On the Undoability Problem for Collaborative Applications

Presented by Asma Cherif

Cassis Team, Loria

March 22, 2012

Introduction

- Undoing operations is an indispensable feature for many collaborative applications
- It provides the ability to restore a correct state of the shared data after erroneous operations
- Selective undo allows:
    - users to undo any operation and is based on rearranging operations in the history

Challenge

- Combining OT and undo approaches is a challenging task
- Undo introduces new kind of operations (inverse)
- Presence of undo puzzles leading to divergence
- Absence of formal guidelines for undo: the correctness of an undo solution is still a challenging problem

- Swap then undo [Prakash]: First selective undo
  - Put the operation in the end then undo it
  - (-) swap is not always possible ? conflict() notion that aborts undo
- Undo/Redo [Ressel]
  - Overcome the conflict by undoing all operations after the one to undo then undo it and redo these operations
  - (-) expensive and does not allow to undo il all cases
- Ferrié Based on SOCT2 (diverge)
- Uno [Weis]
  - Based on TTF
  - Generate an operation having an effect undoing the selected operation
- AnyUndo-X & COT [Sun]
  - Avoid undo properties instead of fulfilling them
- ABTU [Shao]: based on ABT (diverge)

Logging all executed operations is necessary to accomplish an undo scheme.

- each operation *op* has its inverse operation $\overline{op}$.

Logging all executed operations is necessary to accomplish an undo scheme.

- each operation op has its inverse operation $\overline{op}$.
- undo operation op_i from the log $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$:
  1. Find op_i in L;

$$L$$
$$\begin{array}{|c|}\hline op_1 \\ op_2 \\ \vdots \\ op_{i-1} \\ op_i \\ op_{i+1} \\ \vdots \\ op_n \\ \hline\end{array}$$

Logging all executed operations is necessary to accomplish an undo scheme.

- each operation op has its inverse operation $\overline{op}$.
- undo operation op_i from the log
  $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$:
  1. Find op_i in L;
  2. Mark op_i as an undone operation: op_i^*;

$$
\begin{array}{c}
L \\
\boxed{\begin{array}{c} op_1 \\ op_2 \\ \vdots \\ op_{i-1} \\ op_i^* \\ op_{i+1} \\ \vdots \\ op_n \end{array}}
\end{array}
$$

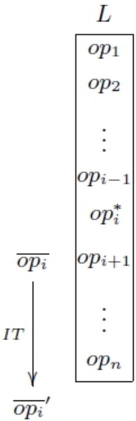
Logging all executed operations is necessary to accomplish an undo scheme.

- each operation op has its inverse operation $\overline{op}$.
- undo operation op_i from the log $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$:
  1. Find op_i in L;
  2. Mark op_i as an undone operation: op_i^*;
  3. Generate $\overline{op_i}$;

$$
\begin{array}{cc}
 & L \\
 & \begin{array}{|c|} \hline op_1 \\ op_2 \\ \vdots \\ op_{i-1} \\ op_i^* \\ op_{i+1} \\ \vdots \\ op_n \\ \hline \end{array}
\end{array}
\quad
\begin{array}{c} \overline{op_i} \end{array}
$$

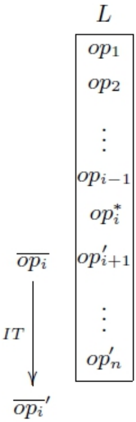
Logging all executed operations is necessary to accomplish an undo scheme.

- each operation op has its inverse operation $\overline{op}$.
- undo operation op_i from the log
  $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$:
  1. Find op_i in L;
  2. Mark op_i as an undone operation: op_i^*;
  3. Generate $\overline{op_i}$;
  4. Calculate $\overline{op}' = IT^*(\overline{op_i}, op_{i+1} \cdot \ldots \cdot op_n)$ that integrates the effect of operations following op_i in L;



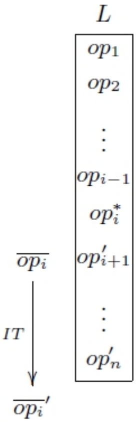
Logging all executed operations is necessary to accomplish an undo scheme.

- each operation op has its inverse operation $\overline{op}$.
- undo operation op_i from the log $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$:
  1. Find op_i in L;
  2. Mark op_i as an undone operation: op_i^*;
  3. Generate $\overline{op_i}$;
  4. Calculate $\overline{op}' = IT^*(\overline{op_i}, op_{i+1} \cdot \ldots \cdot op_n)$ that integrates the effect of operations following op_i in L;
  5. Exclude the effect of op_i from the log by including the effect of $\overline{op_i}$ inside the sublog $op_{i+1} \cdot \ldots \cdot op_n$; The sequence of operations following op_i^* is then $op'_{i+1} \cdot \ldots \cdot op'_n$;



Logging all executed operations is necessary to accomplish an undo scheme.

