Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Verso la convergenza tra Operational Transformation e Change Tracking

Relatore:
Chiar.mo Prof.
Fabio Vitali

Presentata da:
Pietro Lami

Sessione II
Anno Accademico 2017-2018

Indice

Introduzione

Nei sistemi di collaborazione real-time, il problema di scegliere come risolvere e quali tecniche utilizzare per la risoluzione di conflitti tra le operazioni concorrenti non è semplice, poiché se affrontato non correttamente può portare a stati di incoerenza del modello di dati condiviso tra gli utenti connessi al sistema.
Questo problema può essere ulteriormente reso complesso da altri fattori, come il ritardo dovuto alla computazione delle operazioni o alla latenza della rete; questo avviene poiché si tratta quasi sempre di un ambiente distribuito in cui sono presenti un elevato numero di nodi che le informazioni scambiate devono attraversare.
Nei sistemi collaborativi può capitare di non essere solo interessati allo stato corrente di un documento, ma anche alla sua evoluzione e ai cambiamenti che lo hanno portato allo stato attuale e quindi è necessario avere un sistema capace di gestire adeguatamente le versioni create da diversi autori; mantenere le informazioni relative a questi autori e fornire un valido supporto per l'analisi dei cambiamenti.
Nei sistemi di editing collaborativo real-time si adoperano diverse tecnologie, quelle che ci interesseranno maggiormente sono Operational Transformation e Change Tracking. La prima è una tecnologia che è usata per supportare un'insieme di funzionalità per la collaborazione degli utenti: il mantenimento della coerenza di un documento finale, proposta per la prima volta in un articolo scientifico da Ellis e Gibbs nel 1989. La seconda viene utilizzata in questi sistemi per permettere una visione semplice e comoda dell'evoluzione di un documento attraverso il tempo.
L'importanza del sistema di Change Tracking è quindi fondamentale nel supporto per l'analisi delle diverse versioni di un documento. È di utilità fonda-

mentale all'interno di un sistema in cui ogni utente può modificare e creare nuove versioni di un documento. In generale, sarebbe un buon requisito per un qualunque sistema in cui è interessante seguire lo sviluppo dei documenti in funzione del tempo.
Data l'importanza di Operational transformation e Change Tracking in questa tesi verranno dunque analizzati approfonditamente le loro fondamenta teoriche, inoltre verranno successivamente confrontate sia teoricamente sia tramite l'implementazione e la discussione dell'editor collaborativo DOTT (Document Operational Transformation Tracking).
Inizialmente verranno descritti in maniera generale i sistemi di collaborazione real-time, le loro caratteristiche e le tecnologie in Javascript, che permettono la creazione di un canale bidirezionale per la realizzazione del real-time, come: Ajax Long-Polling, che consiste nell'interrogazione del server a intervalli regolari, ricevendo risposta quando il server produce nuovi dati; Server Sent Events, che implicano una comunicazione monodirezionale dal server al client e che permettono al server di inviare dati al client in qualunque momento e WebSockets, che permettono l'apertura di una sessione di comunicazione interattiva tra il browser del client e il server; l'ultima tecnologia è la più indicata per la gestione di applicazioni real-time in Javascript.
Di seguito verranno descritti approfonditamente le fondamenta teoriche di OT: analizzando il suo funzionamento, quali sono i modelli di coerenza principali in questi sistemi, le proprietà delle funzioni di trasformazione, la struttura ad alto livello, il problema del diamante, come si può adattare ai documenti strutturali; poi verranno introdotte e descritte brevemente possibili alternative a OT, come Differential synchronization e Conflict-free replicated data type, dando infine una giustificazione della scelta di OT.
Inoltre si mostrerà la teoria di Change Tracking, due modi per sviluppare un sistema che lo utilizza e due esempi di come si può rappresentare nei sistemi. Successivamente si descriverà una rappresentazione di un file JSON come punto di incontro fra OT e Change Tracking, con il fine di dimostrare che i due metodi, che sono effettivamente diversi, approcciano in realtà lo stesso problema che è quello della rappresentazione delle modifiche in un documento.
Infine si analizzerà l'applicazione web DOTT che è un editor di testo colla-

borativo che utilizza OT per assicurare la convergenza dei file su tutti i client e si serve del file JSON per rappresentare la storia del documento; questo permette di non dover utilizzare diverse tecnologie per applicare le operazioni e mantenere la storia in un documento condiviso.

Capitolo 1

Background

In questo capitolo viene descritto cos'è un sistema di editing collaborativo in tempo reale e le tecnologie Javascript per ottenere canali real-time utilizzati in sistemi collaborativi.

1.1 Sistemi di editing collaborativo real-time

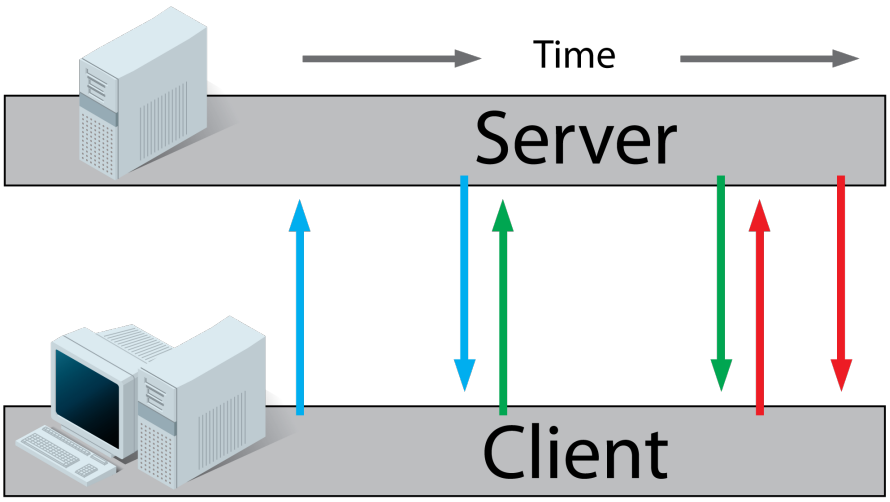
I sistemi di editing collaborativo real-time sono software che permettono agli utenti di modificare lo stesso documento da macchine diverse nello stesso momento. Un sistema di questo tipo deve permettere ad ognuno degli utenti connessi di modificare e vedere le modifiche effettuate degli altri utenti in tempo reale. Esistono due tipi di editor collaborativi: quelli in cui gli utenti possono collaborare in maniera sincrona e quelli in cui cooperano in maniera asincrona.
La collaborazione sincrona è anche definita real-time editing, poiché nel momento in cui un utente effettua delle modifiche al documento, queste sono immediatamente propagate a tutti gli altri clienti connessi al server senza ritardi.
Al contrario, nel caso asincrono gli utenti non hanno modo di collaborare in tempo reale e nel momento in cui vanno a modificare il documento lo fanno a scapito delle modifiche effettuate dagli altri clienti. Ogni singolo utente vedrà una sequenza di eventi differente, anche se in minima parte, rispetto a

quella degli altri utenti. Si necessita quindi di arrivare a una versione convergente del documento che non dipenda dall'ordine o dalla tempistica in cui le operazioni sono state eseguite su di esso.

