Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Informatica

Progettazione di CoVE: studio di tecniche e tecnologie per un editor collaborativo con gestione di versioni per documenti strutturati

Relatore:
Chiar.mo Prof.
Fabio Vitali

Presentata da:
Davide Zanini

Sessione II
Anno Accademico 2015/2016

Indice

Capitolo 1

Introduzione

Scopo di questa dissertazione è affrontare la mancanza di strumenti che offrano un ambiente collaborativo che permetta la stesura e il versionamento di documenti strutturati, proponendo una possibile soluzione: CoVE (Collaborative Version-able Editor).

Uno strumento in grado di offrire un ambiente collaborativo con gestione di versione di documenti strutturati, è uno strumento che permette a più utenti la gestione simultanea di testi che possiedono una struttura ben definita, nella quale si può riconoscere un unico nodo padre che si dirama per nodi intermedi fino a raggiungere le foglie. La mancanza di uno strumento simile ha portato alla ideazione e progettazioe di CoVE, che è il modello per un editor che possa supportare e gestire questo tipo di documenti. Sfruttando tecnologie quali Operational Transformation e il Version Management, vuole offrire un ambiente che permetta a due o più utenti di scrivere un documento e di poter navigare tra le sue versioni, in maniera del tutto simultanea.

Esistono una moltitudine di documenti, ognuno con la propria struttura e il proprio scopo, basti pensare alla differenza tra una fiaba per bambini e un libro di scuola. Un autore che scriva su carta ha la completa libertà di gestire lo spazio del foglio secondo le sue

esigenze, di cambiare l'aspetto dei caratteri a piacere e di fare praticamente tutto quello che la mente gli suggerisce. Una delle sfide più importanti che il settore informatico ha affrontato dalla nascita dei documenti elettronici, è stato riuscire a fornire nuovamente questa libertà agli autori. Ma questa libertà dell'autore comprende due problemi da risolvere: la necessità di linguaggi che permettano agli scrittori di strutturare un documento e la possibilità di fornire strumenti che possano leggere e scrivere i documenti creati con tali linguaggi.

Per far fronte al primo problema, la International Organization for Standardization propose nel 1986 SGML (Standard Generalized Markup Language; ISO 8879:1986) il quale, si fonda su due postulati: (1) Il markup dovrà essere dichiarativo, ovvero dovrà descrivere la struttura stessa del documento e altri attributi piuttosto che specificare i processi da effettuare su di esso; (2) Il markup dovrà essere rigoroso in modo tale che vi possano essere applicati gli strumenti che lavorano su oggetti rigorosamente definiti, come programmi e database. Figlio diretto di SGML, HTML (Hyper Text Markup Lenguage) fu creato da Tim Berners-Lee alla fine degli anni '80 e venne proposto nel '91 insieme al WWW. Anche se ebbe un inizio molto timido, questo portò all'avvento del web. Nel 1996 venne poi creato un altro figlio di SGML: XML (eXtensible Markup Language). Questo nuovo linguaggio, rispetto al fratello HTML che si occupa della formattazione e dell'impaginazione, è molto più incentrato sul definire la struttura logica del documento. Il World Wide Web Consortium (W3C) fu fondato nel 1994 da Tim Berners-Lee con lo scopo di sviluppare le potenzialità del web. Questo ente non governativo, nel 2000 fuse HTML e XML in un unico linguaggio: XHTML, che ad oggi ha raggiunto la sua quinta versione XHTML 5. Questo nuovo linguaggio possiede le caratteristiche di entrambi i linguaggi da cui discende, ovvero definisce sia la struttura logica del documento che la sua struttura visiva, denominata layout. Successivamente mi riferirò con il termine documenti strutturati a documenti che siano esprimibili tramite XHTML 5 ben formato.

La risoluzione del secondo problema portò alla creazione di editor di testo, dei quali

uno dei più famosi e utilizzati è indubbiamente Microsoft Word. Nato nel 1981 dalle menti di Charles Simonyi and Richard Brodie, Word è uno dei primi esempi di editor in cui, in una eventuale stampa del documento, venivano mantenute la formattazione e l'impaginazione del testo come quelle visualizzate a schermo. Un termine coniato proprio in quegli anni serve a identificare editor con questa caratteristica: WYSIWYG che è l'acronimo di What You See Is What You Get, ovvero ciò che vedi è ciò che ottieni. Non tutti gli editor però sono uguali o offrono le stesse funzionalità: si va dall'editor che permette di scrivere puro testo senza alcun tipo di formattazione (un esempio può essere Notepad di Microsoft), a quelli che permettono la stesura di documenti complessi e strutturati (Writer di LibreOffice). Con l'avvento di internet si è sentito il bisogno di avere a disposizione questa tipologia di strumenti indipendentemente dalla macchina su cui si lavora. Per sopperire a tale bisogno, sono stati creati editor raggiungibili attraverso internet o applicabili ad una qualsiasi pagina web, quali: TinyMCE[1], CKEditor[2] , Google Docs[3], Office 365[4] e Etherpad[5], solo per citarne alcuni. Questa particolare tipologia di editor web rientra nella categoria WYSIWYG. Possiamo dividere questa tipologia in due grandi categorie: quelli che non permettono un ambiente collaborativo e quelli che lo permettono. Con ambiente collaborativo intendiamo un ambiente in cui vari utenti possano cooperare alla stesura di un documento simultaneamente. I primi due editor citati fanno parte della prima categoria mentre i restanti alla seconda. Tutti questi editor però hanno in comune due mancanze: (1) la possibilità di gestire la struttura stessa del documento che si sta scrivendo e (2) la possibilità di gestire gli stati che questo attraversa. CoVE vuole essere un editor che permetta la gestione della struttura e delle versioni di documenti strutturati in un ambiente collaborativo.

Con il termine "versione" intendiamo il nome che viene associato ad un determinato

[1]TinMCE : https://www.tinymce.com

[2] CKEditor : http://ckeditor.com

[3]Google Docs : https://docs.google.com

[4]Microsoft Word : https://office.live.com/start/Word.aspx?omkt=it-IT

[5]Etherpad : http://etherpad.org/

stato del documento, che si riconosce essere evidentemente differente da quelli precedenti. La necessità di conoscere queste differenze ha comportato la creazione di tecniche per rilevarle, delle quali si parlerà in maniera più approfondita nel capitolo 2.1.2. Queste tecniche sono ciò che sta alla base dei Version Management System (VMS), sistemi appositamente studiati per risolvere il problema della gestione delle versioni. Un VMS si deve perciò occupare di tenere traccia delle varie versioni e delle varianti che può assumere un documento e di conoscere cosa le differenzia. Il primo sistema pensato per questo si chiama SCCS e risale al 1975[Roc75], e non fu progettato per documenti testuali, bensì per i codici sorgenti di un progetto. Col passare degli anni questi sistemi si sono evoluti, specializzandosi nel versionamento di documenti che non fossero di progetto ma testuali e arrivando a supportare anche ambienti di lavoro collaborativi. Vedremo come questo sia successo e quanto sia importate per CoVE nei capitoli 2.1 e 3.2.4.
Gli editor sopra citati in grado di offrire un ambiente collaborativo (ad eccezione di office 365, che essendo un editor proprietario, non ne conosco le tecnologie utilizzate) sfruttano una tecnologia chiamata Operational Transformation (OT). OT è nato nel 1989 con lo scopo di mantenere la consistenza di un documento condiviso tra più luoghi. Per ottenere lo scopo, OT utilizza il concetto di operazione di base, come l'inserimento o la cancellazione di un carattere all'interno di un testo, trasformando quelle che provengo da sistemi di editazione differenti da quello in cui si trova l'utente. Nel capitolo 2.2 si potranno trovare maggiori dettagli riguardo a questo meccanismo. Il funzionamento di base appena espresso può essere applicato a varie tipologie di documenti come: disegni, testo semplice, documenti strutturati o semi-strutturati e modelli di progettazione 3D. La natura del documento condiviso è ciò che aumenta la complessità del sistema OT che si occupa della condivisione, in quanto le operazioni di base che occorrono per tenere traccia delle modifiche, vanno adattate al modello di documento che si vuole condividere: più complesso è il modello, più informazioni saranno necessarie per tracciare le modifiche. Ad esempio, un sistema pensato per un testo semplice è meno complesso di

quello studiato per la condivisione di modelli 3D. Allo stato attuale non esistono sistemi funzionanti che utilizzino un sistema OT pensato per documenti strutturati. L'adattamento di OT ai documenti strutturati è argomento del capitolo 3.2.2.

Ciò che mi ha spinto a iniziare questa progettazione sono stati degli incontri, a cui ho partecipato, tra membri del mio dipartimento e la sede bolognese di Alstom, gruppo industriale francese che si occupa della costruzione di treni e infrastrutture ferroviarie, nel quale per anche solo un progetto, quale può essere quello di un segnalamento ferroviario o di un'intera stazione, devono essere prodotti svariate centinaia, se non migliaia, di documenti, i quali devono passare attraverso una molteplicità di autori e venire convalidati da numerosi controlli dettati da un processo industriale molto rigoroso. In tale sede ho avuto occasione di capire quanto fosse dispendiosa, in termini di tempo e denaro, la stesura di una mole simile di documenti, oltre ai problemi che si ritrovano a dover affrontare gli scrittori degli stessi. Il problema principale è l'assenza di un ambiente di sviluppo collaborativo, ovvero ogni autore è solo a scrivere il documento, potendo ricevere critiche, commenti e note solo attraverso supporti esterni. Un altro problema molto sentito è il dover gestire il ciclo di vita dei documenti (creazione, stesura, redazione, approvazione/-distruzione) attraverso una moltitudine di strumenti esterni. Questo ha portato all'idea che ci fosse bisogno di un sistema unico che permettesse principalmente: (1) la stesura collaborativa di un documento; (2) la comunicazione tra utenti sia in tempo reale che asincrona; (3) una corretta gestione delle versioni dei documenti. In un ambiente industriale come può essere quello di Alstom, la struttura dei documenti appartenenti alla stessa categoria, come quella dei prodotti offerti o dei progetti consegnati o in sviluppo, deve essere la stessa, perciò il sistema che avrebbe dovuto loro offrire aiuto, avrebbe avuto come target principale dei documenti strutturati, che potranno all'occorrenza essere utilizzati anche solo come testo semplice. Dopo svariate ricerche, non si è trovato un sistema che permettesse questo tipo di gestione, perlomeno non su documenti che

avessero bisogno di essere gestiti anche dal punto di vista della struttura stessa.