- each operation op has its inverse operation $\overline{op}$.
- undo operation op_i from the log
  $L = op_1 \cdot op_2 \cdot \ldots \cdot op_i \cdot \ldots \cdot op_n$:
  1. Find op_i in L;
  2. Mark op_i as an undone operation: op_i^*;
  3. Generate $\overline{op_i}$;
  4. Calculate $\overline{op}' = IT^*(\overline{op_i}, op_{i+1} \cdot \ldots \cdot op_n)$ that integrates the effect of operations following op_i in L;
  5. Exclude the effect of op_i from the log by including the effect of $\overline{op_i}$ inside the sublog $op_{i+1} \cdot \ldots \cdot op_n$; The sequence of operations following op_i^* is then $op'_{i+1} \cdot \ldots \cdot op'_n$;
  6. Execute $\overline{op}'$.



OT Properties

For all op_1, op_2 and op_3 pairwise concurrent operation, IT is *correct* iff the following properties are satisfied:

- **Property TP1**: $st \cdot op_1 \cdot IT(op_2, op_1) = st \cdot op_2 \cdot IT(op_1, op_2)$, for every state st.
- **Property TP2**: $IT(IT(op_3, op_1), IT(op_2, op_1)) = IT(IT(op_3, op_2), IT(op_1, op_2))$.

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

- **Property IP1**: $op_1 \cdot \overline{op_1} \equiv \emptyset$;

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

- **Property IP1**: $op_1 \cdot \overline{op_1} \equiv \emptyset$;
- **Property IP2**: $IT(IT(op_1, op_2), \overline{op_2}) = op_1$;

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

- **Property IP1**: $op_1 \cdot \overline{op_1} \equiv \emptyset$;
- **Property IP2**: $IT(IT(op_1, op_2), \overline{op_2}) = op_1$;
- **Property IP3**: $IT(\overline{op_1}, op_2') = \overline{IT(op_1, op_2)}$ where

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

- **Property IP1**: $op_1 \cdot \overline{op_1} \equiv \emptyset$;
- **Property IP2**: $IT(IT(op_1, op_2), \overline{op_2}) = op_1$;
- **Property IP3**: $IT(\overline{op_1}, op_2') = \overline{IT(op_1, op_2)}$ where
    - $op_1' = IT(op_1, op_2)$

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

- **Property IP1**: $op_1 \cdot \overline{op_1} \equiv \emptyset$;
- **Property IP2**: $IT(IT(op_1, op_2), \overline{op_2}) = op_1$;
- **Property IP3**: $IT(\overline{op_1}, op_2') = \overline{IT(op_1, op_2)}$ where
  - $op_1' = IT(op_1, op_2)$
  - $op_2' = IT(op_2, op_1)$

Undo Properties

Given a correct transformation function IT and any two operations op_1 and op_2:

- **Property IP1**: $op_1 \cdot \overline{op_1} \equiv \emptyset$;
- **Property IP2**: $IT(IT(op_1, op_2), \overline{op_2}) = op_1$;
- **Property IP3**: $IT(\overline{op_1}, op_2') = \overline{IT(op_1, op_2)}$ where
  - $op_1' = IT(op_1, op_2)$
  - $op_2' = IT(op_2, op_1)$
  - $op_1 \cdot op_2' \equiv op_2 \cdot op_1'$

Example

a shared integer register The state of the shared integer register is altered by two operations:

- Inc()
- Dec()
- $IT : IT(op_i, op_j) = op_i$

Verifies **IP1**, **IP2** and **IP3**

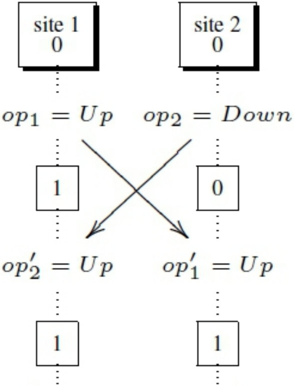
Example

Consider a shared binary register where two primitive operations modify the state of a bite from 0 to 1 and vice versa:

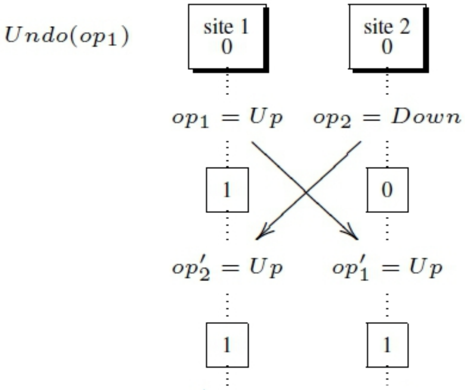
- Up to turn on the register;
- Down to turn off the register;
- IT:
  - $IT(Up, Up) = IT(Up, Down) = IT(Down, Up) = Up$
  - $IT(Down, Down) = Down$

