Operational transform
Bag operations

30 Aug 2005
David Barrett-Lennard

Abstract
This paper provides a solution to the operational transform of bag operations.

Introduction
The document state is assumed to be a "bag" over some element type T. A bag is like a set except that repeats are allowed. The repeated elements are indistinguishable from one another. An analogy is a bag of marbles which can hold a number of marbles of a given colour.

T is a value type, and is assumed to support an equality comparison operator. In this paper we study the operational transform of the following operations

- ins(v) - insert value v into the bag. This operations can always be generated, irrespective of whether the value v is already present in the bag.
- del(v) - delete value v from the bag. It is assumed that value v can be found in the bag when this operation is generated.

We need convergence at quiescence even though operations are executed in different orders at different sites. Specifically we require properties TP1 and TP2 [1].

Properties required to achieve convergence

TP1
$\forall\ O_1, O_2 \quad S + [O_1\ IT(O_2, O_1)] = S + [\ O_2\ IT(O_1, O_2)\ ]$

TP2
$\forall\ O_1, O_2, O_3, \quad IT(\ IT(\ O_3, O_1),\ IT(O_2, O_1)) = IT(\ IT(\ O_3, O_2),\ IT(O_1, O_2))$

Discussion
In the following cases, let O_1, O_2 be concurrent operations that are performed on the same document state S.

Let operation O_1 insert value v_1, and O_2 inserts value v_2. Irrespective of what is already in the bag, and whether $v_1=v_2$, O_1 and O_2 commute and therefore nothing needs to be done under IT.

Let operation O_1 insert value v_1, and O_2 delete value v_2. Irrespective of whether $v_1 \neq v_2$ we can assume v_2 exists in the bag in state S, and these operations commute so nothing needs to be done under IT.

Let operation O_1 delete value v_1, and O_2 delete value v_2. Assuming $v_1 \neq v_2$ we can assume these values exist independently in the bag, and these operations commute so nothing needs to be done under IT.

The only difficult case is where O_1 and O_2 both delete value v. If there are two or more instances of value v in the bag then we could allow both these operations to be applied. Otherwise it will be necessary to disable under IT. For simplicity we choose to *always* disable under IT, as though the two users were fighting over removal of the same instance even though there were more than one available.

It is necessary to only allow an *enabled* delete operation to cause another delete operation to become disabled.

Solution

Let an operation contain the following fields

Field	Description
ins	Boolean flag to indicate whether this is an insert or delete operation
dc	Disable count, only relevant to a delete operation. Initialised to zero when the operation is first generated.
v	Value to be inserted or deleted

The following C++ code shows the implementation of IT/ET

```
void IT(SetOp& O1, const SetOp& O2)
{
    if (!O2.ins && !O1.ins && O1.v == O2.v && O2.dc == 0) ++O1.dc;
}

void ET(SetOp& O1, const SetOp& O2)
{
    if (!O2.ins && !O1.ins && O1.v == O2.v && O2.dc == 0) --O1.dc;
}
```

Future work

For performance this solution should be extended to support operations that insert or delete multiple elements at once. This also allows for operations to be closed under merging. Furthermore it will be possible to compress merged operations by eliminating elements that are inserted then deleted by the operation. Compression can also discard disabled delete operations.

References

[1]	David Barrett-Lennard. Operational transform - Control algorithm. Aug 2005