È stata proprio l'assenza di una piattaforma che soddisfacesse tali requisiti a portare all'elaborazione del modello CoVE. Come si può dedurre dal nome di questo editor, dato un documento, avrà tre caratteristiche chiave: (1) permetterne la stesura simultanea da parte di 2 o più utenti; (2) fornire un sistema di comunicazione che permetta di annotare su tale documento dei commenti riguardanti parti specifiche; (3) fornire un sistema che ne gestisca le eventuali versioni/varianti. Il modello di CoVE nasce perciò con l'obiettivo di dare agli autori uno strumento per migliorare la gestione dell'intero ciclo di vita di un documento, abbattendone così tempi e costi di produzione.

Ciò che ha reso possibile la progettazione di CoVE è stato l'aver trovato un modo per integrare OT con un VMS. Studiando varie tipologie di algoritmi e sistemi basati su OT e vedendo il funzionamento di vari VMS, ho notato un punto di incontro: il mantenimento dei cambiamenti apportati al testo. L'integrazione di queste tecnologie pone le fondamenta a un ambiente di scrittura collaborativo e che possa gestire le versioni dei vari documenti. Ciò che rende unico CoVE è che, mentre gli editor attualmente in circolazione trattano i documenti strutturati come stringhe o sequenze di caratteri, un'implementazione di CoVE ne gestirebbe anche la struttura, rendendo perciò possibili funzionalità che altrimenti non lo sarebbero.

In questa dissertazione si tratterà soltanto dell'architettura e dei principali algoritmi di un tale editor, mentre lascerò la sua implementazione come sviluppo futuro. Nonostante ciò posso però affermare che un sistema simile si potrebbe già espandere permettendo, per esempio, la gestione di documenti diversi da quelli testuali e strutturati.

Capitolo 2

Collaborazione e Versionamento

La scrittura di un documento è un processo che viene affrontato da un singolo o più autori e che passa attraverso vari stadi, che vanno dalla creazione del documento alla sua archiviazione/distruzione a seconda della valutazione finale del prodotto.

Quasi tutte le fasi che attraversa un documento però possono essere fatte in collaborazione, in particolare, le fasi di stesura e revisione. Ovviamente un problema come la scrittura a quattro mani in passato è stato già affrontato, tramite scambi di documenti parziali e incontri fisici tra i vari autori. Ma se pensiamo al caso in cui i diversi autori siano distanti e debbano svolgere il loro lavoro senza problemi in tempo reale, questa situazione crea un problema non indifferente. Inoltre non è detto che il processo di creazione porti a un solo risultato finale, questo potrebbe accadere se durante la stesura ci si accorgesse che, tramite variazioni, i prodotti finali così ottenuti siano adatti a due situazioni simili tra loro. La mancanza di strumenti che permettano la creazione e la stesura di documenti strutturati in ambiente collaborativo, è un problema ancora oggi irrisolto in quanto, gli editor citati nell'introduzione, non permettono la gestione della struttura stessa dei documenti o tanto meno offrono sistemi integrati per la gestione delle versioni che questi possano assumere. Si è perciò costretti ad usare soluzioni non ottimali, per

esempio, scrivendo in ambiente collaborativo usando strumenti esterni per gestire le versioni così ottenute. In letteratura sono presenti soluzioni ad entrambi i singoli problemi, ovvero come ottenere un sistema di gestione delle versioni e come ottenere un ambiente di lavoro collaborativo. La soluzione al primo problema ha portato alla creazione dei Version Management System, ne spiegherò il funzionamento e come ottenerne uno; mentre la soluzione al secondo è ottenuta tramite una tecnologia chiamata Operetional Transformation, la quale si occupa di fornire un ambiente collaborativo per la stesura di documenti .

2.1 Versionamento

Quando si scrive un documento, dal momento della creazione al momento della sua pubblicazione/distruzione(che dipende dalla valutazione positiva o negativa di quest'ultimo), il documento passa attraverso vari stati. Un sistema di versionamento serve proprio a identificare il prodotto nei suoi vari stadi di avanzamento. Ma cosa è una versione? Una versione è un certo nome che associamo ad un certo stato del documento. In questo scenario si è sviluppato quello che si chiama Version Management. Ma essendo che in questa dissertazione siamo interessati a documenti strutturati, perciò esprimibili tramite un linguaggio di mark-up quale XHTML, andrò ad analizzarne una particolare tipologia: quella del XML versioning management. Questo tipo particolare di versionamento è applicabile al linguaggio usato perché XHTML è un HTML che lavora secondo le regole del XML, perciò riuscendo a gestire un XML si può controllare anche un XHTML.

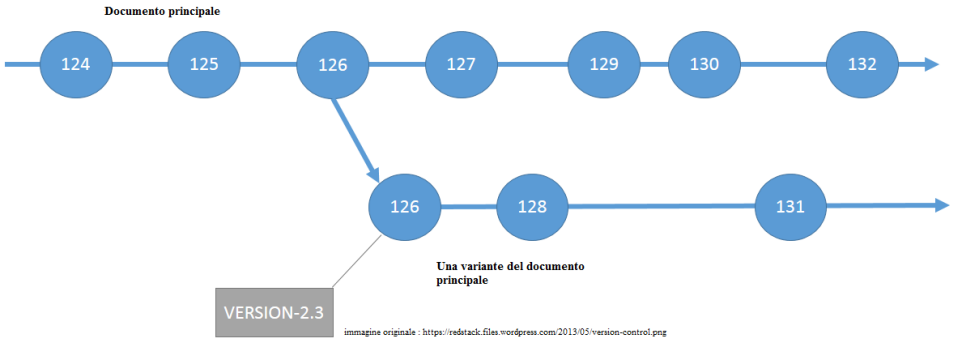

Figura 2.1: Esempio base di variante di un documento

2.1.1 XML versioning management

Il problema di saper gestire diverse versioni di uno stesso documento strutturato è un problema noto e sentito da molti anni, specialmente con l'avvento di internet e dei linguaggi di murk-up, qual'è XML o HTML. In letteratura questo problema è particolarmente sentito per il processo di vita del software nell'area riguardante la documentazione del progetto stesso. Come si può vedere in [Mun01], il problema della documentazione inerente a un progetto non riguarda solo il tipo del documento, ma una serie di variabili come: il tipo del documento, che dovrebbe appartenere a uno standard comune per non essere legati a un determinato tipo/i di programmi per la lettura e scrittura, la struttura del documento e ovviamente i suoi contenuti. L'autore trova in XML uno strumento incredibilmente potente e adattabile per risolvere almeno i primi due problemi(il terzo è sempre inerente al progetto e alla fase che deve descrivere, perciò una soluzione assoluta non può esistere).Ma come sappiamo, i documenti non sono sempre statici, un tipico esempio di un documento che non lo sia è un documento che debba cambiare nel tempo passando attraverso varie versioni. In letteratura per gestire questa particolare tipologia di documenti XML, che avrebbero perciò anche una dimensione temporale, si divide il problema principale in due sotto problemi ovviamente connessi: come individuare e evidenziare i cambiamenti tra le varie versioni e come gestire i vari stati così ottenuti per

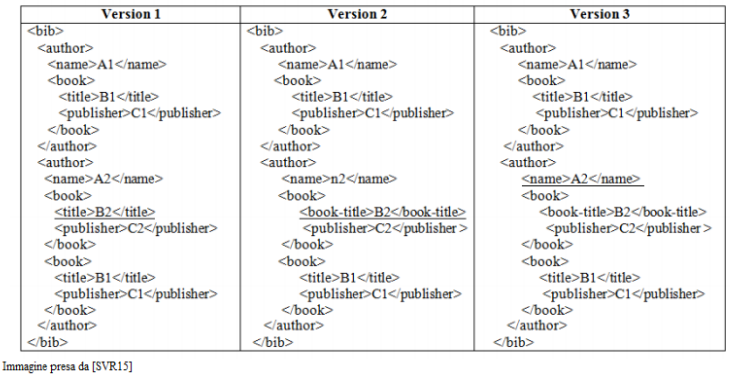

Figura 2.2: Esempio di versioni di un documento con i vari cambiamenti

offrire agli autori degli strumenti per controllare l'avanzamento del documento.

2.1.2 Change detection

Nella stesura di un documento passare di versione in versione fino ad arrivare a quella finale, è il normale ciclo che questo deve affrontare. Quello che differenzia le varie versioni è ciò che l'autore/i del documento hanno deciso di inserire/modificare/eliminare/spostare(delta) tra una versione e la sua successiva, un esempio è dato dalla figura 2.2. Come trovare e identificare queste differenze per capire cosa renda una versione tale, è ciò che sta alla base di un sistema di versionamento, ovvero se non fossimo in grado di identificare l'unicità di una versione non avremmo nulla da gestire. Ma il tracciamento delle modifiche di una versione cambia a seconda della tipologia del documento che può essere strutturato o lineare. Nel secondo caso, il problema è di molto semplificato perché si dovrebbe manipolare le informazioni di un documento unidimensionale (per esempio: una stringa comprendente tutto il testo). Nel mondo odierno però il secondo problema si può dire oramai obsoleto, nel senso che la maggioranza dei documenti sono di tipo strutturato o semi-strutturato proprio per i vantaggi che questi offrono rispetto al tipo unidimensionale. In [Cha96] viene proposto un primo algoritmo per tracciare i cam-

biamenti in informazioni generiche aventi però una struttura gerarchica, che potrebbe perciò essere applicato anche ad un documento XML. Il lavoro presentato nell'articolo ha 4 caratteristiche chiave che identificano i problemi cardine del tracciamento dei cambiamenti:

- Informazioni annidate: rispetto agli algoritmi passati che trattavano di strutture uniformi, in questo articolo si affronta il problema di avere una struttura gerarchica, perciò non interessano solo i cambi ai singoli pezzi della struttura ma anche le relazioni che ci sono tra di essi.

