RC24224 (C0703-012) March 23, 2007
Computer Science

IBM Research Report

Maintaining Consistency in a Web 2.0 Collaborative Editing System

Chang Yan Chi, Wen Peng Xiao
IBM Research Division
China Research Laboratory
Building 19, Zhouguancun Software Park
8 Dongbeiwang West Road, Haidian District
Beijing, P.R.C. 100094

Danny Yeh, Ravi Konuru
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Maintaining Consistency In A Web 2.0 Collaborative Editing System

Chang Yan Chi
IBM China Research Lab
chicy@cn.ibm.com

Wen Peng Xiao
IBM China Research Lab
xiaowp@cn.ibm.com

Danny Yeh
IBM T.J. Watson Research Center
dlyeh@us.ibm.com

Ravi Konuru
IBM T.J. Watson Research Center
rkonuru@us.ibm.com

Abstract. Web based collaborative editors are increasingly required within Web 2.0 era. Since this involves group editing, concurrency control and consistency of the updates needs to be addressed. In this paper we present an optimistically concurrent and serialization algorithm for Web based real-time group editing of XHTML documents. The algorithm uses standard W3C document object model (DOM) as its data representation model in the runtime environment. We implemented this algorithm in a Web 2.0 collaborative editor system, and at this time provide some preliminary observations on the working system.

Introduction

The World Wide Web (WWW) has been developed from Web 1.0 to Web 2.0 (Web 2.0) era, and the emerging technologies like AJAX, blog, and wiki let people rendezvous spontaneously to contribute their thoughts to the web world. As a result, the demands for online collaborative editing become stronger than before. For instance, Wikipedia, a popular wiki application system, has handy on-line editors like FCKEditor that help its users easily and quickly publish their rich format content. The most important characteristic of these on-line editors is their

use of W3C document model (DOM), which is different from the traditional rich text editors like Microsoft Word or OpenOffice Writer.

Within cyber communities, the needs for people to work on the same document synchronically over the Internet are increasing, especially when members of such virtual team across different geographies. The users require multiple peers to interact through a shared document with no additional constraints on their ability to do editing work on the XHTML page. There are many Web 2.0 web sites which providing collaborative document editing services for rich text or spreadsheet. One example is Google Write and its spreadsheet functions to provide on-line document system (Google docs).

The topic of enabling group collaboration in one single document editor has been extensively explored within the field of CSCW. There are many group-editing system in traditional groupware domain, Ellis (Ellis and Gibbs, 1989) gave the characteristics of real-time groupware systems, as well as presented one well-accepted algorithm to maintaining the data model consistency base on operation transformation, coined as OT. At that time, other existing groupware systems like GROVE (Sun, 1990, 1998) and Jupiter (David, 1985) also use plain text document data model to address the content consistence problem. It means the document model is plain text without rich styles, and can be located by linear address mechanism. It doesn't quite handle the vivid formatting of Web 2.0 application (Davis, 2002).

Recently research works try to use tree as the inner data model, for example, Davis (2002) uses general markup language as data model, and treeOPT (Ignat and Norri, 2002) uses document natural syntax structure as document tree representation which different with linear document representation. One limitation of such works is the method of operate the document model is private and hard to reused by other systems.

Operational transformation has been used as an appropriate approach for consistency maintenance in real-time collaborative editing system. Such existing transformation algorithms keep a history of already executed operations in order to compute the proper execution form of new operations. Unfortunately, when the participant sites have limited computation capability such as Web Browser, it's a heavy burden to do transformation within the browser

W3C DOM (Document Object Model 1998) is used as Web 2.0 client programming model. This means all document can be treated as DOM tree, not use internal document definition such as one document is consisting of characters, words, sentences, paragraph, section etc. There is some works to decompose W3C DOM based applications as Web services to be collaborative (Qiu, 2003).