1.2 Tecnologie Javascript per il real-time

La collaborazione real time richiede l'implementazione di un canale bidirezionale tra client e server in modo da permettere l'editing in parallelo. Javascript offre la possibilità di condividere dati in tempo reale attraverso l'utilizzo di chiamate Ajax (Asynchronous Javascript And XML). Di seguito vengono presentate tre tecnologie per la creazione di un canale in Javascript, che al momento sono le più impiegate nello sviluppo di applicazioni real-time, Ajax long-polling, tecnica base di request-response, server sent events e websockets.[1]
In tutti e tre i casi il client inizialmente effettua una richiesta al server attraverso HTTP, ed esegue il codice Javascript che apre una connessione col server dopo:

Ajax long-polling

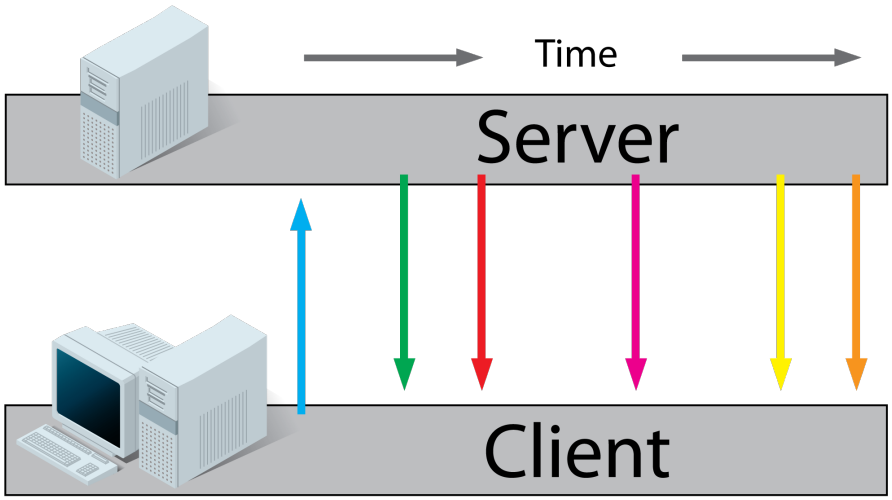


1. Il server non risponde immediatamente, ma attende che ci siano dati disponibili da inviare

2. Quando ci sono nuovi dati disponibili, il server risponde al client inviandoglieli

3. Il client riceve i nuovi dati e inviando successivamente una nuova richiesta al server, fa ricominciare il processo.

Server sent events

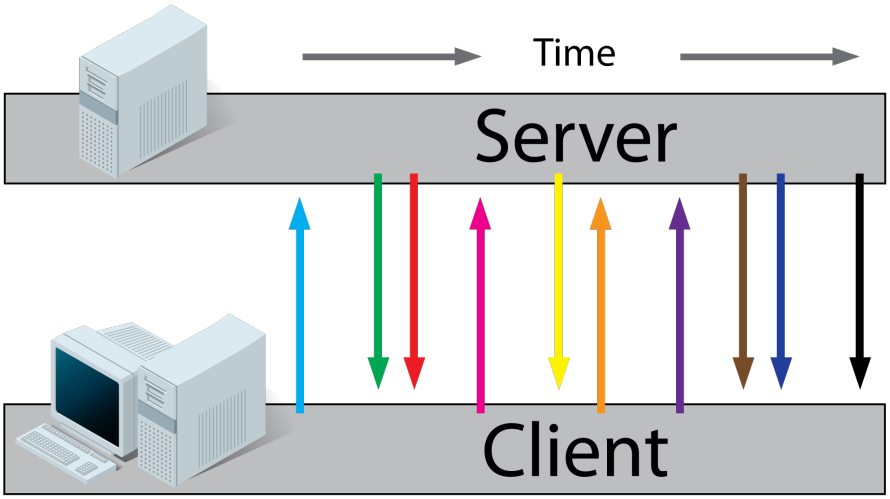


1. Il server risponde con un evento non appena ci sono nuovi dati disponibili.

    - Traffico real time tra server e client

Websockets

1. Server e Client possono inviarsi messaggi a vicenda quando ci sono nuovi dati disponibili (da entrambi gli estremi della comunicazione).



- Traffico real-time da server a client e da client a server

I principali vantaggi di websockets rispetto alle altre tecnologie sono che:

1. WebSocket è una connessione single-socket bidirezionale, full duplex, (comunicazione in cui sia il client che il server possono inviare e ricevere i messaggi l'uno dell'altro contemporaneamente). Con WebSocket, la richiesta HTTP diventa una singola richiesta per aprire una connessione WebSocket e riutilizza la stessa connessione dal client al server e dal server al client.

2. WebSocket riduce la latenza. Ad esempio, diversamente dal polling, WebSocket effettua una singola richiesta. Il server non ha bisogno di attendere una richiesta dal client. Allo stesso modo, il client può inviare messaggi al server in qualsiasi momento. Questa singola richiesta riduce notevolmente la latenza sul polling, che invia una richiesta a intervalli, indipendentemente dal fatto che i messaggi siano disponibili.

Possiamo dire che i WebSocket forniscono un modo molto semplice per fare comunicazioni veloci, solide e molto efficienti tra il client e il server; per questi

motivi questa tecnologia è particolarmente adatta per le applicazioni in cui i dati generati devono essere comunicati rapidamente, e quindi è anche una tecnologia adatta per sistemi collaborativi real-time.

Capitolo 2

Operational Transformation & Change Tracking

In questo capitolo descriverò Operational Transformation, sistema per mantenere la coerenza nella modifica collaborativa di documenti strutturati, e Change Tracking, sistema per la visualizzazione della storia di un documento, e come questi siano fondamentalmente due metodi diversi per rappresentare una stessa sequenza di operazioni su un documento.

2.1 Operational transformation

Operational Transformation (OT), sviluppato dall'idea descritta nel paper di Ellis e Gibbs "Concurrency control in groupware systems" del 1989, è il meccanismo più studiato e più utilizzato nel mondo dello sviluppo software di editor collaborativi per il mantenimento della sincronia di documenti editati in luoghi diversi.
OT viene usato nei sistemi di editing collaborativo, dove il documento condiviso viene replicato per ognuno degli user connessi, così da avere copie differenti del documento per ognuno dei client, le quali necessiteranno di un meccanismo di sincronizzazione per preservare l'integrità del documento finale. La caratteristica più importante dei sistemi che usano OT è il mantenimento della coerenza del documento finale, i meccanismi che permettono

questa proprietà sono classificati in due categorie: ottimistici e pessimistici. Nell'approccio pessimistico si cerca di dare l'impressione ai client che esista un'unica copia del documento all'interno del sistema: la copia di un solo utente è sovrascrivibile mentre quelle degli altri sono accessibili solo in lettura. A differenza del meccanismo pessimistico negli approcci ottimistici, come in OT, si tollera la divergenza momentanea delle copie del documento condiviso tra i client per far convergere ognuna di esse a uno stato finale entro un determinato intervallo di tempo; per questo motivo sono più adeguati all'editing collaborativo.
Gli algoritmi di OT permettono all'utente di modificare una copia del documento localmente senza alcun ritardo di trasmissione e di condividere le operazioni svolte nel file successivamente agli altri utenti connessi al sistema eseguendo le modifiche effettuate su tutte le copie remote del documento. L'applicazione della funzione di trasformazione porta il documento in un nuovo stato, in modo da poter gestire le operazioni eseguite dai vari client. Questo permette di mantenere un'unica copia del documento sul server, il quale diventa la fonte di verità da cui recuperare il documento con facilità, nel caso in cui gli utenti vadano in crash o restino off-line per un lungo periodo. Questa logica obbliga i client ad attendere che il server riconosca le operazioni inviategli, così da mantenere un'unica cronologia delle operazioni server-side senza doverla replicare su ogni singolo documento degli utenti connessi.

2.1.1 Modelli di coerenza nei sistemi OT

Il mantenimento della coerenza del documento finale è una delle caratteristiche più importante di OT. Nel corso degli anni sono stati proposti vari modelli di coerenza.

Causality Convergence (CC)

- *Casualità:* per ogni coppia di operazioni opA e opB se vale la relazione opA $\rightarrow$ opB allora opA viene causalmente prima di opB cioè è stata eseguita prima di essa.

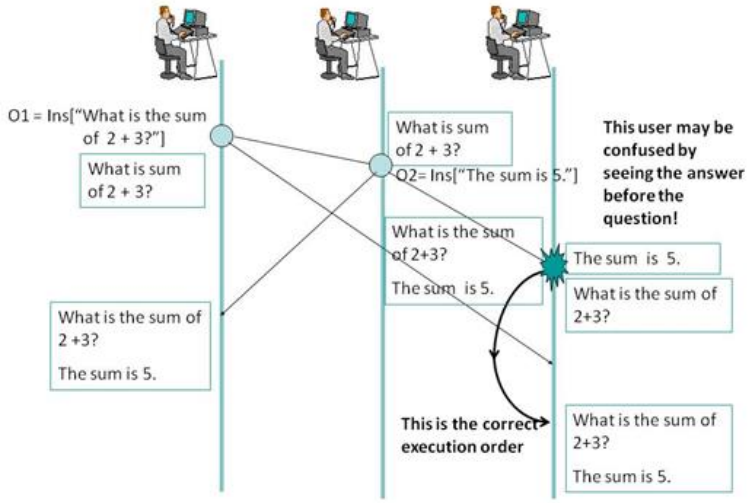


Figura 2.1: Esempio di rispetto della causalità tra tre utenti

- *Convergenza:* quando lo stesso insieme di operazioni viene eseguito da tutti i siti, tutte le copie del documento condiviso sono identiche.