- Identificatori degli oggetti non dati a priori: per massimizzare la generalizzazione del lavoro presentato, si parte dal presupposto che non ci siano identificativi univoci che permettano l'identificazione rapida di informazioni all'interno della struttura tra una versione e l'altra.

- Comparazione tra una versione vecchia e una nuova: tracciare i cambiamenti avvenuti tra una vecchia e una nuova versione.

- Alte prestazioni: far si che l'algoritmo trovato abbia complessità lineare rispetto al numero di cambiamenti.

In [SR15] vengono comparati vari lavori che utilizzano differenti schemi per approcciarsi al problema. L'autore identifica 5 approcci differenti basati su:

- Algoritmo di "Diff".

- Variazioni avvenute nel testo del documento.

- Cambiamenti della struttura del documento.

- Chiavi.

- Classificazione del documento stesso.

Ognuno di questi metodi serve per identificare e differenziare le varie possibili versioni di un documento, aprendo però un altro problema: come gestire le varie versioni e le relazioni tra esse.

2.1.3 Version management system

Ogni cambiamento o serie di cambiamenti significativi in un determinato prodotto, porta quest'ultimo dalla sua attuale versione a una nuova. L'importanza di tracciare questi cambiamenti, potendo così identificare e distinguere le varie versioni, ha fatto si che negli anni si sviluppassero dei programmi o dei servizi (`git:https://git-scm.com/`) apposta per questo o che venissero incorporati in ciò che si pensava averne bisogno(per esempio i word processors). Alcuni dei primi sistemi che si occuparono di cercare di risolvere questo bisogno sono SCCS[Roc75] e RCS[Tic85]. SCCS che è ad oggi considerato obsoleto e fu progettato per il versionamento di codici sorgente, ha 4 caratteristiche chiave nelle quali si possono rispecchiare tutti i VMS:

- Immagazzinamento: tutte le versioni di un modulo sono salvate nello stesso file. Non ci sono duplicazioni di codice nel caso in cui questo sia in comune tra diverse versioni. Tutte le versioni sono accessibili.

- Protezione: nel caso in cui un utente non abbia accesso a determinati moduli o versioni, SCCS si occuperà di monitorare le attività applicando le restrizioni volute.

- Identificazione: il sistema si occuperà di identificare in maniera univoca tutto ciò che verrà caricato, fornendo così un metodo per il recupero delle informazioni volute.

- Documentazione: SCCS registra automaticamente chi ha apportato modifiche e tutte le informazioni utili su di esse.

SCCS è stato il principale sistema di controllo delle versioni fino all'arrivo di RCS, il quale tratta ogni file singolarmente costringendo i vari collaboratori a rimanere sulla macchina ove era mantenuta la storia dei file. Inoltre RCS permetteva solo ad un autore alla volta di modificare un determinato file tramite un sistema di lock/unlock, venendo poi evoluto in sistemi quali: CVS(Concurrent Versions System) e PRCS(Project Revision Control System). SCCS e RCS sono effettivamente inefficaci quando entriamo in un ambito di versionamento di file strutturati con possibilità di ambiente condiviso.
In [FMR14] si discute di come trattare la gestione di documenti XML con multiple versioni, cercando però una soluzione che gestisca al meglio sia tempo che spazio. Per ottenere ciò, vengono usati due concetti:

- Forward deltas: con questo termine di intendono i cambiamenti da applicare alla versione corrente per raggiungere quella successiva. Tutti le variazioni al documento verranno salvate alla fine del documento stesso.

- Reform e numero di versione specifico: il termine "Reform" indica che il sistema ricomporrà il documento completo applicando i delta salvati, dopo un numero di versioni preciso che deve essere fissato sin dall'inizio del sistema. Per fare un esempio, se questo numero dicesse di rigenerare il documento ogni 4 versioni, questo verrà rigenerato alla 5a, alla 9a e così via. Attenzione al fatto che le sotto-versioni(2.1,4.7,...) non faranno azionare il sistema di "Reform".

Usando questo meccanismo il sistema proposto rimane efficiente sia in termini di spazio, perché a ogni n (fissato) di versioni verrammp applicati i delta salvati al documento, sia in termini di tempo perché se si vuole una specifica versione, gli identificatori dei vari delta faranno si che si ottenga la versione voluta velocemente.

Usando il change detection per cotruire un VMS si è in grado perciò di fornire

strumenti agli utenti per:

- Creare e identificare univocamente multiple versioni di un documento.
- Riconoscere le differenze tra le varie versioni.
- Creare varianti di un documento principale.
- Recuperare in maniera efficace versioni (o varianti) passate.
- Sapere chi e quando modifica i documenti gestiti e il corpo della modifica.

Questo apporta notevoli migliorie nella creazione e nella gestione di una moltitudine di documenti strutturati e non.

2.2 Operational Transformation

Operational transformation è una tecnologia nata nel 1989 atta a supportare i controlli sulla concorrenza negli editor collaborativi in tempo reale su documenti operanti su plain text, ovvero serve per mantenere il documento condiviso tra i vari autori in uno stato coerente e corretto nonostante le varie modifiche avvengano da luoghi differenti. In [EG89] non si parla ancora di OT, ma viene affrontato per la prima volta il problema usando un approccio basato su operation transformation, identificandone i punti cardine:

1. La lontananza degli autori.
2. La necessità di essere in tempo reale.
3. La possibile fragilità del documento condiviso, visto il mezzo.
4. I possibili conflitti che potrebbero avere le modifiche apportate al documento dagli autori.

Il nome OT nacque nel 1998, anno in cui venne fondato il SIGCE (Special Interest Group on Collaborative Computing) un gruppo atto a promuovere la comunicazione tra ricercatori sugli editor e su OT, il quale ha portato un enorme contributo a questa tecnologia permettendole di raggiungere il suo stadio attuale. Grazie a OT oggi è possibile collaborare su:

1. editing di testo.
2. editing grafico.
3. editing di html/xml e rich text

e tanto altro, che si differenziano in base al modello dei dati e delle operazioni.
OT è una tecnologia che si basa su vari concetti : le sue operazioni di base, il modello di consistenza, il mantenimento della consistenza, la compressione delle operazioni e la loro

trasformazione. E proprio grazie a queste sue caratteristiche, che tra poco mostrerò un po' più nel dettaglio, che OT si presta molte bene al mondo di internet, potendo gestire i problemi che sorgono da questo potente mezzo e perciò essere applicata alle possibile tipologie di web editor.

2.2.1 Le operazioni di OT

Come si può notare da [SE98],le operazioni di base di OT sono 2:

1. Insert : nella forma Insert[numero, carattere alfanumerico], è l'operazione che dice di inserire alla posizione dettata dal numero indicato il carattere alfanumerico voluto.

2. Delete : nella forma Delete[numero1, numero2], è l'operazione che dice di cancellare alla posizione numero1 un numero di caratteri pari a numero2.

La cosa fondamentale, è che le operazioni rispettino determinate proprietà dettate dal modello di consistenza[Sun98]:

1. Convergenza : Quando un set di operazioni viene portato a termine da tutti gli autori, tutte le copie del documento saranno identiche.

2. Conservazione della causalità : Per ogni coppia di operazioni O_a e O_b se $O_a \rightarrow O_b$, allora O_a verrà eseguito prima di O_b da ogni autore.

3. Preservazione dell'intento : Per ogni operazione O, l'effetto di O su ogni autore è il medesimo per il quale O è stata creata e l'esecuzione di O non può modificare l'effetto di operazioni indipendenti.

Ognuna di queste proprietà deve essere rispettata dall'esecuzione di una operazione, se così non fosse, non si potrebbe ottenere un documento identico tra tutti i partecipanti alla stesura. Il caso del plain text è un caso molto particolare di OT in quanto le operazioni

sopra descritte, bastano al sistema per mappare ciò che può succedere al documento. Nel caso in cui però si voglia applicare OT a qualcosa di più complesso, come ad esempio i documenti strutturati, bisogna espandere il set base delle operazioni con delle nuove come ad esempio Update[Sun04], la quale ha il compito di inviare al server centrale le operazioni fatte perché vengano propagate nel resto del sistema. Il set di operazioni è ciò che sta alla base di OT, bisogna però avere delle funzioni che ne gestiscano gli effetti, in modo tale da sventare eventuali conflitti.

2.2.2 Le funzioni di OT

Le funzioni in un sistema basato su OT, devono quindi manipolare gli effetti che le operazioni di base porterebbero al documento, facendolo così passare correttamente da uno stato documentale iniziale S ad uno nuovo S' identico fra tutti gli utenti nel sistema. Queste funzioni posso essere di due tipi, entrambe le tipologie devono però soddisfare 2 Convergence properties (CP, anche chiamate Transformation properties) e, nel caso si voglia permettere la proprietà di undo, 3 Inverse properties (IP):

- trasformazione inclusiva (IT): tipologia di trasformazione tale per cui, date due operazioni O_1 e O_2, l'effetto di O_2 viene incluso in O_1.