Violates **IP1**, **IP2** and **IP3**

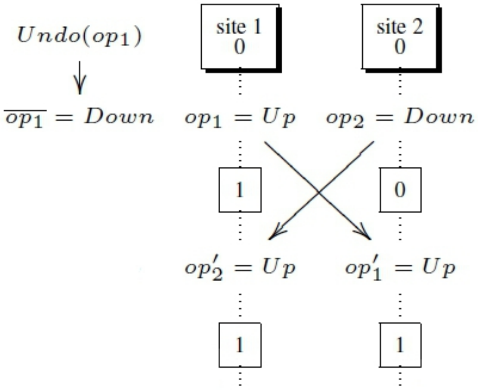
IP3 violation



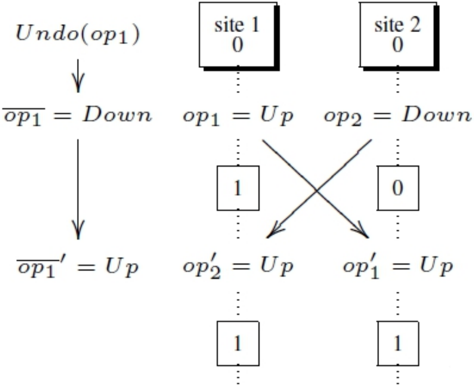
IP3 violation



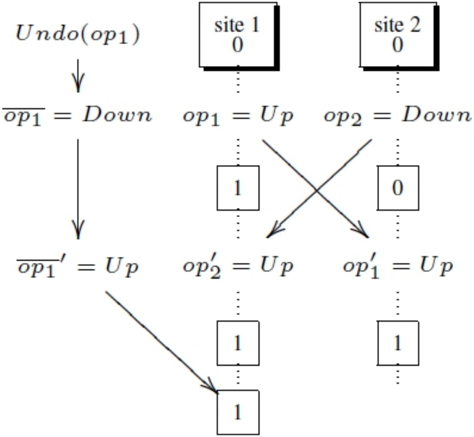
IP3 violation



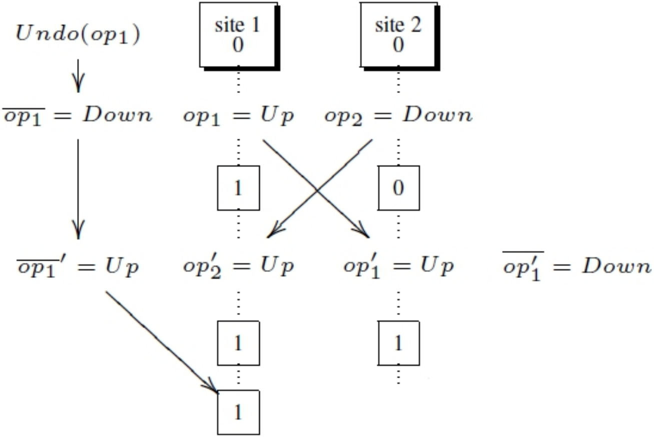
IP3 violation



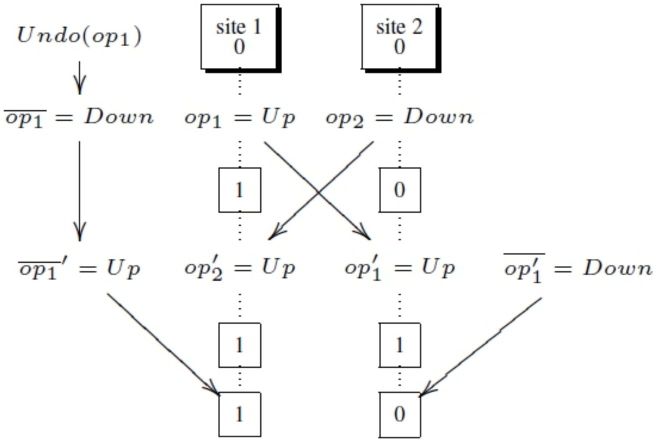
IP3 violation



IP3 violation



IP3 violation



Definition of undoability

Undoability

A set is undoable iff IP1/2/3 are preserved

Definition of CCO

Consistent Collaborative Object (CCO)

A *Consistent Collaborative object* is a triplet $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$ such that:

- $\mathcal{S}t$ is the set of object states
- $\mathcal{O}p$ is the set of primitive operations executed by the user to modify the object state. This set is characterized by the following properties:
  - for every operation $op \in \mathcal{O}p$ there is *unique inverse* $\overline{op} \in \mathcal{O}p$ such that $op \neq \overline{op}$ and $st \cdot op \cdot \overline{op} = st$ for all states $st \in \mathcal{S}t$;
  - for every operation $op \in \mathcal{O}p$ there exists a state $st \in \mathcal{S}t$ such that $st \cdot op = st'$ where $st' \neq st$.
- $IT : \mathcal{O}p \times \mathcal{O}p \rightarrow \mathcal{O}p$ is a correct transformation function

Formal Problem Statement

Problem: Given a consistent collaborative object $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, what are the necessary and the sufficient conditions that C is undoable?

