Outline

Introduction
Formal Semantics of Editing Operations
Related Work
Client-side Specification
Server-side Specification
Model Checking the Specification
Implementation of Shared Buffer
Conclusions and Future Work
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.
A real-time collaborative editor enables multiple users to work on the same document \textit{simultaneously}.

This thesis is on developing a protocol for enabling real-time collaboration in existing editors.

The main problem is handling \textit{concurrent} edits.
The Naïve Algorithm

Figure 1: A minimal conflict with two clients.
The Naïve Algorithm

Figure 1: A minimal conflict with two clients.
Figure 1: A minimal conflict with two clients.
The Naïve Algorithm

Figure 1: A minimal conflict with two clients.
The Naïve Algorithm

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

We could disallow 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 disallow 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.
• We use model checking for validating a *formal* specification.
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,
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,
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
The operations we are concerned with is insertions and deletions, denoted $ins(i, c)$ and $del(i, c)$ respectively.
The operations we are concerned with is \textit{insertions} and \textit{deletions}, denoted $\text{ins}(i, c)$ and $\text{del}(i, c)$ respectively.

- All $\text{nop}, \text{ins}(i, c), \text{del}(i, c) \in \mathcal{O}$, 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 $ins(i, c)$ and $del(i, c)$ respectively.
- All $nop, ins(i, c), del(i, c) \in \mathcal{O}$, 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.
The operations we are concerned with is insertions and deletions, denoted $\text{ins}(i, c)$ and $\text{del}(i, c)$ respectively.

All $\text{nop}, \text{ins}(i, c), \text{del}(i, c) \in \mathcal{O}$, 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 $O : \mathcal{B} \rightarrow \mathcal{B}$. 
The operations we are concerned with is *insertions* and *deletions*, denoted $\text{ins}(i, c)$ and $\text{del}(i, c)$ respectively.

- All $\text{nop}, \text{ins}(i, c), \text{del}(i, c) \in \mathcal{O}$, 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 $O : \mathcal{B} \rightarrow \mathcal{B}$.
- The set of operations $\mathcal{O}$ is closed under composition and forms an algebraic structure $\langle \mathcal{O}, \circ \rangle$. 


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

All \( \text{nop}, \text{ins}(i, c), \text{del}(i, c) \in \mathcal{O} \), 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 \( O : \mathcal{B} \rightarrow \mathcal{B} \).

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
The idea of Operational Transformation is to *transform* concurrent operations.
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.
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".
Pioneering work by Ellis and Gibbs [1].

The Jupiter System leverages a server-client architecture for OT.

The GOT algorithm introduces: inclusion and exclusion transformation functions, a undo/do/redo scheme.
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,
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
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.
• 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.
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.
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:
  - 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).
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.
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:
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:
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
● 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.
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.
Figure 2: A minimal conflict resolved by Shared Buffer.
Figure 2: A minimal conflict resolved by Shared Buffer.
Figure 2: A minimal conflict resolved by Shared Buffer.
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.
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

\[ \text{makeOp}(H', H, t) = \text{compose}(\text{until}(H', t)) \circ \text{compose}(\text{until}(H, t))^{-1} \]
Conforming to the History

\[
\text{makeResponse}(O, O', R, t) = O' \circ \text{compose}(\text{rejected}(R, t)) \circ O^{-1}
\]
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.
We cannot report *strong* results wrt. verification, but are no longer able to produce counter examples.

<table>
<thead>
<tr>
<th>Initial buffer size</th>
<th>Clients</th>
<th>Operations</th>
<th>CPU time</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>3</td>
<td>0.3 seconds</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>2.4 seconds</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>22.8 seconds</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>3</td>
<td>2 minutes 25 seconds</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2 minutes 58 seconds</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td>10 minutes 20 seconds</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>12 minutes 7 seconds</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>3</td>
<td>3 hours 8 minutes 18 seconds</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>3 hours 41 minutes 25 seconds</td>
</tr>
</tbody>
</table>
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.
- 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*.
It is portable, the clients are *thin*.  
Generally good responsiveness, but clients do not get intermediate results *during* a conflict.
• 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.
Questions?