- trasformazione esclusiva (ET): tipologia di trasformazione tale per cui, date due operazioni O_1 e O_2, l'effetto di O_2 viene completamente annullato.

- CP1/TP1: per ogni coppia di operazioni O_1 e O_2 definite nello stesso stato documentale, la funzione trasformante T soddisfa la proprietà CP1/TP1 se e solo se: $O_1 \rightarrow T(O_2,O_1) \equiv O_2 \rightarrow T(O_1,O_2)$.
  Precondizione: questa proprietà è richiesta se il sistema OT vuole permettere che 2 operazioni possano essere eseguite in ordine differente.

- CP2/IP2: per ogni tripla di operazioni O_1, O_2 e O_3 definite nello stesso stato documentale, la funzione trasformante T soddisfa la proprietà CP2/TP2 se e solo se: $T(O_3,O_1 \rightarrow T(O_2,O_1)) = T(O_3,O_2 \rightarrow T(O_1,O_2))$. Precondizione: questa proprietà è richiesta nel caso in due operazioni O_1 e O_2 possano essere IT-trasformate in due stati documentali differenti(portando però sempre al medesimo effetto).

- IP1: dato uno stato documentale S e due operazioni O e $\overline{O}$, abbiamo che S $\rightarrow$ O $\rightarrow \overline{O}$ = S, che vuol dire che la concatenazione delle due operazioni, restituisce lo stesso stato documentale. Precondizione: questa proprietà è richiesta dal sistema OT per ottenere un corretto effetto di undo.

- IP2: questa proprietà dice che la sequenza di due specifiche operazioni O e $\overline{O}$, non apporta alcun tipo di modifica nella trasformazione di una terza operazione ovvero: $T(O_3,O \rightarrow \overline{O}) = O_3$. Precondizione: questa operzione è richiesta se il sistema permette la trasformazione di una operzione O_3 rispetto ad una coppia di operazioni do e undo O $\rightarrow \overline{O}$,una per una.

- IP3: una funzione di trasformazione T soddisfa questa proprietà se e solo se: date due operazioni O_1 e O_2 allora vale che $\overline{O_1}' = \overline{O_1'}$ dove $\overline{O_1}' = T(\overline{O_1}, T(O_2,O_1))$ e $\overline{O_1'} = \overline{T(O_1, O_2)}$. Precondizione: questa proprietà e richiesta se il sistema OT permette che un'operazione inversa $\overline{O_1}$ possa essere trasformata rispetto ad una operazione O_2 concorrente e definita nello stesso stato documentale di O_1.

2.2.3 Mantenimento della consistenza

L'idea per cui questa tecnologia è nata, ovvero far si che il documento che due o più autori condividono, resti sempre identico durante la stesura. Prendiamo per esempio il seguente caso:

Come si può notare dall'immagine 2.3, Autore 1 trasforma inizialmente la stringa abc

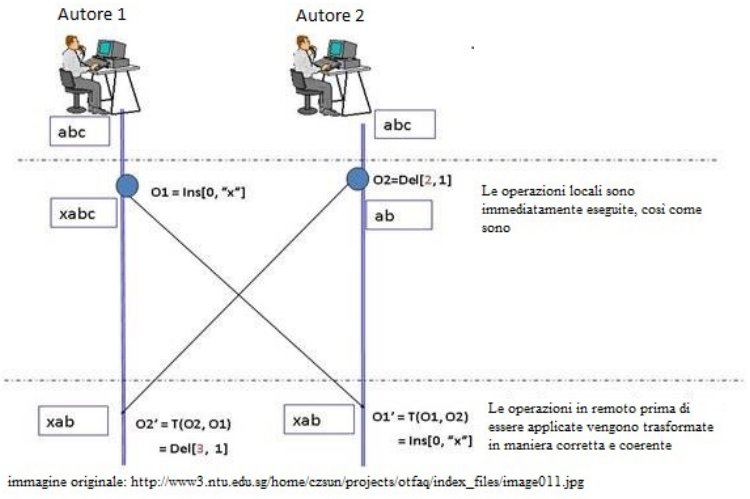

Figura 2.3: Idea alla base della consistenza in OT

in xabc, ovvero genera un'operazione O_1 definita come: Insert[0,"x"]; mentre Autore 2 genera un'operazione O_2 definita come: Delete[2,1]. Entrambi gli autori modificano immediatamente secondo la propria operazione il documento (i.e. : la stringa "abc") e, solo successivamente, l'operazione dell'altro autore verrà presa e trasformata per essere poi applicata nel seguente modo:

1. Caso "Autore 1" : viene applicata la propria operazione, cambiando la stringa iniziale in "xabc". Arriva poi l'operazione generata da Autore 2 che deve essere trasformata di conseguenza, in quanto il documento iniziale è cambiato aggiungendo un carattere in posizione 0, perciò O_2' = T(O_1,O_2) = Delete[3,1]. Eseguendo l'operazione così ottenuta, si ottiene la stringa "xab".

2. Caso "Autore 2" : viene applicata la propria operazione, cambiando la stringa in "ab". Arriva poi l'operazione generata da Autore 1 che, in questo caso, non deve essere in alcun modo trasformata in quanto opera sulla posizione 0, la quale non è stata modifica da Autore 2. Eseguendo l'operazione si otterrà, anche in questo caso, la stringa "xab".

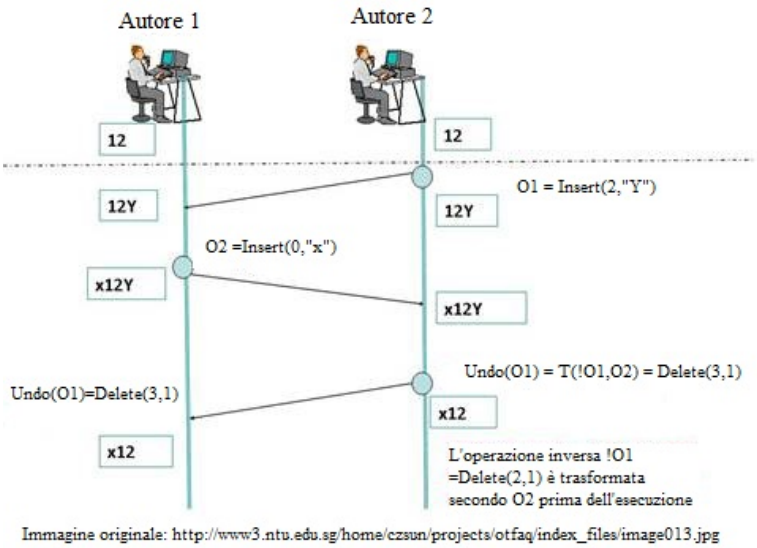

Figura 2.4: Esempio base di undo in OT

Seppur l'esempio sia molto basilare, mostra un aspetto importante: non sempre bisogna trasformare le operazioni per mantenere la consistenza del testo. Un'operazione avrà bisogno di essere trasformata se e solo se applicandola si andrebbe ad infrangere la terza proprietà delle operazioni.

2.2.4 L'undo

Questa funzione di OT fa sì che se un autore decidesse di cancellare una certa operazione, il cui set di appartenenza risulta essere già applicato, allora solo l'effetto di tale operazione verrebbe rimosso. Per rendere tutto ciò possibile ci sono vari algoritmi, alcuni dei quali sono :

- GOTO[SE98] : uno dei primi algoritmi proposti in grado di gestire dati unidimensionali, nato per ottimizzare l'algoritmo GOT riducendone il numero di trasformazioni effettuate. Da solo non riesce a implementare la funzione di undo.

- ANYUNDO[Sun02] : un algoritmo generico per il controllo delle trasformazioni delle operazioni. Insieme a GOTO va a formare uno dei primi sistemi che riesce perfettamente a soddisfare le tre proprietà delle operazioni atte a garantire il undo effect.

- COT (Context-based OT)[SS09] : algoritmo che supporta le operazioni di undo inizializzate dall'utente, con la particolarità di non aver bisogno di soddisfare la proprietà IP1 in quanto non utilizza funzioni di tipo ET.

In ogni caso perché questi algoritmi possano essere efficaci, vi è la necessità di ricordare le operazioni effettuate fino a quel momento dove, a seconda dell'algoritmo scelto, questo viene fatto tramite un history buffer (HB per goto e anyundo) o altri metodi (context-vector per COT). Vorrei fare un esempio del tipico caso in cui si parla di undo per chiarire al meglio cosa si intende con tale termine. Nell'immagine 2.4 si possono vedere 2 autori intenti a lavorare sul documento composto dalla stringa "12". Il secondo autore genera inizialmente una operazione O_1 per inserire il carattere Y in posizione 2 modificando il documento iniziale in "12Y". L'operazione viene propagata poi al primo autore, il quale genera una operazione per inserire il carattere "x" in posizione 0. Il documento diventa per entrambi gli autori la stringa "x12Y". A questo punto il secondo autore decide però di non voler attuare l'operazione da lui creata ma, essendo che il documento si trova in uno stato diverso da quello iniziale, l'operazione che il secondo autore desidera eliminare deve essere trasformata secondo le modifiche avvenute fino a quel momento sul documento. Ecco allora che non si ha un Delete(2,"Y"), che sarebbe l'operazione opposta a quella generata dal secondo autore, bensì un Delete(3,1), in quanto il primo autore ha inserito un carattere in posizione 0. Si ottiene perciò l'effetto undo desiderato, generando la stringa "x12". L'operazione così trasformata dovrà poi essere propagata al primo autore, in quanto bisogna soddisfare la proprietà di convergenza.