L'esecuzione parallela delle operazioni porterà ogni replica a uno stato divergente, siccome in generale non può esserci commutatività. Ellis e Gibbs proposero l'introduzione di uno state vector per definire la precedenza tra un'operazione e l'altra. [5]

Causality Convergence Intention Preservation (CCI)

- *Casualità:* per ogni coppia di operazioni opA e opB se vale la relazione opA $\rightarrow$ opB allora opA viene causalmente prima di opB cioè è stata eseguita prima di essa.
- *Convergenza:* quando lo stesso insieme di operazioni viene eseguito da tutti i siti, tutte le copie del documento condiviso sono identiche.
- *Conservazione delle Intenzioni:* per ogni operazione op, gli effetti della sua esecuzione su tutti i siti devono rispettare le intenzioni di op.

Questo secondo modello introduce il concetto di conservazione delle intenzioni, che differisce da quello di convergenza. La prima è sempre raggiungibile

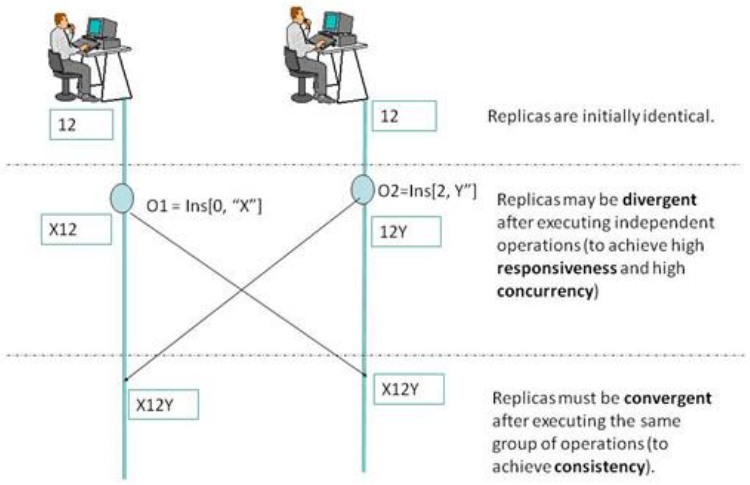


Figura 2.2: Esempio di convergenza del documento

con una semplice serializzazione delle operazioni, mentre la seconda no. Raggiungere la conservazione di intenzioni non serializzabili è la grande sfida che impegna ogni implementatore di meccanismi che adottano OT.[6]

Causality Single Multiple (CSM)

- *Casualità:* per ogni coppia di operazioni opA e opB se vale la relazione opA $\rightarrow$ opB allora opA viene causalmente prima di opB cioè è stata eseguita prima di essa.

- *Effetto di una Singola operazione:* eseguire l'operazione in qualunque stato di esecuzione ha lo stesso effetto di eseguirla nel suo stato di generazione.

- *Effetto di operazioni Multiple:* la relazione che intercorre tra ogni coppia di operazioni resta tale in qualunque stato esse vengano eseguite.

La conservazione delle intenzioni del modello CCI non poteva essere formalizzata, per questo motivo è stato proposto il modello CSM.[7]

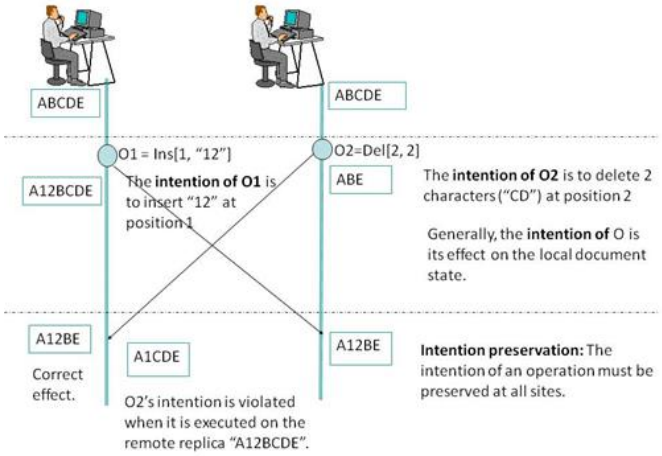


Figura 2.3: Esempio di conservazione delle intenzioni tra due utenti

Convergence Admissibility (CA)

- *Casualità:* per ogni coppia di operazioni opA e opB se vale la relazione opA $\rightarrow$ opB allora opA viene causalmente prima di opB cioè è stata eseguita prima di essa.

- *Ammissibilità:* ogni operazione è ammessa nel proprio stato di esecuzione.

Queste due condizioni implicano convergenza mantenendo un ordinamento basato sull'effetto della generazione delle operazioni. Vi sono ulteriori vincoli imposti da esse, ma ciò le rende più forti della sola convergenza.[8]

2.1.2 Proprietà della trasformazione

Per gestire le operazioni concorrenti la funzione di trasformazione prende in input due operazioni che sono state applicate allo stesso document state da client differenti e restituisce in output una nuova operazione che può essere applicata dopo la seconda, preservando le intenzioni della prima.[9]

Esistono due tipologie di funzione di trasformazione:

- Trasformazione Inclusiva (IT): definita come IT(a,b), trasforma opA in rapporto a opB in modo che l'impatto di opB venga incluso.
- Trasformazione Esclusiva (ET): definita come ET(a,b), trasforma opA in rapporto a opB in modo che l'impatto di opB venga escluso.

2.1.3 Struttura del sistema OT

OT è un sistema formato da più componenti, una strategia di progettazione di questi sistemi consiste nel separare gli algoritmi di controllo di trasformazione di alto livello da quelle di basso livello.[5]
L'algoritmo di controllo richiama un insieme corrispondente di funzioni di trasformazione, che determinano come adattare una modifica rispetto a un'altra in base ai tipi di operazione e alle posizioni. Le responsabilità di correttezza di questi due livelli sono specificate da un insieme di proprietà e condizioni di trasformazione. Diversi sistemi OT con diversi algoritmi di controllo, funzioni e topologie di comunicazione richiedono il mantenimento di diversi set di proprietà di trasformazione.
La separazione di un sistema OT in questi due strati consente la progettazione di algoritmi di controllo generici applicabili a diversi tipi di applicazione con diversi dati e modelli operativi. [8]

2.1.4 Problema del diamante

Il problema che verrà descritto in questa sezione viene chiamato problema del diamante.
Immaginiamo di avere due utenti A e B che modificano contemporaneamente lo stesso documento, in questo caso avremo due operazioni inviate al server che dovrà mantenere l'atomicità delle due operazioni (per evitare condizioni di race-condition), quindi una di queste operazioni verrà applicata per prima. Tuttavia, non appena viene applicata una di queste operazioni, la conservazione per l'altra non sarà più valida.

Figura 2.4: Rappresentazione di due modifiche eseguite da due utenti diversi sulla stessa versione del documento

Considerando il diagramma sopra. L'operazione a è eseguita dal client A e la modifica b è svolta dal client B. Ciò naturalmente presuppone che il server abbia scelto uno dei due utenti come "vincitore" delle race-condition, poniamo questo uguale al client A. Non possiamo semplicemente applicare in modo semplice l'operazione b sul server e a sul client B, altrimenti potremmo ricavare diversi stati del documento. Quello che bisogna adattare l'operazione a all'operazione b e viceversa.
Questo lo si può fare usando una trasformazione T(a,b) che restituisce due nuove operazioni a' e b'.

$$\mathrm{T}(a,b) = (a',b') \text{ dove } a \circ b' \equiv b \circ a'$$

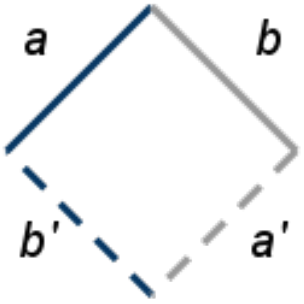


Figura 2.5: Rappresentazione della convergenza tramite l'applicazione del risultato della funzione di trasformazione