This paper presents a specialized consistency maintenance algorithm for the web 2.0 environments that is based on DOM representation and defines the atomic operators in DOM-based share editing system to maintain document consistency in a single-server, multiple client configurations. To solve the

browser processing limitations, we adopt a server centric master-slave model that simplifies some of the issues in maintaining consistency. An addressing mechanism is defined to fine-grain synchronization of editing operations. Finally we present the implementation and observations of a group web-document editor. At this point, the results are preliminary and we plan to submit an updated set of observations if accepted.

The rest of the paper is structured as follows. In section 2, we introduce the document format that we focus on, and define a new addressing scheme, and the operators to manipulate a DOM tree. Typical usage and conflict scenarios are also discussed in this section. Section 3 presents our algorithm design. Section 4 briefly presents the implementation of web2.0 document editor and some preliminary results. Section 4 summarizes related works. Finally, we conclude in section 5.

Collaboration over XHTML Documents in Web Browser

To consider a web page rendered by modern web browsers like Internet Explore, Mozilla Firefox, Safari, and others, the content of the XHTML (XHTML1.0 specification) is not only plain text, but also rich styling information like font size, font color, and images. Web browsers act as a user-friendly viewer to render these contents.

In figure 1, we show a sample document view from user's perspective on the left side. This document fragment consists of a paragraph and a bullet list. The corresponding document model expressed in XHTML is illustrated on the right side.

The document's model is different with user's view as the style formatting such as font size, font color and metadata will be inserted as standard XHTML elements or attributes. It's different with Ignat (Ignat, 2006), which classified the nodes of the XML document into private types, not W3C DOM. In DOM specification, the attributes are distinguished from the children of the element.

Before presenting algorithm details, we briefly describe the document model in an on-line collaborative editing system. XHML is a tree-based document model that composed of nodes. A node (***N***) is a standard XHTML node:

- Element
  - Element node has zero or more than one children
- Attribute
  - Embedded as properties in the element node.
- Text
  - Text node is the leaf node

Every element is identified by its identifier but not position in the document structure. The unique identifier can be used to address every node in the XHTML tree model.

Figure 1 Standard XHTML Page and View

There are different synchronization granularity levels in the group editing systems, and some existing solutions use character based synchronization granularity. Since the document in our system is represented as nodes, the data synchronization also occurs at node level. For instance, if user has made some changes in his editor, e.g. insert characters or change the formatting styles, the nodes that are affected by the changes are sent in their entirety to the server for propagation to other clients. Depends on document structure, the node can be very large or very small.

Usage Scenarios

In a typical group editing session, multiple authors can interactively edit a shared document without additional constraints, so concurrent operations on the same nodes by different users are unavoidable. One typical case is the modify-delete scenario where one user deletes a node that is being modified by another user, and another case is where one or more users modifying the same node in the document. We consider these two scenarios below.

Imagine there are two users Tracy and Gary, and they are working on the same document through the Internet.

Situation 1: un-executable operations

Tracy inserts a new node when Gary deletes its parent node and nested children. As the result, if the deletion operation is executed firstly under server side, the sub sequential insertion operation will not be executed because it lost the reference nodes. Figure 2 illustrates this scenario in detail.

It's a conflict problem, not same with the intention violation (Sun, 1996). In one dimension editing system, they used linear address to locate the operations, so operations always can be executed.

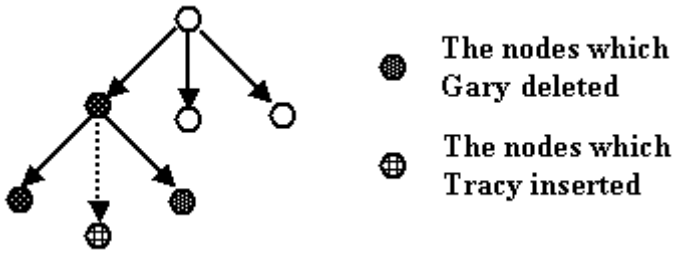

Figure 2 un-execute operations in situation 1

Situation 2: update same attribute

Tracy and Gary are changing the same attribute of the same node. It's also a very common scenario and similar with original OT scenario which has been addressed in a linear addressing editor.

