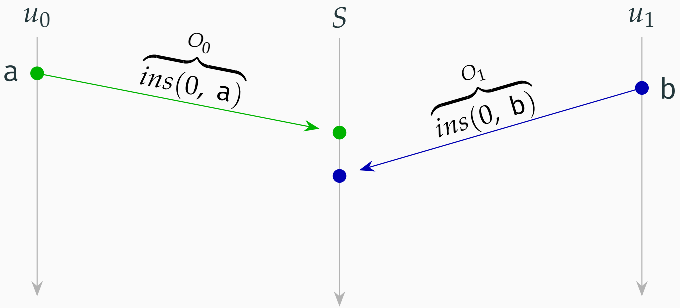


Figure 2: A minimal conflict resolved by Shared Buffer.

Example

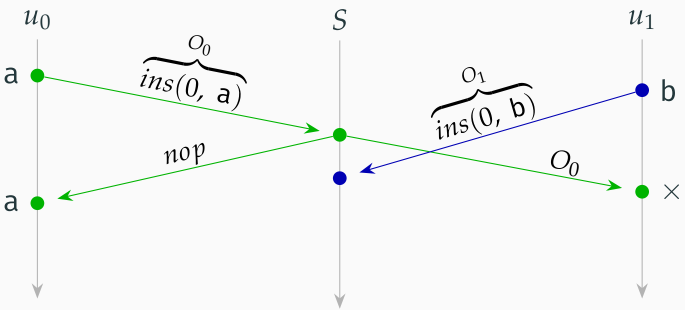


Figure 2: A minimal conflict resolved by Shared Buffer.

Example

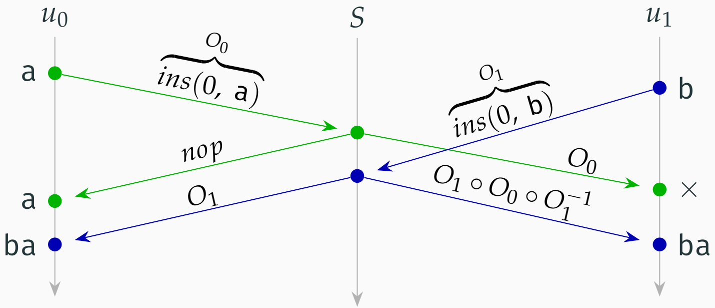


Figure 2: A minimal conflict resolved by Shared Buffer.

Building a History of Events

- An event $\langle O, t, m, u \rangle$ consists of an operation, a token, a time of arrival and a user identifier.

Building a History of Events

- An event $\langle O, t, m, u \rangle$ consists of an operation, a token, a time of arrival and a user identifier.
- The history respects the happened before relation [5].

Building a History of Events

- An event $\langle O, t, m, u \rangle$ consists of an operation, a token, a time of arrival and a user identifier.
- The history respects the happened before relation [5].
- The history further respects a precedence-relation, prioritizing operations working on larger buffer positions.

Building a History of Events

- An event $\langle O, t, m, u \rangle$ consists of an operation, a token, a time of arrival and a user identifier.
- The history respects the happened before relation [5].
- The history further respects a precedence-relation, prioritizing operations working on larger buffer positions.
- We use a subset of the transformation functions from [9] to deal with remaining problems.

Conforming to the History

Two equations can summarize how we construct operations such that clients become consistent with the server's history.

Conforming to the History

$$makeOp(H', H, t) = compose(until(H', t)) \circ compose(until(H, t))^{-1}$$

Conforming to the History

$$makeResponse(O, O', R, t) = O' \circ compose(rejected(R, t)) \circ O^{-1}$$

Model Checking the Specification

Experience

- "A programmer's approach to model checking".

Experience

- "A programmer's approach to model checking".
- Invaluable for finding errors in thinking.

Experience

- "A programmer's approach to model checking".
- Invaluable for finding errors in thinking.
- Working from counter examples was a positive experience.

Results

We cannot report *strong* results wrt. verification, but are no longer able to produce counter examples.

Initial buffer size	Clients	Operations	CPU time
0	2	3	0.3 seconds
1	2	3	2.4 seconds
1	3	3	22.8 seconds
2	3	3	2 minutes 25 seconds
1	4	3	2 minutes 58 seconds
1	2	4	10 minutes 20 seconds
3	3	3	12 minutes 7 seconds
3	4	3	3 hours 8 minutes 18 seconds
2	2	4	3 hours 41 minutes 25 seconds

Implementation of Shared Buffer

State of the Implementation

- Supports multiple concurrent sessions.

State of the Implementation

- Supports multiple concurrent sessions.
- Each session supports an arbitrary number of clients.

State of the Implementation

- Supports multiple concurrent sessions.
- Each session supports an arbitrary number of clients.
- Uses web-friendly technologies, making it easy to include in modern editors.

State of the Implementation

- Supports multiple concurrent sessions.
- Each session supports an arbitrary number of clients.
- Uses web-friendly technologies, making it easy to include in modern editors.
- Transformation functions are currently not included in the implementation.

Cloc

- The server implementation is around 300 cloc.
- The Emacs extension is around 200 cloc.
- The Python library (for testing) is around 150 cloc.
- The Ace extension in Clojurescript is around 70 cloc.

Conclusions and Future Work

Future Work

- Separating *local* and *global* undo, like [8].