Quindi, sul lato client, riceviamo l'operazione b dal server, l'accoppiamo con a per produrre (a', b'), e poi componiamo b' con a per produrre il nostro stato finale del documento. Eseguiamo un processo analogo sul lato server così da ottenere una convergenza dei documenti.[10]

La trasformazione T

Nel problema del diamante è stata definita la funzione di trasformazione T, che risulta:

$$\mathrm{T}(a,b) = (a',b') \text{ dove } a \circ b' \equiv b \circ a'$$

dove a, b, a', b' sono modifiche da applicare ad un documento.
Questa funzione dipende dal tipo di operazioni che si possono eseguire su un file, se consideriamo che si possano fare solo inserimenti ed eliminazioni questa funzione a seconda di a e b dovrà restituire dei risultati diversi. L'output di T dovrà essere applicato da uno o dall'altro utente, ora verrà descritta la sua forma in base a cosa un utente dovrà applicare:

- se entrambe le operazioni sono inserimenti allora l'utente che ha eseguito la modifica in posizione minore dovrà applicare l'operazione dell'altro utente shiftata in avanti della lunghezza del proprio inserimento, mentre il secondo utente dovrà applicare la modifica del primo utente senza che debba essere modificata

- se un'operazione è un inserimento e l'altra è un'eliminazione allora avremmo tre possibili situazioni:

    1. quella in cui l'eliminazione comprende l'inserimento allora all'utente che ha fatto l'eliminazione dovrà applicare un operazione nulla, mentre l'utente che ha eseguito l'inserimento dovrà applicare un'eliminazione appropriatamente estesa per comprendere anche la cancellazione dell'inserimento già applicato

2. quella in cui l'operazione di eliminazione è eseguita in una posizione minore di quella dell'inserimento allora l'utente che ha già eseguito l'eliminazione dovrà applicare l'operazione di inserimento shiftata all'indietro della lunghezza dell'eliminazione, mentre l'altro utente dovrà applicare l'eliminazione senza che sia modificata

3. quella in cui l'operazione di eliminazione è eseguita in una posizione maggiore di quella dell'inserimento allora l'utente che ha già eseguito l'inserimento dovrà applicare l'operazione di eliminazione shiftata in avanti della lunghezza dell'inserimento, mentre l'altro utente dovrà applicare l'inserimento senza che sia modificato

- se sono entrambe operazioni di eliminazioni allora avremmo tre possibili casi:

  1. quello in cui le due operazioni si sovrappongono completamente (cioè una delle due modifiche comprende anche l'altra) allora l'utente che ha eseguito l'operazione che include la modifica dell'altro utente non dovrà eseguire nessuna operazione (operazione vuota), mentre il secondo utente dovrà applicare un'ulteriore eliminazione che comprenderà la parte che non aveva eliminato la sua operazione

  2. quello in cui le due modifiche si sovrappongono parzialmente allora entrambi gli utenti dovranno eseguire un'ulteriore eliminazione che comprenderà la parte che non aveva eliminato la sua operazione

  3. quello in cui le due operazioni non si sovrappongono allora l'utente che ha eseguito la modifica in posizione minore dovrà applicare l'operazione dell'altro utente shiftata all'indietro della lunghezza della proprio eliminazione, mentre il secondo utente dovrà applicare la modifica del primo utente senza che debba essere modificata

Adesso che è stato descritto il comportamento di T mostreremo un esempio del problema del diamante

Esempio

Immaginiamo di avere un documento testuale con la presenza solo della stringa *ci*, se due utenti A e B facessero contemporaneamente un'inserimento alla fine della stringa (cioè in posizione 2), rispettivamente dei caratteri *a* e *o*, ci potranno essere due casi di documento finale: *ciao* e *cioa*.
Questo perché entrambi gli utenti hanno bisogno della modifica dell'altro utente per aver la possibilità di convergere, quindi se entrambi gli utenti facessero l'operazione dell'altro senza che sia stato modificato oteremmo che:

- A farebbe un inserimento di *o* in posizione due nella stringa *cia* ottenendo così *cioa*
- B farebbe un inserimento di *a* in posizione due nella stringa *cio* ottenendo così *ciao*

Nel caso in cui il server ponga l'operazione di A in precedenza a quella di B bisognerà applicare la funzione di trasformazione T (descritta precedentemente) ottenendo in questo caso:

$$\mathrm{T}(< a, 2 >, < o, 2 >) = (< a, 2 >, < o, 3 >)$$

dove $< \ldots, \ldots >$ corrisponde alla modifica descritta come carattere, posizione.
In questo modo si avrà la convergenza del documento testuale sulla stringa *ciao*.
Invece nel caso in cui il server ponesse l'operazione dell'utente B in precedenza a quella di A il documento convergerebbe sulla stringa *cioa*.

Fino ad ora abbiamo è stato descritto OT su documenti testuali, quindi su documenti in cui l'utente non ha la possibilità di fare operazioni sulla struttura del documento. Ora descriveremo come OT possa essere applicato su documenti strutturali senza eccessivi sforzi.

2.2 OT su documenti strutturati

In questo capitolo descriveremmo cosa sono i documenti strutturati e come OT si possa adattare anche a questo tipo di documenti.

Un documento strutturato contiene informazioni addizionali riguardo la propria struttura e su come è aggregato il proprio contenuto, non solo sul proprio layout.
La struttura gerarchica descrive quali operazioni possono essere effettuate su di esso, dove e in quale ordine. A partire dal 1986 con la nascita di SGML (Standard Generalized Markup Language), sono stati derivati HTML (Hypertext Markup Language) e XML (Extensible Markup Language), rispettivamente nel 1993 e nel 1998, come metodi di marcatura di documenti e sequenze di caratteri per definirne le caratteristiche strutturali e di layout. Nello specifico XML, così come HTML, ha una peculiare struttura gerarchica ad albero. Ogni nodo è un elemento che può essere decorato definendo degli attributi su di esso e può avere svariati figli, ognuno dei quali è ancora un nodo che può avere figli, o un nodo di testo, da qui la definizione di struttura ad albero con contenuto misto.
Data questa struttura ad albero, XML (e anche HTML) fornisce una rappresentazione lineare attraverso l'inserimento di tag di apertura e di chiusura, rispettivamente all'inizio e alla fine di ogni nodo.
OT può essere utilizzato anche su questo tipo di documenti, l'utente può così modificare la struttura del documento e non solamente il testo. Quindi ha senso definire due livelli di operazioni: quelle di basso livello che considera i nodi come del testo (usando le operazioni base di OT come se non fosse su documenti strutturati) e quelle di alto livello che considera la modifica dei nodi diversamente da quella sul testo, questo è realizzabile se vengono aggiunte delle operazioni che riguardano la modifica della struttura del documento.[12]

2.3 Alternative a operational transformation

In questa sezione verranno inizialmente descritte Differential Synchronization e Conflict-free Replicated Data Type che sono le alternative proposte ad Operational Transformation e dopo ciò verrà motivata la scelta di OT a discapito delle possibili alternative proposte.

Differential synchronization

Differential synchronization è un algoritmo simmetrico per stabilire la coerenza tra i dati da una sorgente a una memoria di dati target e viceversa, e la continua sincronizzazione dei dati nel tempo; questo algoritmo fa uso di un ciclo infinito di operazioni Diff e Patch. In questo modello ogni utente possiede due versioni del documento, che viene editata e che mantiene l'ultima versione prima delle modifiche. Infatti la differenza (Diff) tra queste due copie viene inviata al server successivamente a ogni modifica effettuata dal client. Il server ha tre versioni del documento: quella corrente, una copia ombra e una copia di backup. L'operazione Diff ricopre due ruoli completamente differenti all'interno del ciclo di sincronizzazione: il primo è di tenere aggiornata l'ombra del Server con il contenuto corrente del documento del Client e dell'ombra del Client che come risultato dovrebbe convergere a tre copie identiche del documento. La seconda operazione è quella più complessa da eseguire e consiste nel tenere aggiornato il documento del Server seguendo le modifiche che vengono effettuate al documento del Client; il file sul Server potrebbe essere stato modificato nel frattempo, quindi la Diff deve aggiornarlo in maniera semanticamente corretta. L'operazione Patch è critica tanto quanto le operazioni del sistema infatti Patch deve esaminare due variabili (potenzialmente in conflitto) quando si tenta di trovare la posizione corretta per effettuare un inserimento. Il primo è trovare il testo con il minimo numero di operazioni elementari di differenza(che corrisponde alla distanza di Levenshtein [3]) tra il testo atteso (basato sul contesto della patch) e il testo effettivo. Il secondo è trovare una posizione ragionevolmente vicina alla posizione prevista della patch per poi applicarla.

