Tree-based model algorithm for maintaining consistency in real-time collaborative editing systems

C. Ignat, M. C. Norrie
Institute for Information Systems
Swiss Federal Institute of Technology (ETH)
ETH-Zentrum, CH-8092 Zurich, Switzerland

ignat@inf.ethz.ch
norrie@inf.ethz.ch
Outline

- Consistency Maintenance in Real-time Collaborative Editing Systems
  - Operational transformation algorithms
- Motivation for a New Algorithm
- The treeOPT Algorithm
  - Main characteristics
  - Proposed document structure
  - Functioning
- Future Work
- Conclusions
Consistency Maintenance in Real-time Collaborative Editing Systems

- **Real-time collaborative editing systems**: Groupware systems that allow members of a team to simultaneously edit shared documents from different sites.

- **Operational transformation**: dOPT, adOPTed, GOT, GOTO, SOCT2, SOCT3, SOCT4
Motivation(1)

- Existing algorithms – linear representation of the document
  - A single history buffer
    - large number of transformations (low efficiency, response-time)
    - all operations “interfere” with each other

=> we need an algorithm that reduces the no. of transformations, operations at completely different positions not interfering with each other
Motivation (2)

- Existing algorithms – enforce **syntactic consistency**, not **semantic consistency**

  **Example 1** (working at character level):
  Initial document: “Helo everybody.”
  - First user: inserts an “l”
    “Hello everybody.”
  - Second user: deletes “Helo” and writes “Bye”
    “Bye everybody.”

  There is no semantically correct way to solve the conflict automatically

  Algorithms may obtain “Byel everybody.”
Motivation(3)

Example 2 (working at word level)

Initial document:

“The child go alone to school.”

First user: deletes “go”, inserts “goes”

“The child goes alone to school.”

Second user: inserts “can”.

“The child can go alone to school.”

Result: “The child can goes alone to school.”

=> we need an algorithm that addresses semantic consistency, by allowing users to work at coarser (and selectable) granularity
New Algorithm - Characteristics

- Supports a hierarchical structure of the document
- Improves efficiency
- Allows working at any level of granularity: paragraph, sentence, word, character
- Step towards semantic consistency
Proposed document structure

- **Document**
  - Pa 1
  - Pa 2
  - Pa 3
  - **Document History**
    - Se 3.1
    - Se 3.2
    - Pa 3 History
      - W 3.1.1
      - W 3.1.2
      - W 3.1.3
        - Se 3.1 History
          - C 3.1.2.1
            - “C”
          - C 3.1.2.2
            - “A”
          - C 3.1.2.3
            - “R”
        - W 3.1.2 History
          - History for operations at paragraph level
          - History for operations on sentences in paragraph Pa3
          - History for operations on words in sentence Se3.1
          - History for operations on characters in word W3.1.2
Structure of a node

**Node** – a structure of the form:

\[ N = \{\text{level}, \text{children}, \text{length}, \text{history}, \text{content}\} \]

- **level** – granularity level, \( \text{level} \in \{0, 1, 2, 3, 4\} \)
- **children** – ordered list of nodes \( \{\text{child}_1, \ldots, \text{child}_n\} \)
  
  \[ \text{level} (\text{child}_i) = \text{level} + 1, \text{ for all } i \in \{1, \ldots, n\} \]

- **length** -
  \[
  \begin{cases} 
  1, \quad \text{if } \text{level} = 4 \\
  \sum_{i=1}^{n} \text{length} (\text{child}_i), \quad \text{otherwise}
  \end{cases}
  \]

- **content** -
  \[
  \begin{cases} 
  \text{undefined}, \quad \text{if } \text{level} < 4 \\
  \text{aCharacter}, \quad \text{if } \text{level} = 4
  \end{cases}
  \]
Structure of a Composite Operation

- Composite operation - a structure of the form
  \[ cOp = \{ \text{level}, \text{type}, \text{position}, \text{content}, \text{statevector}, \text{initiator} \} \]

  - \textit{level} - granularity level, \( \text{level} \in \{1, 2, 3, 4\} \)
  - \textit{type} - type of the operation, \( \text{type} \in \{\text{Insertion, Deletion}\} \)
  - \textit{position} - vector of positions
    \[ \text{position}[i] = \text{position for the } i^{\text{th}} \text{ granularity level, } i \in \{1, \ldots, \text{level}\} \]
  - \textit{content} - a node, representing the content of the operation
  - \textit{statevector} - state vector of the generating site
  - \textit{initiator} - initiator site identifier

- Example: \( cOp(2, \text{Insertion}, [3,1,x,x], \text{sentence}, \text{statevector}) \)
  an insertion of the sentence given by node “sentence”, timestamped by “statevector”, in paragraph 3, as sentence 1 in the paragraph
The treeOPT Algorithm – Functioning (1)

Example – a site receives the remote operation \texttt{InsertWord}(3,1,3,"love");
The treeOPT Algorithm — Functioning (2)

- A concurrent operation inserts a paragraph

**InsertWord**$(3,1,3,\text{“love”}) \Rightarrow **InsertWord**$(4,1,3,\text{“love”})$

- We apply the algorithm for all granularity levels, beginning with paragraph level, and ending with level of the operation.
The algorithm works with any existing concurrency control algorithm relying on a linear document structure.

We tested the operation of the algorithm when combined with the GOT algorithm, and the corresponding undo/do/redo scheme.
Future Work

- Adapting the algorithm for a *graphical editor*
- *Locking* at different granularity levels
- *Social aspects* (audio communication, chat systems)
- UIP (Universal Information Platform) project
Conclusion

- **treeOPT** is a consistency maintenance algorithm relying on a *tree representation* of the document and applying the operational transformation mechanism over different document levels.

  - Increases efficiency
  - Provides means to enforce “semantic” consistency
  - Allows flexibility of granularity