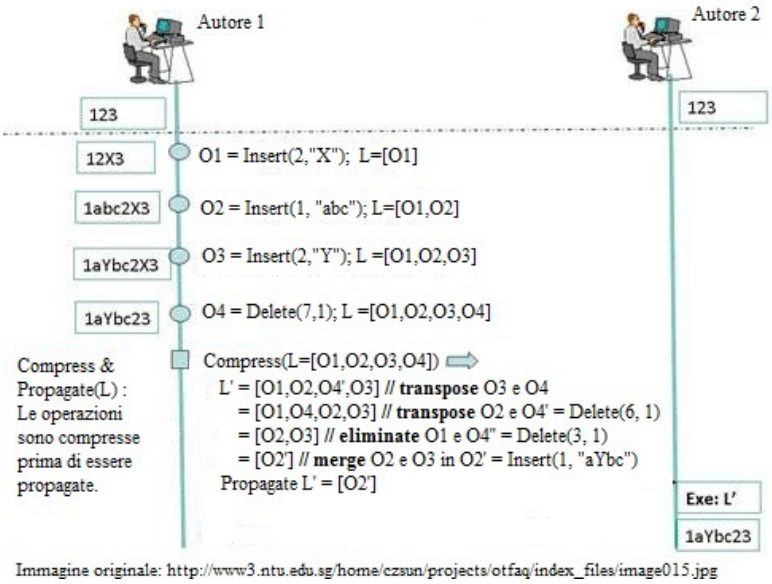

Figura 2.5: Esempio base di compressione delle operazioni in OT

2.2.5 La compressione delle operazioni

In una stesura a più mani di un documento tramite un editor, la probabilità che due o più autori scrivano esattamente nello stesso punto del documento in maniera casuale è veramente bassa, arrivando a renderla nulla se nell'editor fosse possibile vedere la posizione del cursore degli altri autori. La tecnica di compressione delle operazione è stata ideata per ridurre al minimo le operazioni da propagare nel sistema e che gli altri autori dovranno poi applicare al documento. Il fatto che la probabilità che due autori scrivano nello stesso punto possa essere annullata, può togliere un grosso problema da gestire, ma questo caso lo si ottiene solo estendendo la proprietà di preservazione dell'intento agli autori, supponendo che questi vogliano costruire una frase di senso compiuto. Questo lo si potrebbe concretizzare separando i rispettivi cursori, ad esempio ponendo il cursore dell'utente che inizia a scrivere per secondo dietro a quello dell'utente che ha cominciato a scrivere prima. Da notare che anche la posizione del secondo cursore deve essere modifica in base alle operazioni apportate al testo. Per capire il concetto, si guardi

la figura 2.5. Si hanno 2 autori che condividono il documento composto dalla stringa "123", e l'autore 1 genera 4 operazioni da applicare al documento: l'inserimento del carattere "X" in posizione 2(O_1), l'inserimento dei caratteri "abc" in posizione 1(O_2), l'inserimento del carattere "Y" in posizione 2(O_3), la cancellazione di 1 carattere in posizione 7(O_4) e poniamo tutte e quattro le operazioni, in maniera ordinata, in un log L. Ora prima di propagare queste operazioni all'altro autore possiamo comprimerle tramite un algoritmo di compressione delle operazioni di OT[SS02], il quale si occupa di scansionare le operazioni presenti in L, una per una dall'ultima alla prima, esaminare le relazioni che ci sono tra ogni coppia adiacente e decidere se applicare transpose, eliminate o merge. Vediamo i passi che si compierebbero nel nostro esempio:

1. L'operazione O_4 viene scambiata con l'operazione a lei adiacente: transpose(O_3,O_4) = [O'_4,O'_3], dove O'_4 = Delete(6,1) e O'_3 = O'_3, creando così L' = [O_1,O_2,O'_4,O_3]

2. O'_4 viene poi scambiato anche con O_2: transpose(O_2,O'_4) = [O''_4, O'_2], dove O''_4 = Delete(3,1) e O'_2 = O_2, ottendendo così L' = [O_1,O''_4,O_2,O_3]

3. Esaminando le operazioni adiacenti O_1 e O''_4 si nota che non si otterrebbe nessun effetto sul documento tramite la loro applicazione in sequenza, perciò verranno entrambe eliminate, ottenendo così L'=[O_2,O_3]. Da notare che non si sarebbe potute scambiare O''_4 e O_1 in quanto si sarebbe violato il principio di preservazione dell'intento di O_4 (ovvero di cancellare il carattere "X").

4. Come ultimo passo si possono fondere O_2 e O_3 in una unica operazione O'_2 = Insert (1, "aYbc") ottenendo cosi L'=[O'_2] che sarà quello che verrà poi propagato al secondo autore.

Per ottenere una compressione delle operazioni corretta, dovrà essere assolutamente vero che l'applicazione di L e di L' portino il documento esattamente nel medesimo stato.

Uno dei più recenti e avanzati algoritmi di OT è chiamato POT(Pattern-Based OT)[XS16] che possiede questo set di caratteristiche unico:

1. Funziona direttamente seguendo le condizioni e i pattern per evitare la CP2(per conoscerle nel dettaglio, leggere l'articolo).

2. Deriva le relazioni all'interno del contesto e la concorrenzialità delle operazioni senza ricorrere all'utilizzo di uno schema basato su vettori. Questo è molto positivo per sistemi collaborativi su larga scala con un numero dinamico arbitrariamente alto di utenti .

3. Propaga le operazioni locali senza imporre alcun tipo di restrizione il che, aiuta a incrementare la robustezza del sistema e a migliorare l'esperienza dell'utente.

4. Attua tutte le trasformazioni in maniera locale senza necessitare di un server specializzato in questo, ciò comporta una semplificazione della struttura del sistema OT e ne migliora la scalabilità.

5. Lavora con differenti schemi di ordinamento totale rendendo il sistema adattabile ad ambienti client-server o p2p.

6. Ha complessità lineare nel numero delle operazioni concorrenti che è la miglior efficienza, in termini di tempo, per OT.

7. Supporta sia il mantenimento della consistenza che l'undo inizializzato da utente.

Tramite operational transformation si è perciò in grado di fornire ad un gruppo di autori un ambiente di lavoro collaborativo che mantenga coerente tra tutti i partecipanti il documento/i condivisi anche nel caso in cui il mezzo di comunicazione non sia perfettamente stabile (come potrebbe essere internet). Questo, se usato nella maniera corretta,

permetterebbe ai vari autori di scrivere contemporaneamente un documento e perciò di diminuirne i tempi e i costi di produzione.

Capitolo 3

CoVE

La struttura di un documento è un concetto che ci viene insegnato all'inizio della carriera scolastica. La struttura del documento varia a seconda della tipologia dello stesso, ovvero la struttura di un documento scientifico è diversa dalla struttura di un romanzo. Con l'avvento dei documenti elettronici, si è sentito il bisogno di strumenti sempre più efficienti per scrivere/leggere e che potessero replicare o addirittura migliorare quello che si potesse fare su carta, portando però anche notevoli problemi come: la codifica dei caratteri, che ha portato alla creazione di formati quali UTF-16 e UTF-8, e il numero incredibile di formati per documenti che ci sono nel mondo. Il secondo problema è diventato ancora più sentito con l'avvento di internet, la cui soluzione ha portato alla creazione di linguaggi come: HTML, XML e XHTML . Tramite questi linguaggi è infatti possibile descrivere la struttura di un documento e di leggerlo usando oramai un qualunque browser. Ma essendo oramai possibile leggere documenti di varia natura, si è poi arrivati in maniera naturale ad essere capaci anche di scriverli(https://html-online.com/, google docs...). Cercando un editor di questo questo tipo, per ragioni che spiegherò tra breve, mi sono imbattuto in etherpad-lite (https://github.com/ether/etherpad-lite) che è un editor che permette la stesura collaborativa tra vari utenti. Etherpad-lite è figlio diretto delle ricerche su OT, in quanto discende dal contributo della comunità sul progetto etherpad

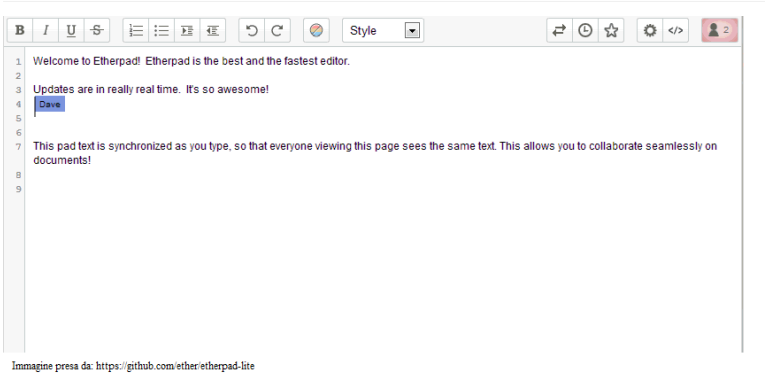

Figura 3.1: Screenshot di una schermata tipica di etherpad-lite

il quale è il progetto googlewave(da cui discende google docs) reso open-source. L'idea alla base del modello CoVE si basa su etherpad-lite(fig: 3.1) e tra poco ne spiegherò le relazioni.

3.1 Nascita e funzionalità

La progettazione di CoVE è nata all'inizio di quest'anno quando il mio relatore mi coinvolse in incontri tra il mio dipartimento e la sede bolognese di Alstom, la quale chiese al dipartimento di aiutarla nel trovare una soluzione per l'abbattimento dei tempi e costi per la stesura di documenti, data l'incredibile mole che l'azienda necessita di produrre. La richiesta principale era che questo strumento permettesse ai vari autori di collaborare sullo stesso documento nello stesso istante. Andando avanti con gli incontri, durante i quali la mia presenza era voluta dal mio relatore perché io capissi e mi rendessi conto dei problemi in questo ambito, ho avuto occasioni di ascoltare le problematiche e le osservazioni dei veri e propri scrittori:

- L'importanza delle varie versioni che attraversa un documento e come queste possano essere sfruttate.
- Il bisogno di comunicare tra i vari autori durante la stesura.
- La necessità di un eventuale redattore di segnalare il tempestivamente possibili errori/modifiche da dover apportare al documento.
- Il fatto di dover riscrivere parti di documenti esattamente uguali a parti di documenti già pubblicati.