Confilict-free replicated data type

I Conflict-free Replicated Data Type (CRDT) sono oggetti che possono essere aggiornati senza grandi sforzi di sincronizzazione, poiché ogni replica può eseguire un'operazione senza doversi sincronizzare con tutte le altre. Tale operazione viene inviata asincronamente e ogni replica applica tutti gli aggiornamenti, possibilmente in ordini differenti. CRDT fanno uso di un

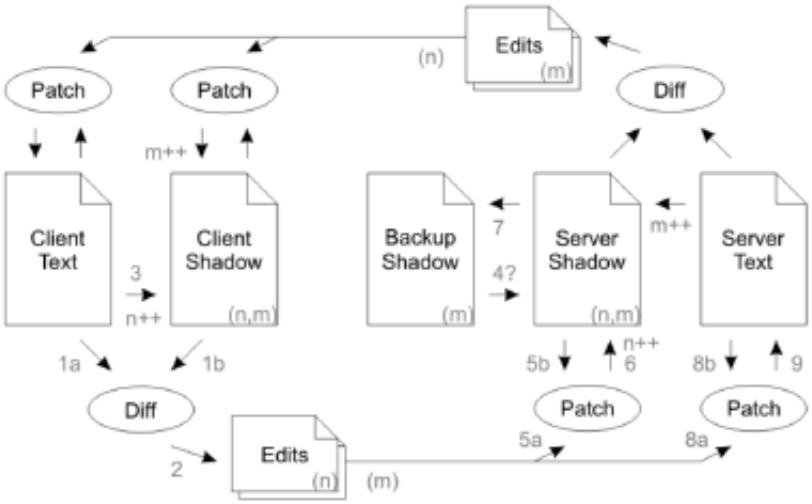


Figura 2.6: è un diagramma di flusso dei dati idealizzato per Differential synchronization

algoritmo che stabilisce la convergenza di tutti gli aggiornamenti conflittuali in background garantendo così la convergenza eventuale se tutti gli aggiornamenti concorrenti sono commutativi e se sono eseguiti da ogni replica eventualmente.[4]

Operational Transformation

In questo paragrafo verrà spiegato brevemente il motivo per cui è stato scelto Operational Transformation invece che gli altri due metodi di gestione della concorrenza e coerenza di un documento.
È stato preferito studiare e sfruttare Operational Transformation come protocollo di gestione della concorrenza e della coerenza del documento finale poiché: Differential Synchronization appesantisce troppo server e client visto che bisogna mantenere rispettivamente tre e due copie del documento.
CRDT invece, richiede l'invio degli update a tutte le repliche comprendendo anche l'intero stato del documento, risultano troppo onerosi a livello computazionale e di occupazione di memoria.
OT a differenza di questi due protocolli, se implementata nella maniera ade-

guata, risulta più flessibile e alleggerisce di molto il carico di lavoro del server, assicurando la convergenza in maniera più efficace.

In questa sezione è stato affrontato, descritto e analizzato a fondo il funzionamento di OT, nei prossimi due capitoli verrà descritto change-tracking e come questi due siano metodi diversi che approcciano lo stesso problema: la rappresentazione delle modifiche in un documento.

2.4 Change tracking

In questo capitolo verrà descritto change-tracking, con questo termine denotiamo il processo di osservazione e visualizzazione delle differenze fra gli stati di una risorsa documentaria durante il suo sviluppo [11]. Questo processo ha significato durante la fase di controllo dei documenti quando si ha bisogno di modificare il documento correggendone gli errori e rinnovando i contenuti per mantenerli aggiornati.
L'uso di sistemi di questo tipo, permette di monitorare la trasformazione del documento tramite i cambiamenti che descrivono l'evoluzione della risorsa fra stati consecutivi.
Consideriamo la fase di stesura di un documento e immaginiamo, in diversi istanti del tempo, di fare delle fotografie. Quando la stesura sarà completa avremo diverse immagini del documento $f_1, \dots f_n$. Lo stato che aveva assunto il documento al tempo t sarà l'immagine f_t. Se volessimo mantenere le informazioni relative a questo processo, possiamo pensare di memorizzare interamente tutti gli stati del documento, occupando così una quantità di memoria pari alla dimensione del documento per il numero di stati.
Una soluzione più efficiente è memorizzare lo stato f_i attraverso le modifiche applicate allo stato f_{i-1}, indicheremo con Δ l'espressione di queste modifiche. Il Δ_i sarà l'insieme delle modifiche per passare dallo stato f_i allo stato f_{i+1} del documento. Attraverso questo procedimento possiamo quindi esprimere l'intera evoluzione di un documento a partire dallo stato iniziale f_0 e applicando iterativamente i Δ_i calcolati fra f_i e f_{i+1}.

$f_1 = f_0 + \Delta_0$

...
$f_i = f_{i-1} + \Delta_{i-1}$

Con questo metodo avremo un modo efficiente di salvare i documenti, anche perchè le dimensioni delle modifiche applicate al documento saranno minori della sua dimensione.
L'importanza dei sistemi di change-tracking è basilare nel supporto per l'analisi delle diverse versioni di un documento, per questo è un'unità fondamentale all'interno di un sistema in cui ogni utente può modificare e creare nuove versioni di un documento.
In generale, un qualunque sistema in cui è importante osservare lo sviluppo dei documenti in funzione del tempo dovrebbe avere un sistema di change-tracking perché questi sistemi permettono una visione rapida e comoda del cambiamento di un documento attraverso il tempo.
Un esempio si può vedere nella figura 2.7 nella quale si può osservare una parte di codice di Change Tracking del web editor TinyMCE. Si nota che questo codice non è formattato e viene visualizzato tutto compatto senza facilitare il riconoscimento del codice dal contenuto del documento. Comunque è possibile riconoscere le operazioni di inserimento e cancellazione che sono delimitate da markup aggiuntivo rispettivamente corrispondente ai tag $< insert >$ e $< delete >$ con attributi come: data, ora, titolo e nome utente.
Un altro modo per rappresentare un Δ in Change Tracking potrebbe essere l'utilizzo di un oggetto JSON strutturato in questo modo:

```
[{
  "id": "edit-00003",
  "op": "del",
  "pos": 100,
  "content": "text"
},{
  "id": "edit-00004",
  "op": "ins",
  "pos": 500,
  "content": "new text"
}    ...  ]
```

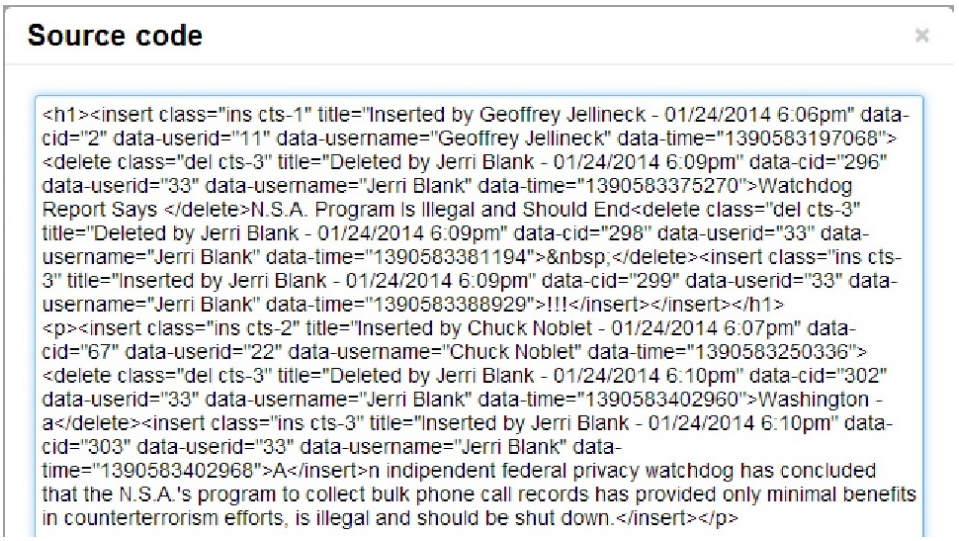


Figura 2.7: Esempio Change Tracking in TinyMCE

Nel prossimo capitolo vedremo come un Δ rappresentato con un oggetto JSON possa essere il punto di incontro fra OT e Change Tracking.