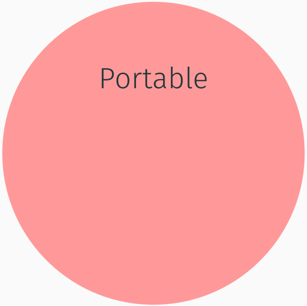
Future Work

- Separating *local* and *global* undo, like [8].
- Investigate if the GOT algorithm can be moved to the server.

Future Work

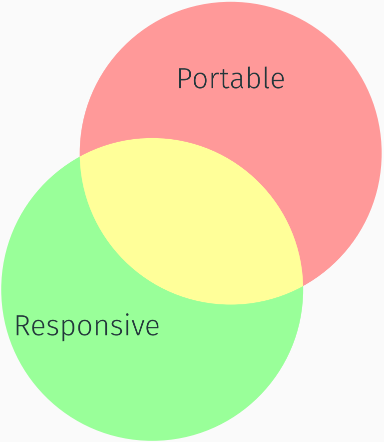
- Separating *local* and *global* undo, like [8].
- Investigate if the GOT algorithm can be moved to the server.
- Fostering a community around Shared Buffer.

Conclusion



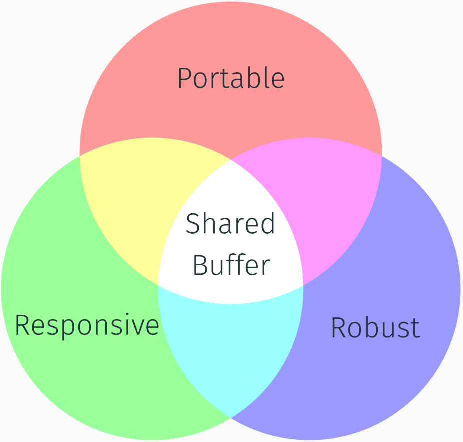
- It is portable, the clients are *thin*.

Conclusion



- It is portable, the clients are *thin*.
- Generally good responsiveness, but clients do not get intermediate results *during* a conflict.

Conclusion



- It is portable, the clients are *thin*.
- Generally good responsiveness, but clients do not get intermediate results *during* a conflict.
- The system is validated to be eventually consistent, and handles most conflicts gracefully.

Thank you

Questions?

References

Clarence A Ellis and Simon J Gibbs. “Concurrency control in groupware systems”. In: *Acm Sigmod Record*. Vol. 18. 2. ACM. 1989, pp. 399–407.

Abdessamad Imine et al. “ECSCW 2003: Proceedings of the Eighth European Conference on Computer Supported Cooperative Work 14--18 September 2003, Helsinki, Finland”. In: ed. by Kari Kuutti et al. Dordrecht: Springer Netherlands, 2003. Chap. Proving Correctness of Transformation Functions in Real-Time Groupware, pp. 277–293. ISBN: 978-94-010-0068-0. DOI: 10.1007/978-94-010-0068-0_15. URL: http://dx.doi.org/10.1007/978-94-010-0068-0_15.

Abdessamad Imine et al. “Formal design and verification of operational transformation algorithms for copies convergence”. In: *Theoretical Computer Science* 351.2 (2006). Algebraic Methodology and Software TechnologyThe 10th International Conference on Algebraic Methodology and Software Technology 2004, pp. 167–183. ISSN: 0304-3975. DOI: http://dx.doi.org/10.1016/j.tcs.2005.09.066. URL: http://www.sciencedirect.com/science/article/pii/S030439750500616X.

- Abdessamad Imine et al. "Proving Correctness of Transformation Functions Functions in Real-Time Groupware". In: *Proceedings of the Eighth European Conference on Computer Supported Cooperative Work, 14-18 September 2003, Helsinki, Finland*. Ed. by Kari Kuutti et al. Springer, 2003, pp. 277–293. URL: http://www.ecscw.org/2003/015Imine_ecscw03.pdf.
- Leslie Lamport. "Time, clocks, and the ordering of events in a distributed system". In: *Communications of the ACM* 21.7 (1978), pp. 558–565.

David A. Nichols et al. "High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System". In: *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*. UIST '95. Pittsburgh, Pennsylvania, USA: ACM, 1995, pp. 111–120. ISBN: 0-89791-709-X. DOI: 10.1145/215585.215706. URL: http://doi.acm.org/10.1145/215585.215706.

Aurel Randolph et al. "On Consistency of Operational Transformation Approach". In: *Proceedings 14th International Workshop on Verification of Infinite-State Systems, Infinity 2012, Paris, France, 27th August 2012*. Ed. by Mohamed Faouzi Atig and Ahmed Rezine. Vol. 107. EPTCS. 2012, pp. 45–59. DOI: 10.4204/EPTCS.107.5. URL: http://dx.doi.org/10.4204/EPTCS.107.5.

- Matthias Ressel, Doris Nitsche-Ruhland, and Rul Gunzenhäuser. "An Integrating, Transformation-Oriented Approach to Concurrency Control and Undo in Group Editors". In: *CSCW '96, Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work, Boston, MA, USA, November 16-20, 1996*. Ed. by Mark S. Ackerman, Gary M. Olson, and Judith S. Olson. ACM, 1996, pp. 288–297. ISBN: 0-89791-765-0. DOI: 10.1145/240080.240305. URL: http://doi.acm.org/10.1145/240080.240305.
- Chengzheng Sun et al. "Achieving Convergence, Causality Preservation, and Intention Preservation in Real-Time Cooperative Editing Systems". In: *ACM Trans. Comput.-Hum. Interact.* 5.1 (1998), pp. 63–108. DOI: 10.1145/274444.274447. URL: http://doi.acm.org/10.1145/274444.274447.