Per l'azienda invece era importante sapere chi fossero gli autori del documento, quando è iniziata la stesura e le differenze tra una versione e l'altra. Da qui nacque allora l'idea che questo strumento dovesse essere un editor con determinati requisiti, il quale dovrà:

- Permettere la stesura collaborativa di un documento.
- Saper identificare i vari autori.
- Permettere agli autori di dichiarare il passaggio dalla versione corrente a quella nuova.

- Gestire multiple versioni di un documento.
- Sapere le differenze tra le varie versioni.
- Permettere agli scrittori di comunicare tra loro.
- Permettere a un redattore, anch'egli identificato, di lasciare commenti su parti di documento.
- Permettere di importare parti specifiche di documenti che non siano necessariamente in stesura.

Il mio relatore mi chiese dunque di farmi un'idea suggerendomi tecniche e tecnologie da guardarmi. Fu così che conobbi OT e le tecniche di versionamento. Cercando un esempio funzionante che sfruttasse almeno in parte le tecnologie consigliatemi dal mio relatore, trovai etherpad-lite che, come detto sopra, è un editor web collaborativo.

3.2 Descrizione del nuovo sistema

Pur essendo un ottimo strumento etherpad-lite risulta insufficiente ai fini della mia progettazione, in quanto tratta i documenti come lista di caratteri e perciò utilizza un sistema OT apposta per questo. Con tale soluzione si semplifica il sistema di collaborazione di molto sacrificando però i vantaggi che offre una struttura gerarchica del documento. Conseguentemente a ciò appena detto, etherpad-lite dovrebbe essere adeguatamente modificato per poter soddisfare i requisiti richiesti. Le componenti che necessitano una modifica sono:

- L'interfaccia dell'editor.
- Il sistema OT.

Cambiando il sistema OT sottostante a etherpad, si dovrà rivedere anche tutti i meccanismi ad esso connessi come per esempio, la gestione delle versioni.

3.2.1 Il nuovo editor

L'interfaccia proposta da etherpad-lite è molto valida perché intuitiva, però per supportare un documento gerarchico aggiungerei dei pulsanti(con conseguenti hot keys) per aggiungere capitoli, intestazioni, paragrafi e sotto-paragrafi. Con questi pulsanti oltre ad avere lo stesso effetto visivo che si otterrebbe ora con gli strumenti offerti da etherpad-lite, si otterrebbe inoltre la modifica alla struttura del documento inserendo i nodi opportuni per una corretta gestione della struttura. Questo effetto è necessario per poter gestire correttamente alcune funzionalità richieste per questo editor. Nella sezione I template spiegherò meglio questo concetto. La struttura dei documenti visualizzati sarà perciò data da tag XML, che forniranno meta-dati utili per una corretta gestione da parte di OT, il cui corpo sarà del HTML ben formato che verrà visualizzato all'utente.

3.2.2 Il sistema OT

Il sistema di OT offerto da etherpad-lite, gestendo il documento come descritto sopra, possiede delle operazioni di base e un algoritmo completamente incompatibile con un documento dalla struttura gerarchica. Per gestire una struttura gerarchica con OT, le operazioni di base devono essere necessariamente cambiate per supportare 3 parametri in questo modo: Insert(nodo, posizione, carattere) dove posizione ha la stessa funzione descritta sopra e carattere potrà però assumere anche valori speciali , mentre nodo specifica quale nodo della struttura sarà oggetto della modifica. I caratteri speciali saranno , <h1-6>,<section> e <p> che serviranno a specificare se si sta inserendo un capitolo, una intestazione, una sezione o un paragrafo. Come sistema OT, le cui operazioni di base saranno quelle appena descritte, verrà utilizzato TIBOT 2.0[XS16] in quanto utilizza l'HB che sarà fondamentale per la gestione delle versioni.

3.2.3 TIBOT 2.0

Questo particolare algoritmo per essere utilizzato, richiede che i vari autori comunichino tramite un mezzo affidabile e che garantisca un servizio di tipo FIFO. Per identificare un'operazione viene usato un meccanismo di marcatura oraria in questo modo:

- Ogni sistema collegato deve mantenere un timer locale che incrementi il valore di un Intervallo di Tempo (TI, per non confonderlo con la tipologia delle funzioni). Ogni sistema deve incrementare tale TI allo stesso modo, ovvero l'insieme dei valori assumibili da questa variabile deve essere lo stesso per tutti i sistemi, senza però aver necessità che questi siano sincronizzati.

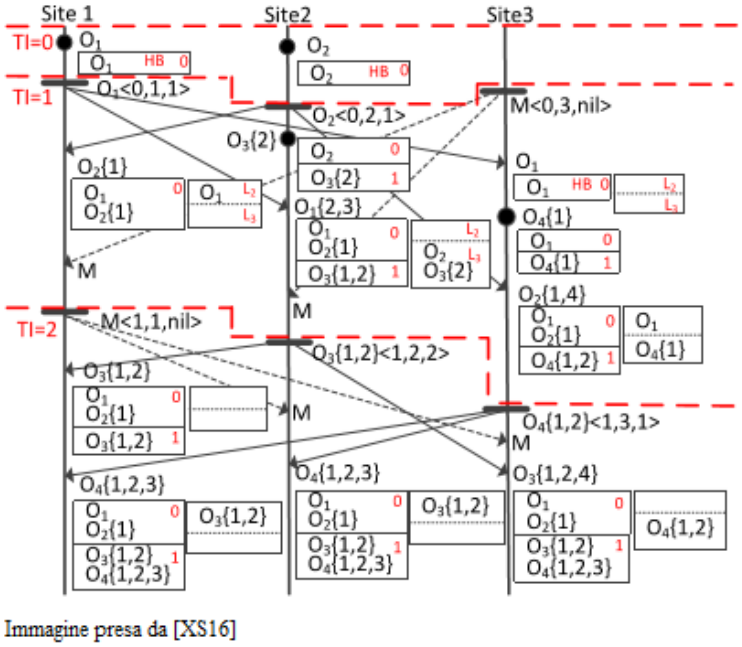

Figura 3.2: Immagine esemplificativa di TIBOT 2.0

- Quando un'operazione viene generata, viene marcata con una tupla definita come <TI, IL, SN > dove TI è l'intervallo di tempo, IL è l'identificatore del sistema sul quale si sta operando e SN è il numero sequenziale dell'operazione a livello locale.

Questo permetto perciò di definire uno schema di ordinamento totale basato su TI, dove date due operazioni O_a e O_b con rispettive marcature T_a e T_b varrà che $O_a \rightarrow O_b$ se e solo se:

- $T_a.TI < T_b.TI$; oppure
- $T_a.TI = T_b.TI$ e $T_a.IL < T_b.IL$; oppure
- $T_a.TI = T_b.TI$ e $T_a.IL = T_b.IL$ e $T_a.SN < T_b.SN$.

Avendo definito lo schema, possiamo ora vedere in dettaglio come TIBOT 2.0 si comporta in locale e in remoto.

Processo locale

Quando O_x viene generata:

1. Esegue immediatamente O_x e la marca con T_x.
2. Salva O_x nell'history buffer(HB) e aspetta fino a quando: (1) il timer locale è avanzo fino ad avere un valore più grande di T_x.TI; e (2) O_x è stata trasformata da una serie di operazioni in remoto, denotate come L_1, che includeranno tutte le operazioni con TI inferiori a T_x.TI(funzioni di tipologia IT); indichiamo ciò con $LT(O_x,L_1) = O_x\{ L_1\}$.
3. Propaga $O_x\{ L_1\}$ con la marcatura T_x.

Ad ausilio di questo si usano 2 schemi per la propagazione delle operazioni:

1. Tutte le operazioni compiute localmente nello stesso intervallo di tempo saranno compresse e propagate insieme.
2. Se non vengono eseguite operazioni locali durante l'intervallo di tempo, verrà propagato un messaggio speciale <TI,IL,nil>.

Processo remoto

In remoto $O_x\{ L_1\}$ viene gestito seguendo questo preciso ordine:

1. Aspetta fino a quando: (1) il timer locale sia più grande di T_x.TI; (2) le operazioni locali con un TI uguale a T_x siano state propagate; e (3) tutte le operazioni precedenti a $O_x\{ L_1\}$ siano state eseguite localmente.
2. Trasforma $O_x\{ L_1\}$ secondo una sequenza di operazioni nel HB, chiamata L_2, che include tutte le operazioni con TI uguale a T_x.TI e precedenti a $O_x\{ L_1\}$; questa trasformazione può essere espressa come $LT(O_x\{ L_1\}, L_2) = O_x\{L_1,L_2\}$.

3. Salva $O_x\{L_1,L_2\}$ alla fine di L_2 nel HB.

4. Trasforma $O_x\{L_1,L_2\}$ simmetricamente con una sequenza di operazioni nel HB, denotata come L_3, che è successiva a $O_x\{L_1,L_2\}$; questa trasformazione può essere espressa come $SLT(O_x\{L_1,L_2\},L_3) = (O_x\{L_1,L_2\}, L_3\{O_x\})$.