In figure 3, Tracy and Gary's changed the font color to read and blue respectively, and then two synchronization messages are sent to server. Suppose server will processes these messages sequentially. When the Tracy's message arrived at server, the server model is changed to red, and then the change is broadcast to Gary's clients. According to the synchronization message, the document model is changed *red* from *blue* under Gary's client. The server continues to process the message sent by Gary, in this time, the server model is changed to blue. Following the same steps, the message is also broadcasted to Tracy's client, so the color would be changed from red to blue for Tracy. As a result, Gary's client will keep a consistency document model with server, but Tracy's client will have an in-consistency document model.

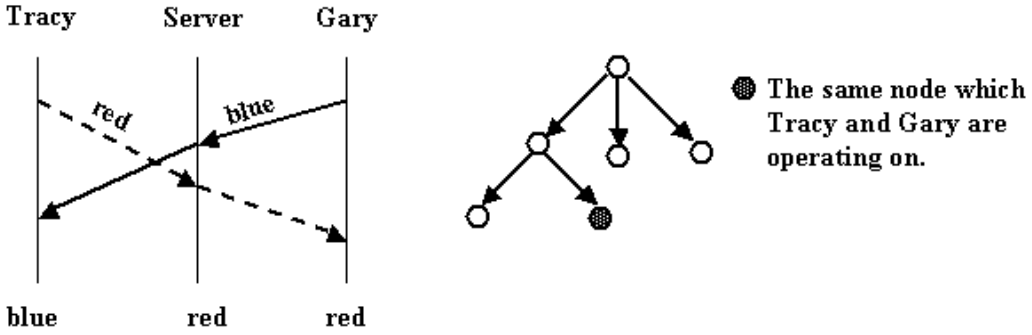

Figure 3: modify the same text node in situation 2

From the above scenarios, there are two cases of inconsistencies: the first one is the propagated operation cannot be executed at the server side, and the second one is the executed result is not the same at the client sites. In this paper, we present an optimistic serialization algorithm to solve these issues in the Web context. The detail is described in the latter section, but first we describe our address schema.

Operation Addressing Schema

In order to reproduce the same operations on the DOM at remote sites, an address scheme is needed to locate the correct node accurately. Fortunately, DOM defines a unique identity key for every node, that every node can host an attribute named ***id***, whose value is a unique string.

When user edits the document tree in one site, the actions are transformed to operate the DOM tree directly. The editor captures user's operations such as inserting a char, deleting a selection, or changing the style of an element. Editor should record the changes, for example, in figure 1, when user wants to insert a new bullet element "*Plan*" behind list "*Goal*", the inserted result should be:

<li id="xxxx"*><div id=*"yyy"*><span id=*"*zzzz*"*>PLAN *

On the other site, how will the above fragment be executed? It is very clear that the related location information also need be provided in the synchronization event. We propose a new addressing scheme for such tree model.

The operation address in our system is defined as a triple below:

Address {elementId, attrName, textIndex}

- elementId: if the node type is element, it will have a unified ID
- attrName: specifies the attribute name of one element node
- textIndex: specifies the index of text content if the node is text

For instance, in the figure 1, the address of element LI is {id, null, null}, its font style attribute address is {id, "font-weight", null}, the address of "*G*" of "*Goal*" is {id, null, 1}.

Because tree model is two-dimension structure, only one address is insufficient to locate the unify position in some cases. For example, assume an element ***A*** in one document tree, and one site appends a new element ***B*** as a child of element ***A***, to accurately propagate this operation to other sites that assures the consistency among all participants, we adopt two address parameters that defined above to describe one operation position. The first one is called as *target address*, and the second one is the *reference address*. Figure 4 illustrates how to insert one node in these two different situations above.

The dashed circle represents the node to be inserted. In (a), the node will be inserted between node 3 and node 4, so the *target* address is (2, null, null), and the *reference* address is (4, null, null). In (b), the node will be appended as the last

child of node 2, so the *target* address is (2, null, null), and the *reference* address is null.