Steps of the proof

Main Theorem

A CCO is undoable iff Op is commutative

Steps of the proof

Main Theorem

A CCO is undoable iff Op is commutative

To prove the the main theorem, we follow these steps:

1. define commutativity for concurrent operations (using IT)

Steps of the proof

Main Theorem

A CCO is undoable iff Op is commutative

To prove the the main theorem, we follow these steps:

1. define commutativity for concurrent operations (using IT)
2. prove that commutativity is necessary for undoability

Steps of the proof

Main Theorem

A CCO is undoable iff Op is commutative

To prove the the main theorem, we follow these steps:

1. define commutativity for concurrent operations (using IT)
2. prove that commutativity is necessary for undoability
3. prove tha commutativity is sufficient to reach undoability

Definition of commutativity in terms of IT

- Given two concurrent operations op_1 and op_2
- Correct IT (TP1 & TP2)

op_1 commutes with op_2 iff

- $IT(op_1, op_2) = op_1$
- $IT(op_2, op_1) = op_2$

Commutativity implies undoability

If $\mathcal{O}p$ is commutative then $IP2$ and $IP3$ are preserved:

Commutativity implies undoability

If $\mathcal{O}p$ is commutative then $IP2$ and $IP3$ are preserved:

- IP2:

$$IT(IT(op_1, op_2), \overline{op_2}) = IT(op_1, op_2) = op_1$$

Commutativity implies undoability

If $\mathcal{O}p$ is commutative then $IP2$ and $IP3$ are preserved:

- IP2:
$$IT(IT(op_1, op_2), \overline{op_2}) = IT(op_1, op_2) = op_1$$
- IP3:
$$IT(\overline{op_1}, IT(op_2, op_1)) = IT(\overline{op_1}, op_2) = \overline{op_1}$$

Undoability implies Commutativity

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, proving that undoability implies commutativity requires

to find all evaluations of IT that respect $TP1/2$ and $IP1/2/3$

Undoability implies Commutativity

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, proving that undoability implies commutativity requires

to find all evaluations of IT that respect $TP1/2$ and $IP1/2/3$

- Combinatorial problem

Undoability implies Commutativity

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, proving that undoability implies commutativity requires

to find all evaluations of IT that respect $TP1/2$ and $IP1/2/3$

- Combinatorial problem
- We have no information about the result of IT function for a couple (op_1, op_2)

Undoability implies Commutativity

Given a CCO $C = \langle \mathcal{S}t, \mathcal{O}p, IT \rangle$, proving that undoability implies commutativity requires

to find all evaluations of IT that respect $TP1/2$ and $IP1/2/3$

- Combinatorial problem
- We have no information about the result of IT function for a couple (op_1, op_2)
- We resort to CSP theory to overcome the proof's difficulty

CSP Theory

$$\langle \mathcal{X}, \mathcal{D}. \mathcal{C} \rangle$$

Set of variables

$$\{IT(op_i, op_j) | (op_i, op_j) \in \mathcal{O}p^2\}$$

Domain of values

$$\mathcal{O}p$$

Set of constraints

$$TP1/2 - -IP1/2/3$$

Necessity proof by induction on the size of $\mathcal{O}p$

- use induction on the size of $\mathcal{O}p$
- basic cases of induction proved with a CSP solver
  - CCO of two operations
  - CCO of four operations

Necessity Proof

basic cases of induction proved with a CSP solver

- CCO of two operations: two possible cases produced by our CSP solver:
  - $\forall op_1, op_2 \in \mathcal{O}p \; IT(op_1, op_2) = op_1$
  - $\mathcal{O}p$ is 4-periodic and is commutative
- CCO of four operations:
  - $\forall op_1, op_2 \in \mathcal{O}p \; IT(op_1, op_2) = op_1$
  - $\mathcal{O}p$ is 4-periodic and is commutative
  - $\mathcal{O}p$ is 6-periodic and is commutative

Conclusion

- Contributions:
  - we address the undo problem from a theoretical point of view
  - we propose a necessary and sufficient condition for undoing replicated objects based on OT with respect to inverse properties IP1/2/3
  - Use of CSP theory to overcome the difficulty of necessity proof in order to cover all possible transformation cases
  - main result: it is impossible to achieve a correct undo for applications based on non commutative operations
- Future work:
  - generalize the result to cover the special case where an operation is equal to its inverse
  - relax our condition and see if the use of idle operations may ensure undoability