5. Esegui $O_x\{L_1,L_2,L_3\}$; sostituisci L_3 con $L_3\{O_x\}$ nel HB.

3.2.4 La gestione delle versioni

Modificando il sistema di OT però cambia in maniera forzata anche la gestione delle versioni. Etherpad-lite prevede un pulsante per dichiarare una nuova versione e tenerne così memoria. La mia idea è quella di espandere questa funzione permettendo oltre alla dichiarazione di una nuova versione, la creazione di una variante del testo tramite un altro pulsante. Entrambe le operazioni verrebbero registrate tramite una nuova operazione di base: Setversion(branch). Definire una nuova operazione di base è fondamentale per tenere traccia di quel che si sta facendo al documento per, nel caso, doverlo annullare. Questa nuova operazione prende un solo parametro che verrà settato a seconda del pulsante premuto: true se si è scelto di creare una variante, false altrimenti. Il nome della nuova versione verrà automaticamente assegnato secondo una semplificazione della regola che potremmo chiamare "versionamento a L", che mi è stata mostrata dal mio relatore e ha la particolarità di essere pensata in una situazione dove documento principale e variante non passano fare merge. Questa dinamica la si può pensare per la natura dei documenti testuali: se decido di creare una variante è presumibile che lo si faccia perché questa versione abbia un obiettivo diverso da quello principale. In questo modo nel momento in cui si crea una variante, si crea un nuovo documento con un nuovo history buffer che parte dalla creazione della nuova versione, ma il cui inizio punta al Setversion all'interno del history buffer del padre(per riuscire a risalire al documento principale). In questo modo creiamo un ambiante in cui le differenze tra due diverse versioni sono tutte

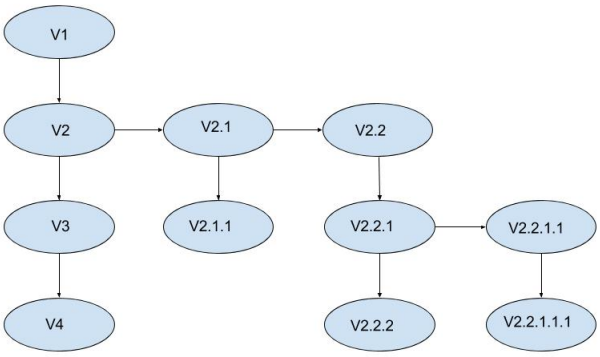

Figura 3.3: Esempio della numerazione a L

contenute negli history buffer e , per generare a video la versione richiesta, basta scorrere dal basso verso l'alto l'HB della versione desiderata fino al raggiungimento dell'origine del HB del documento padre. La generazione di una nuova versione principale verrà consentita solo al creatore del documento, in modo tale da evitare creazioni contemporanee ed erronee di nuove versioni(nel senso che si avrebbe un potenziale salto di N versioni, dove N rappresenta il numero dei partecipanti).

3.2.5 Il versionamento a L

In questa sezione vorrei spiegare più in dettaglio questa regola secondo la quale si assegneranno i nomi alle varie versioni. Partiamo dal concetto di versione e variante espressi precedentemente. La numerazione procede secondo una rappresentazione a L, ovvero si vede lo schema come un albero dove la radice contiene il valore più basso e scendendo

verso le foglie il valore aumenta e dove i nodi intermedi possono estendersi solo verso destra. Per fare un esempio prendiamo un documento senza varianti. Progredendo tra le varie versioni, il nome del documento varierà in questo modo: documento_V1, documento_V2, documento_V3 seguendo la sequenza dei numeri naturali. Ora pensiamo di creare una variante nella seconda versione del documento. Il nome di questa nuova versione sarà documento_V2.1 Da questo momento, avanzando da questa versione del documento(2.1) non si progredirà più secondo una sequenza naturale sulla prima cifra, bensì si otterrà documento_V2.1.1. Il fatto di aver creato una variante però non influirà sui nomi da assegnare alla versione padre. Inoltre se volessimo creare una nuova variante sempre della seconda versione, questa si chiamerebbe documento_V2.2 che se venisse portata ad una nuova versione sarebbe documento_V2.2.1. Pensiamo ora di creare una variante di documento_V2.2.1, questo sarebbe documento_V2.2.1.1 che progredendo diverrebbe documento_V2.2.1.1.1. Perciò si può notare come, trasformando l'insieme delle versioni in un albero, il che è possibile perché non è permesso fare merge tra le varie versioni differenti, le cifre dispari ci dicono quanto muoverci in basso mentre le cifre pari, se presenti, quanto muoverci a destra per raggiungere la versione desiderata. La figura 3.3 mostra ciò appena descritto.

3.2.6 I template

Nella creazione di documenti seriali, il riutilizzo di parti di documenti già scritti può essere molto importante, in quanto può capitare di dover spiegare o mostrare esattamente la stessa cosa in differenti contesti. Per supportare ciò, ho pensato di espandere il sistema OT con un ulteriore operazione di base: la Import(nodo, posizione, nome, template) dove nodo e posizione hanno lo stesso significato che assumono nelle altre operazioni di base adattate, nome indica il nome del documento da cui si vuole prelevare e template

indica il pezzo di testo che si vuole importare. Questo template sarà definibile attraverso un particolare tag XML <template>, che avrà come attributi: il nome che si desidera dare al template, con l'unica restrizione che il nome del template sia unico all'interno del documento; la tipologia del template; l'autore del template. I template potranno essere di 3 tipi:

1. Template non modificabile. Questo tipologia di template dice che l'autore che deciderà di importare tale parte, non potrà in alcun modo modificarla. Questo tipo è pensato per parti di documenti che debbano essere particolarmente precise e dettagliate e il cui autore non intende concedere alcuna modifica.

2. Template modificabile ma le cui modifiche non saranno propagate. Questo particolare template può essere modificato da chi lo importa, ma tali modifiche rimarranno esclusivamente nel documento in cui questo template è stato importato.

3. Template completamente modificabile. L'ultimo tipo ti template permette di essere modificato da chi lo importa e inoltre tali modifiche verranno propagate al documento originale.

Il nome dell'autore è importante per quanto riguarda le prime due tipologie, in quanto permette di contattare l'autore per segnalare eventuali errori nel template o per offrire suggerimenti.

3.2.7 I commenti

Etherpad-lite prevede una chat in tempo reale tra i vari partecipanti alla stesura, il che è un'ottima funzionalità per agevolare la comunicazione all'interno del team di autori.

Per permettere anche una comunicazione asincrona, ovvero un commento da parte di un eventuale redattore che non partecipa alla stesura e che non sia sincronizzato con gli autori, ho pensato ad un meccanismo azionabile tramite un pulsante sull'editor, che utilizzi i tag <spam> del HTML e un tag XML <commento>. Per lasciare un commento si dovrà selezionare la parte del testo al quale si è interessati, spingere poi il pulsante per dichiarare il commento il quale farà uscire una finestra di dialogo in cui inserire il commento vero e proprio che verrà salvato nel tag XML. Se la selezione coprisse più nodi, si creeranno tanti <spam> quanti i nodi coperti dalla selezione, fino ad avere il completo avvolgimento del testo. Perciò le due tipologie di tag saranno in rapporto di n a 1. Gli autori potranno segnare il commento come risolto oppure nasconderlo se indesiderato.

3.2.8 Le funzionalità valide di etherpad-lite

Alcune delle funzionalità offerte da etherpad lite sono indipendenti dal sistema OT sottostante. Queste funzionalità sono:

- Un sistema di accounting che permette di identificare i vari utenti.
- Un metodo di importazione ed esportazione in altri formati diversi dal HTML.

L'unica cosa è che queste funzionalità andrebbero estese: la prima andrebbe resa obbligatoria per accedere al sistema tramite un id fissato e significativo, la seconda dovrebbe comprendere anche il formato .doc(estensione dei documenti creati con Microsoft Word), in quanto è ancora uno dei formati più diffusi. Per la seconda espansione, si possono usare le librerie di libreoffice che permettono di passare da HTML a PDF a DOC e viceversa, andando però a gestire il codice HTML cosi ottenuto come tale e non come stringa.

Capitolo 4

Valutazioni

Essendo che Etherpad-lite è stato l'editor che più ha mi ispirato alla progettazione, la seguente tabella mostra le principali differenze che avrebbe con una eventuale implementazione di CoVE:

	Etherpad-lite	CoVE
Ambiente collaborativo	X	X
Autenticazione degli utenti	X	X
Gestione della struttura del documento		X
Chat in tempo reale	X	X
Commenti su porzioni specifiche di testo		X
Gestione delle versioni	X	X
Gestione delle varianti		X
Importazione di template		X
Importazione ed esportazione di documenti esterni	X(con plugin dedicati)	X

Come si può notare, CoVE migliorerebbe alcuni aspetti di Etherpad-lite. Questo è dovuto al fatto che il modello CoVE, rispetto a quello di Etherpad-lite, vede i documenti

strutturati nella loro interezza.

CoVE vuole essere un sistema capace di offrire un ambiente collaborativo con un sistema di versionamento integrato pensato appositamente per i documenti strutturati. Uno degli aspetti fondamentali è l'interfaccia dell'editor. Questa permetterebbe tramite i pulsanti sopra descritti, la gestione della struttura stessa del documento, se così non fosse, si avrebbe un editor esattamente come gli altri. Il fulcro del progetto è l'integrazione tra un Operational Transformation e un Version Management Sytstem studiati appositamente per documenti dalla struttura gerarchica, in modo tale che queste due tecnologie ne possano condividere le modifiche apportate dagli utenti. Questo avviene tramite la condivisione del componente chiamato History Buffer, il quale ha il compito di memorizzare tutte le operazioni che verranno applicate al testo. Con operazioni intendiamo: l'inserimento e la cancellazione di un qualsiasi carattere, la dichiarazione di una nuova versione/variante e l'importo di un template. I commenti sul testo risultano completamente ininfluenti, in quanto non porterebbero loro stessi dei cambiamenti alla struttura o al contenuto del testo, perciò sarebbero considerati a tutti gli effetti dei metadati applicabili ad esso. Perché tutto ciò possa portare benefici, deve essere usato nella maniera corretta perché un ambiente collaborativo comporta anche dei rischi, in quanto un team che decidesse di utilizzare un ambiente simile, necessiterebbe di una ottima capacità organizzativa a priori, in modo tale che i vari membri del team non si intralcino a vicenda, comportando in questo modo un peggioramento della produttività. Il cambio del modello di documento porta ad un aumento della complessità globale del sistema, in quanto la possibilità di gestire la struttura stessa del documento, implica di fatto la trasposizione del documento ad un albero. Per riuscire ad applicare le operazioni ad una struttura tale, queste devono essere trasformate per indicare anche il nodo dell'albero affetto dalle modifiche quindi, oltre al costo per applicare l'operazione, si aggiungerà quello per la ricerca del nodo voluto. L'algoritmo TIBOT 2.0, che gestirà la propagazione e le

trasformazioni delle operazioni, ha complessità pari al numero di operazioni effettuate nell'arco dell'incremento dell'Interval Time, ovvero $O(n)$.