2.5 Metodi a confronto

Scopo di questa dissertazione è sostenere che OT e Change Tracking siano fondamentalmente modi diversi per rappresentare una stessa organizzazione dei dati basata sulla rappresentazione sequenziale delle operazioni su un documento.
OT viene usato per il mantenimento della coerenza di un documento che viene modificato da più utenti.
Change Tracking invece viene utilizzato per mantenere la storia di un documento.
In particolare la somiglianza fra questi due sistemi si nota dal fatto che il Δ fra due versioni di un documento in Change Tracking, cioè le differenze fra due versioni del documento, può corrispondere ad una modifica del documento in OT, cioè ad una singola operazione.
Una possibile soluzione per far in modo che le modifiche effettuate da un algoritmo OT e che il file di mantenimento della storia di Change Tracking

abbiano lo stesso formato è rappresentarle tramite una struttura di dati gerarchica esprimibile con un ogetto JSON; prima di descriverlo però bisogna mostrare le possibili operazioni in OT.
Come si può evincere dalla tesi di D. Montanari in OT sono presenti due operazioni di basso livello (o meccaniche) e sei operazioni di alto livello (o strutturali). [12]
Tutte le operazioni strutturali possono essere ricondotte a un'insieme di operazioni di basso livello. Quelle di basso livello sono:

- INS: corrisponde all'inserimento di testo e markup
- DEL: corrisponde all'eliminazione di testo e markup

Sono scritte con tre lettere per distinguerle da quelle di alto livello. Queste due operazioni agiscono esclusivamente a livello di stringa di testo, quindi comprendendo anche il markup. Le operazioni di alto livello sono:

- Insert: inserimento di testo ed eventualmente di nuovi nodi nel documento
  Esempio:inserimento della stringa *new text*
- Delete: eliminazione di testo ed eventualmente di nuovi nodi nel documento
  Esempio:eliminazione della stringa *text*
- Join: unione di due nodi fratelli e dello stesso tipo. Il nodo unito prende le caratteristiche del primo dei due nodi, e quelle del secondo nodo vengono perse.
  Esempio: eliminazione di che corrisponde all'unione di due paragrafi ($\cdots$ corrisponde ad un paragrafo)
- Split: separazione di un nodo in due nodi fratelli dello stesso tipo. Entrambi i nodi prendono le caratteristiche del nodo unito.
  Esempio:inserimento di che corrisponde alla divisione di un paragrafo in due

- Wrap: annidamento di uno o più nodi con un nuovo nodo. Il contenuto non cambia, ma scende di un livello.
  Esempio: formattare una parola (o di una frase) in corsivo che corrisponde all'inserimento, non adiacente (cioè due inserimento separati), di *e*

- Unwrap: rimozione di un nodo con promozione del suo contenuto al livello del nodo rimosso. Il contenuto non cambia, e sale di un livello.
  Esempio: togliere la formattazione in grassetto di una parola (o di una frase) che corrisponde all'eliminazione, non adiacente, di *e*

Sono operazioni a livello strutturale sequenze di uno o più operazioni meccaniche che hanno un senso proprio sul documento anche se vengono svolte in più fasi o in locazioni diverse. Le operazioni strutturali possono risolversi all'interno di un unico nodo di testo (e parleremo allora di operazioni sul testo) o su strutture di markup complesse (e parleremo allora di operazioni sulla struttura). Le operazioni strutturali vengono svolte atomicamente (ad esempio, un wrap avviene atomicamente anche se corrisponde a due insert indipendenti).
Ora che sono state descritte le possibili operazioni di OT si può descrivere la struttura del documento JSON che racchiuderà le operazioni di OT e Change Tracking.
Ogni operazione meccanica nel file JSON viene descritta in questo modo:

```
{
  "id": "edit-00003",
  "op": "operazione meccanica",
  "pos": 100,
  "content": "testo e/o markup"
}
```

Dove :

- id: è l'identificativo dell'operazione meccanica

- op: corrisponde all'operazione (INS/DEL)
- pos: la posizione dell'inserimento/eliminazione viene contata sulla stringa dei caratteri del testo, considerando anche il markup e contandolo.
- content: cosa viene inserito/eliminato dall'operazione

Mentre le operazioni strutturali (che sono considerate atomiche anche se possono essere formate da più operazioni meccaniche) vengono descritte così:

```
{
  "id": "structural-00026",
  "op": "operazione strutturale",
  "by": "autore",
  "timestamp": "2018-03-10T07:25:23.891Z",
  "items": [{
                        "id" : "edit-00072",
                        "op":"operazione macchina 1",
                        "pos": 512,
                        "content": "testo e/o markup"
                },{
                        "id" : "edit-00072",
                        "op":"operazioni macchina 2",
                        "pos": 512,
                        "content": "testo e/o markup"
                }]
}
```

Dove:

- id: è l'identificativo dell'operazione strutturale
- op: è l'operazione strutturale
- by: indica l'autore della modifica
- times stamp: indica quando è stata fatta l'operazione

- items: è l'insieme di una o più operazioni meccaniche che compongono l'operazione strutturale

Il file JSON è composto da tutte le operazioni strutturali fatte al documento nella forma descritta precedentemente.
Con il file JSON strutturato così come è stato descritto è semplice mostrare che un operazione in OT si riesce a descrivere facilmente.
Quindi è semplice avendo l'insieme delle operazioni eseguite ottenere il file JSON corrispondente. Questo dimostra che con il file JSON si possono rappresentare le modifiche OT e che il file si può usare anche per rappresentare Change Tracking visto che ogni modifica corrisponde a un Δ.
Quindi un file nella forma appena descritta può essere usato sia per descrivere le modifiche fatte in OT sia per rappresentare Change Tracking per questo motivo possono essere definiti modi diversi per rappresentare una stessa organizzazione dei dati basata sulla rappresentazione sequenziale delle operazioni su un documento.

Capitolo 3

DOTT

In questo capitolo descriverò un progetto volto a dimostrare che una modifica fatta in OT corrisponde e può avere la stessa forma di un Δ fra due versioni diverse in Change Tracking, che verrà reso evidente attraverso la realizzazione di una semplice applicazione dimostrativa chiamata DOTT (Document Operational Transformation Tracking)

3.1 DOTT: Caratteristiche e funzionalità

Un editor collaborativo è un'applicazione software collaborativa che consente a più persone di modificare un file utilizzando diversi computer. Esistono due tipi di modifica collaborativa: in tempo reale e non in tempo reale. Nell'editing collaborativo real-time, gli utenti possono modificare lo stesso file contemporaneamente, a differenza nella modifica collaborativa non in tempo reale le modifiche vengono apportate a turni esplicitamente concessi ai vari client.
Gli editori visuali sono caratterizzati da una comoda interfaccia attraverso la quale si possono creare le proprie pagine, il più delle volte hanno anche una sorta di browser integrato per visualizzare l'anteprima della pagine realizzate e constatare se il risultato raggiunto sia ciò che si desiderava o meno. La maggior parte di questi applicativi consente la realizzazione della pagine in due modalità:

- Codice sorgente: si scrive normalmente il codice come se fosse un editor testuale
- Interfaccia utente: si agisce mediante un'interfaccia che genera in background il codice HTML necessario

DOTT è un sistema di editing collaborativo *real-time*, ovvero un'applicazione web in cui più utenti possono modificare contemporaneamente il medesimo documento.
L'interfaccia dell'applicazione è composta da un foglio bianco con una barra menù, dove sono presenti i comandi per la formattazione del testo, l'inserimento di link, elenchi ed altre funzioni. È possibile quindi convertire il testo in titolo e creare una lista.

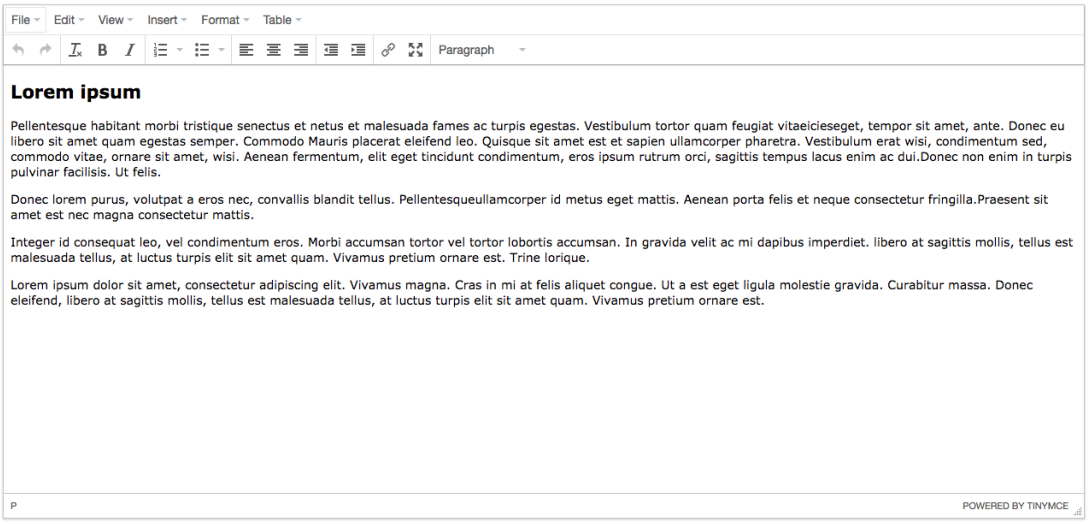


Figura 3.1: Interfaccia di DOTT

Questa applicazione è un editor di testo di markup in quanto le istruzioni di formattazione sono direttamente integrate nel testo e possono assumere la forma di marcatori o tag determinati dal particolare linguaggio utilizzato. Il documento risultante generalmente non mostra le istruzioni utilizzate ma solo le modifiche derivanti da queste.

Esempio: Il codice che fa visualizzare la stringa “DOTT” come un link ad una pagina web (in questo caso, “http://site1821.tw.cs.unibo.it”) è DOTT.

Questo esempio viene mostrato per far capire che le modifiche effettuate da un utente se riguardano lo stile del testo sono tutte modifiche di markup che quindi non riguardano il contenuto del documento. Questo esempio serve anche per far capire che l'editor trasforma le istruzioni di formattazione del testo nella formattazione corrispondente per l'utente.

3.2 DOTT: Architettura e algoritmi fondamentali

La struttura di questa applicazione può essere divisa in due parti: una server-side in cui si gestiscono le modifiche inviate dagli utenti e l'altra client-side in cui si gestiscono le modifiche locali degli utenti.
Nella prima parte della mia applicazione ho deciso di usare un modello in cui sono presenti solo due operazioni: *insert* e *delete* (corrispondenti alle modifiche meccaniche descritte precedentemente). Questa scelta è dovuta dal fatto che tutte le altre modifiche possono essere ricondotte ad un insieme di *insert* e *delete*.
In questo modo in un futuro aggiornamento dell'applicazione, nella storia del documento, si potranno raggruppare o dividere più modifiche per formare una o più operazioni strutturali.

Esempio: un *insert* di *text*   *other text*, corrisponde a un *insert* di *text*, uno *Split* e un *insert* di *other text*. Un *insert* di   corrisponde ad un *Split* perché un paragrafo corrisponde a ... e quindi l'inserimento all'interno del paragrafo di   equivale

a separare in due il paragrafo.
Mentre un *delete* di *text   other text*, corrisponde a un *delete* di *text*, un *Join* e un *delete* di *other text*. Un *delete* di * * corrisponde ad un *Join* perché due paragrafi consecutivi corrispondono a * ... ... * e quindi l'eliminazione di * * equivale all'unione dei due paragrafi.

Quindi un aumento delle operazioni può essere elaborato attraverso le funzioni di trasformazione che ho implementato.
Nella mia applicazione ogni modifica è rappresentata da una tripla formata da:

- tipo di operazione
- posizione
- cosa inserire/eliminare

Indicheremo questa tripla così: *[op,pos,text]*.
Se consideriamo A e B due modifiche, nella forma $A = [opA, posA, textA]$ e $B = [opB, posB, textB]$, e sia B la modifica da adattare in base all'applicazione di A allora le funzioni di trasformazioni sono descritte in questo modo: $\downarrow$ indica la posizione in cui viene fatta l'operazione *insert*, con [....] si indica la porzione di testo che viene eliminata con l'operazione di *delete*. Le scritte in rosso indicano la modifica A e quelle in blu la modifica B. I casi che non vengono discussi sono quelli in cui non vengono modificate le operazioni.

opA - opB

Insert - Insert: Se $posB > posA \,||\, posB == posA$ shifto in avanti l'*insert* B, cioè aumento $posB$ di $textA.length$.

Insert - Delete: Se $posB > posA$ shifto in avanti il *delete*, cioè aumento $posB$ di $textA.length$.
Invece se l'operazione di *delete* comprende la posizione dove verrà fatto

l'*insert* cioè se $posB < posA$ && $posA > fineB$ (dove $fineB$ si può ricavare sommando $posB$ e $textB.length$) verrà aggiornato il *delete* aggiungendo all'interno di esso $textA$
Esempio:
Ciao ma[m ↓ ma] → Ciao ma

Delete - Insert: Se $posB > posA$ e il *delete* non comprende l'*insert* (cioè $posB > fineA$ dove $fineA$ si può ricavare sommando $posA$ e $textA.length$) shifto all'indietro l'*insert*, cioè diminuisco $posB$ di $textA.length$.
Invece se l'operazione di *delete* comprende la posizione dove verrà fatto l'*insert* cioè se $posB < posA$ && $posB < fineA$ viene eseguita solo *delete* e l'*insert* verrà annullata (return null o NoOperation)
Esempio:
Ciao ma[m ↓ ma] → Ciao ma

Delete - Delete: Se $posB > posA$ allora può succedere che:

1. il *delete* A non comprende il *delete* B (cioè $posB > fineA$ dove $fineA$ si può ricavare sommando $posA$ e $textA.length$) allora shifto all'indietro il *delete* B, cioè diminuisco $posB$ di $textA.length$.
   Esempio:
   Ci[ao] ma[mma] → Ci ma

2. che i *delete* si sovrappongano e quindi che il *delete* B elimini una parte presente anche nel *delete* A, cioè che $posB < fineA$ && $fineB > fineA$ (dove $fineB$ si può ricavare sommando $posB$ e $textB.length$), in questo caso il *delete* B verrà ridotto ad un *delete* che eliminerà la parte di testo compresa fra $fineA$ e $fineB$
   Esempio:
   Ciao[ma[mm]a] → Ciao

3. eliminino la stessa parte di codice e quindi che il *delete* B sia incluso nel *delete* A, cioè $posB < fineA$ && $fineB < fineA$, in questo caso il *delete* B verrà annullato (return null o NoOperation)
   Esempio:
   Ciao[ma[mm]a] → Ciao

Altrimenti se $posB < posA$ può accadere che:

1. (analogo al caso due precedente) che i *delete* si sovrappongano e quindi che il *delete* B elimini una parte presente anche nel *delete* A, cioè che $fineB > posA$ && $fineB < fineA$ in questo caso il *delete* B verrà ridotto ad un *delete* che eliminerà la parte di testo compresa fra $posB$ e $posA$
   Esempio:
   Ciao[ma[mm]a] → Ciao

2. (analogo al caso tre precedente) eliminino la stessa parte di codice e quindi che il *delete* A sia incluso nel *delete* B, cioè $posB < fineA$ && $fineB > fineA$, in questo caso il *delete* B verrà ridotto fra in un *delete* che elimina le parti comprese fra $posA$-$posB$ e fra $fineA$-$fineB$
   Esempio:
   Ciao[ma[mm]a] → Ciao

In questa parte dell'applicazione, oltre ad applicare le funzioni di trasformazione alle modifiche che verranno salvate in un file a parte per mantenere la storia, ogni volta che un nuovo client si connette si ricava e si invia a quest'ultimo il documento. Inoltre quando un utente lo richiede gli vengono inviate le modifiche da applicare per arrivare alla versione corrente del documento.
Nella seconda parte dell'applicazione, in cui vengono gestite le modifiche degli utenti, ho deciso di utilizzare l'editor TinyMCE. è stato scelto di utilizzare

questo software al posto di altre piattaforme di editing HTML, come CKEditor, per la semplicità di come si potesse adattare all'applicazione e per la facilità di integrazione con javascript.
TinyMCE è una piattaforma web per l'editing HTML scritta in Javascript. Ha l'abilità di convertire i campi HTML TEXTAREA o altri elementi HTML in istanze di editor.[13]
In questa parte dell'applicazione vengono raccolte tutte le modifiche fatte al documento da un utente e inviate alla parte server-side dell'applicazione che applicherà le funzioni di trasformazione sulle modifiche che verranno poi applicate dal client sul proprio documento.
Per raccogliere tutte le modifiche effettuate da un client TinyMCE offre un supporto per vedere le operazioni di modifica che un utente ha applicato nell'editor. Raccolte queste modifiche l'utente le invia tramite una websoket al server che in base a quando sono arrivate e alle modifiche degli altri utenti le applica al testo presente nel server applicando su di esse, se necessario, le funzioni di trasformazione con tutte le operazioni che non sono ancora state inviate al client. Dopo di che al client vengono inviate le operazioni che non erano ancora state effettuate pronte per essere applicate al suo documento.