Tramite l'ambiente offerto dal modello CoVE sarebbe perciò possibile ottenere un abbattimento dei tempi, e di conseguenza anche dei costi, per la produzione di un documento.

Capitolo 5

Conclusioni

Il problema principale trattato da questa dissertazione è la mancanza di uno strumento che offra un ambiente collaborativo per la stesura di documenti gerarchici e che ne gestisca le varie versioni. CoVE si offre come possibile soluzione a questo problema in quanto modello di editor che offre un ambiente collaborativo con gestione di versioni integrato. Questo è l'aspetto che lo rende unico nel suo genere. Tale caratteristica la si ottiene tramite l'integrazione di due tecnologie: Operational Transformation e Version Management System. La prima occorre per creare un ambiente di lavoro collaborativo, la seconda gestisce le varie versioni che un documento può assumere. Le due hanno un punto in comune, il dover tracciare e tenere memoria delle modifiche che vengono apportate al testo. Perciò trovando un sistema per condividere la memoria delle modifiche, diventa possibile ottenere una gestione delle versioni espansa (nel senso che diventa possibile gestire anche le varianti) direttamente nell'ambiente collaborativo. La soluzione che ho trovato risiede nell'utilizzo dell'algoritmo TIBOT 2.0, il quale utilizza una particolare componente: l'history buffer. Questa ha il compito di memorizzare le operazioni che sono state applicate al testo, e condividerle per creare un VMS che possa essere gestito facilmente da utente.

Il difetto più grande di CoVE è la complessità che lo contraddistingue dai modelli alla base degli editor in circolazione. Questo aumento è in conseguenza all'aver reso più complesso il sistema OT sottostante. Se trattassimo i documenti come stringhe si otterrebbe una notevole semplificazione del sistema globale, ma si perderebbero alcune funzioni quali: l'importo dei template, i commenti specifici e il versionamento espanso. Nella progettazione ciò che ho trovato più arduo comprendere è il funzionamento di OT. Ad una prima analisi potrebbe sembrare una tecnologia semplice, e sotto certi punti di vista si può dire che sia intuitiva, ma questo lo si può pensare solo delle versioni di base. Nel momento in cui si entra nel mondo gerarchico (che può essere pensato come un albero di stringhe) questo si complica enormemente, motivo per cui non ci sono editor che utilizzino tali tecniche. Grazie alle sue caratteristiche, CoVE risponde pienamente alle esigenze espresse dagli scrittori incontrati. Una sua implementazione offrirebbe un ambiente di lavoro collaborativo permettendo a 2 o più utenti di scrivere un documento simultaneamente. Se usata correttamente, questa funzionalità sarebbe in grado di abbattere il tempo della stesura. Il sistema di versionamento studiato permetterebbe la creazione di versioni e varianti in modo intuitivo ed efficiente, mentre il metodo dei template fornirebbe la possibilità di non dover riscrivere parti di documenti già scritti, abbattendo ulteriormente il tempo necessario alla stesura. La comunicazione offerta da CoVE velocizzerebbe anche il processo redazionale, permettendo anche in fase di stesura di ricevere consigli o modifiche direttamente sull'opera. Andando perciò a migliorare la gestione della creazione/distruzione, della stesura e della fase redazionale di un documento, penso che CoVE sarebbe un ottimo strumento per gestirne l'intero ciclo di vita e, abbattendone i tempi, si ha come conseguenza che si riuscirebbe anche a diminuirne i costi. Uno sviluppo fondamentale sarebbe riuscire a concretizzare questo modello ma è bene tenere in considerazione che, data la vastità e la complessità del sistema proposto,

una sola persona sarebbe inefficace.

Come ogni progetto appena nato, anche CoVE ha ampi margini di espansione. Queste sono alcune idee che ho avuto:

1. Fornire un sistema di lock a livello di sezione o capitolo.
2. Permettere ai partecipanti alla stesura di enunciare quale ruolo vogliano assumere.
3. Oltre alla chat in tempo reale, far si che questa possa essere anche vocale.

Una volta finita l'implementazione di CoVE, credo che questo aprirà le porte a editor simili che possano gestire tipologie differenti di documenti, come i disegni. Dico ciò perché, nel caso si intenda cambiare la tipologia dei documenti della quale bisogna occuparsi, si avrà anche da modificare l'editor. Ciò porterebbe a dover modificare anche il sistema OT sottostante e perciò le operazioni che lo caratterizzano, dovendo così trovare un nuovo modo per integrarlo con un VMS.

Bibliografia

[Cha96] Sudarshan S. Chawathe et al. «Change Detection in Hierarchically Structured Information». In: *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*. SIGMOD '96. Montreal, Quebec, Canada: ACM, 1996, pp. 493–504. ISBN: 0-89791-794-4. DOI: 10.1145/233269.233366. URL: http://doi.acm.org/10.1145/233269.233366.

[EG89] C. A. Ellis e S. J. Gibbs. «Concurrency Control in Groupware Systems». In: *Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data*. SIGMOD '89. Portland, Oregon, USA: ACM, 1989, pp. 399–407. ISBN: 0-89791-317-5. DOI: 10.1145/67544.66963. URL: http://doi.acm.org/10.1145/67544.66963.

[FMR14] Sayed MD Fahim Fahad, Md Abdur Rafi Ibne Mahmood e Mohammad Zahidur Rahman. «Reform Based Version Management System for XML Data». In: *International Journal of Computer and Information Technology* 3.6 (2014), pp. 1299–1304.

[Mun01] David Mundie. «Using XML for Software Process Documents». In: *Proc. of the Workshop on XML Technologies and Software Engineering (XSE2001)*. 2001.

[Roc75] Marc J Rochkind. «The source code control system». In: *IEEE Transactions on Software Engineering* 4 (dic. 1975), pp. 364–370. ISSN: 0098-5589. DOI: 10.1109/TSE.1975.6312866.

[SE98] Chengzheng Sun e Clarence Ellis. «Operational Transformation in Real-time Group Editors: Issues, Algorithms, and Achievements». In: *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*. CSCW '98. Seattle, Washington, USA: ACM, 1998, pp. 59–68. ISBN: 1-58113-009-0. DOI: 10.1145/289444.289469. URL: http://doi.acm.org/10.1145/289444.289469.

[SR15] Vijay R. Sonawane e D. R. Rao. «A Comparative Study: Change Detection and Querying Dynamic XML Documents». English. In: *International Journal of Electrical and Computer Engineering* 5.4 (ago. 2015). Copyright - Copyright IAES Institute of Advanced Engineering and Science Aug 2015; Ultimo aggiornamento - 2015-08-14, pp. 840–848. URL: http://search.proquest.com/docview/1695014710?accountid=9652.

[SS02] Haifeng Shen e Chengzheng Sun. «Flexible Merging for Asynchronous Collaborative Systems». In: *On the Move to Meaningful Internet Systems, CoopIS/4th DOA/ODBASE'02*. A cura di Robert Meersman e Zahir Tari. Vol. 2519. Lecture Notes in Computer Science (LNCS). Irvine, California, USA: Springer-Verlag (New York), ott. 2002, pp. 304–321.

[SS09] David Sun e Chengzheng Sun. «Context-based operational transformation in distributed collaborative editing systems». In: *IEEE Transactions on Parallel and Distributed Systems(TPDS)* 20.10 (ott. 2009), pp. 1454–1470. ISSN: 1045-9219. DOI: 10.1109/TPDS.2008.240.

[Sun02] Chengzheng Sun. «Undo As Concurrent Inverse in Group Editors». In: *ACM Trans. Comput.-Hum. Interact.* 9.4 (dic. 2002), pp. 309–361. ISSN: 1073-0516.

DOI: 10.1145/586081.586085. URL: http://doi.acm.org/10.1145/586081.586085.

[Sun04] David Sun et al. «Operational Transformation for Collaborative Word Processing». In: *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work*. CSCW '04. Chicago, Illinois, USA: ACM, 2004, pp. 437–446. ISBN: 1-58113-810-5. DOI: 10.1145/1031607.1031681. URL: http://doi.acm.org/10.1145/1031607.1031681.

[Sun98] Chengzheng Sun et al. «Achieving Convergence, Causality Preservation, and Intention Preservation in Real-time Cooperative Editing Systems». In: *ACM Trans. Comput.-Hum. Interact.* 5.1 (mar. 1998), pp. 63–108. ISSN: 1073-0516. DOI: 10.1145/274444.274447. URL: http://doi.acm.org/10.1145/274444.274447.

[Tic85] Walter F. Tichy. «Rcs — a system for version control». In: *Software: Practice and Experience* 15.7 (1985), pp. 637–654. ISSN: 1097-024X. DOI: 10.1002/spe.4380150703. URL: http://dx.doi.org/10.1002/spe.4380150703.

[XS16] Yi Xu e Chengzheng Sun. «Conditions and Patterns for Achieving Convergence in OT-based Co-editors». In: *IEEE Transactions on Parallel and Distributed Systems* 27.3 (mar. 2016), pp. 695–709. ISSN: 1045-9219. DOI: 10.1109/TPDS.2015.2412938.