Capitolo 4

Valutazione

L'obbiettivo di questo progetto era creare un editor di testo che integrasse i sistemi di OT e Change Tracking, cioè un'applicazione web di editing testuale che avesse un unico metodo per cui più utenti potessero modificare il documento contemporaneamente e mantenere la storia, facendo così in modo che nell'applicazione la forma di una modifica fatta in OT corrispondesse e avesse la stessa forma di un Δ fra due versioni diverse in Change Tracking. Le specifiche di OT da rispettare erano quelle descritte nella tesi di Montanari: cioè che le operazioni potessero essere di basso livello:

- ins corrisponde all'inserimento di testo e markup
- del corrisponde all'eliminazione di testo e markup

a cui tutte le operazioni di alto livello potessero ricondursi, queste operazioni sono:

- insert inserimento di testo ed eventualmente di nuovi nodi nel documento
- delete eliminazione di testo ed eventualmente di nuovi nodi nel documento
- join unione di due nodi fratelli e dello stesso tipo. Il nodo unito prende le caratteristiche del primo dei due nodi, e quelle del secondo nodo vengono perse

- split separazione di un nodo in due nodi fratelli dello stesso tipo. Entrambi i nodi prendono le caratteristiche del nodo unito
- wrap annidamento di uno o più nodi con un nuovo nodo. Il contenuto non cambia, ma scende di un livello
- unwrap rimozione di un nodo con promozione del suo contenuto al livello del nodo rimosso. Il contenuto non cambia, e sale di un livello

che sono operazioni a livello strutturale cioè che riguardano la struttura di un singolo nodo o la struttura di più nodi (cioè operazioni di murkup complesse). [12]
L'integrazione di questi due sistemi è avvenuta creando un file con le caratteristiche descritte nella sezione 2.3:

```
{
  "id": "edit-00003",
  "op": "operazione meccanica",
  "pos": 100,
  "content": "testo e/o markup"
}
```

Questo obbiettivo è stato raggiunto creando l'applicazione descritta precedentemente. Possiamo definire questa applicazione però come una prima versione perché si sono usati solamente due tipi di operazioni OT, quelle meccaniche:

- *insert*: corrisponde all'inserimento di testo e markup
- *delete*: corrisponde all'eliminazione di testo e markup

Quindi le specifiche sono state rispettate solamente in parte visto che inizialmente il set di operazioni da usare era maggiore. Questa scelta però non ha influenzato troppo il risultato perché più *insert* e *delete* uniti formano

tutte le possibili operazioni di alto livello, un'altro motivo per cui l'influenza di questa scelta non molto importante è il fatto che per raggiungere questa specifica basterebbe, quando si vanno a salvare le operazioni sul file di storia vedere se una o più di queste corrispondono a una o più operazioni strutturali (di alto livello).

Capitolo 5

Conclusioni

In questa tesi sono state analizzate approfonditamente le fondamenta teoriche di Operational Transformation e Change Tracking che sono state successivamente confrontate sia teoricamente sia tramite l'implementazione e la discussione dell'editor collaborativo DOTT (Document Operational Transformation Tracking).
Inizialmente si sono descritti in maniera generale i sistemi di collaborazione real-time, le loro caratteristiche e le tecnologie in Javascript che permettono la creazione di un canale bidirezionale per la realizzazione del real-time.
Di seguito sono state analizzate le fondamenta teoriche di OT: è stato analizzato il suo funzionamento, quali sono i modelli di coerenza principali in questi sistemi, le proprietà delle funzioni di trasformazione, la struttura ad alto livello, il problema del diamante, come questo sistema si possa adattare a documenti strutturati. Dopo si sono introdotte e descritte brevemente possibili alternative a OT, come Differential synchronization e Confilict-free replicated data type, dando infine una giustificazione della scelta di OT.
Inoltre è stato mostrata la teoria di Change Tracking, due modi per sviluppare un sistema che lo utilizza e sono stati mostrati due esempi di come si può osservare il codice prodotto da questa tecnologia.
Successivamente è stata riprodotta una rappresentazione di un file JSON come punto di incontro fra OT e Change Tracking, con il fine di dimostrare che i due metodi, che sono effettivamente diversi, approcciano in realtà lo stesso problema che è quello della rappresentazione delle modifiche in un documen-

to.

Da ultimo è stato analizzato l'applicazione web DOTT che è un editor di testo collaborativo che utilizza OT per assicurare la convergenza dei file su tutti i client e si serve del file JSON per rappresentare la storia del documento; questo permette di non dover utilizzare diverse tecnologie per applicare le operazioni e mantenere la storia in un documento condiviso.

Sono giunto alla conclusione che OT e Change Tracking siano fondamentalmente modi diversi per rappresentare una stessa organizzazione dei dati basata sulla rappresentazione sequenziale delle operazioni su un documento.

Bibliografia

[1] T. van Veen. What are long-polling, websockets, server-sent events (sse) and comet? 2016
https://stackoverflow.com/questions/11077857/what-are-long-polling-websockets-server-sent-events-sse-and-comet/12855533#12855533.

[2] N. Fraser. Differential Synchronization.
https://neil.fraser.name/writing/sync/eng047-fraser.pdf.

[3] Levenshtein, Binary codes capable of correcting deletions, insertions, and reversals, in Soviet Physics Doklady, vol. 10, 1966, pp. 707-810.

[4] M. Shapiro, N. Preguiça, C. Baquero and M. Zawirski, "Conflict-free Replicated Data Types". In Xavier Défago, Franck Petit, and Vincent Villain, editors, SSS 2011 - 13th International Symposium Stabilization, Safety, and Security of Distributed Systems, volume 6976 of Lecture Notes in Computer Science, pages 386-400, Grenoble, France, October 2011. Springer.

[5] S.J. Gibbs C.A. Ellis (1989)"Concurrency control in groupware systems. IEEE Transactions on Parallel and Distributed Systems", 18(2):399-407.

[6] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen, "Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. ACM Trans". Comput.-Hum. Interact., 5(1):63-108, March 1998.

[7] Du Li, Rui Li. "A new operational transformation framework for real-time group editors". IEEE Transactions on Parallel and Distributed Systems, 18(3):307-319, 2007.

[8] Du Li, Rui Li. "Commutativity-based concurrency control in groupware". Proceedings of the First IEEE Conference on Collaborative Computing: Networking,Applications and Worksharing, 2005.

[9] X. Jia, C. Sun, D. Chen. "Reversible inclusion and exclusion transformation for string-wise operations in cooperative editing systems". Proceedings of The 21st Australasian Computer Science Conference, 1998.

[10] Daniel Spiewak, "Code Commit - Understanding and Applying Operational Transformation". 2010
http://www.codecommit.com/blog/java/understanding-and-applying-operational-transformation.

[11] C. Del Grosso "Analisi di algoritmi per il rilevamento delle differenze tra documenti XML e integrazione con gli editor". Tesi Unibo Laurea in Informatica 2012

[12] D. Montanari "Operational Transformation su documenti strutturati" Tesi Unibo Laurea in Informatica 2017

[13] Documentazione di TinyMCE: https://www.tiny.cloud/