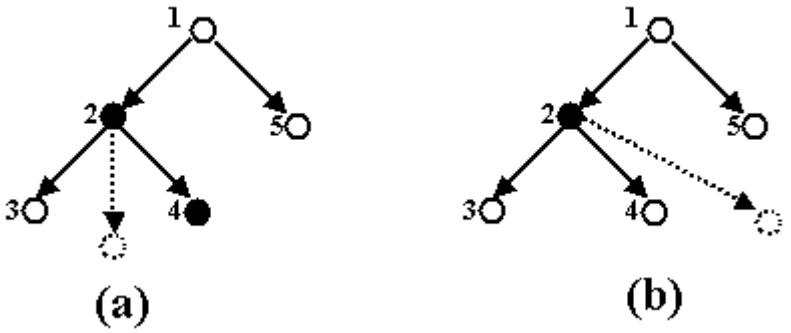

Figure 4 the tree graph of figure 1

Synchronized Operations

A set of operations contains basic *insert* and *delete* operations like existing group editing system (Ellis and Gibbs 1989) is also provided here; they have been proved suitable for systems that use linear address space to execute the operations. According to the inherent properties of tree based data module, we also add a new operation: *replace*.

Insert: add a new node into the document tree. This operation will be ignored if the node has the same address already existed in document tree.

Insert (new node, target address, reference address)

The first parameter is the new node. It's a XHTML fragment that includes node id, attributes and children node if necessary. The rest of parameters locate the position where the node will be added.

Delete: remove one existing node from the document tree. The operation will be ignored if the node specified by given address does not exist in this document.

Delete (target address)

***Replace*:** change the content of one text node, or change the attributes of one node

Replace (target address, reference address, raw data)

The third parameter "*raw data*" will replace original data located by the target address and reference address. If reference address is empty, it means updating attribute.

Optimistic Serialization Algorithm for DOM

In this section, we will present the optimistic serialization algorithm that we defined for maintaining the consistency of DOM based groupware applications.

We follow the same consistency mode that claimed by Ellis and Sun (1985). The most important three properties are described as following:

- *Convergence* property means each participant has the same structure after all sites have executed the same set of operations
- *Causality* preservation property means operations' execution order is same on each site
- *Intention preservation* property means if a given operation ***O*** is executed on a site, the execution effect is exactly same as the original intention of ***O*** at the site where it was generated.

According to our server centric synchronization model, all clients' operations are executed in server site sequentially, which achieve the second causality preservation property at the server side and we use the master-slave relationship to simplify the decision of the ordering of operations for the clients.

Design Disciplines

The system architecture of our collaborative editing system is client/server where multiple clients (***C***) are allowed to take part in the same editing session, and a central server (***S***) is used to coordinate the changes and to keep each client synchronized. A copy of document object model (***DOM)*** is maintained at each site (both the clients and the server) to represent current state of the document that is being edited, and each user's editing operations is modeled as operation (***O***), which changes the state of the ***DOM***.

Rule 1: The operation which applied to server document model DOM(S) is irrepealably.

Policy and algorithm should be designed to guarantee that the **DOM(S)** always in the right and properly situations at any time in order we can use it as the reference base in this system at any time.

Rule 2: Operations on DOM(S) should also be applied to client document model DOM(C).

Supposes the sign "==" denotes that two ***DOM*** are equal, and the pre-condition is **DOM(C1) == DOM(S1)**. We also suppose an operation **O(1)** is executed at server side which changes **DOM(S1)** to **DOM(S2)**. At client side, a corresponding operation should also be applied to make **DOM(C1)** change to **DOM(C2)**, and the post-condition of this operation must satisfy **DOM(C2) == DOM(S2)**.

Rule 3: The final sequence of operations applied in DOM(S) and DOM(C) should be same.

In order to keep the document content consistently in different sites, a basic disciplines of this system is keeping the final sequence of operations exactly same at each site.

Basic System Modeling

Let's start from the simplest collaborative editing system which embodies a server (***S***) and two clients (***C1*** and ***C2***). At the startup time (***t0***), the content of document in each site is same:

$$DOM(S, t0) == DOM(C1, t0) == DOM(C2, t0)$$

Consequentially, some modification happens in ***C1*** at the time of ***t1***, which makes the document model changed to ***DOM(C1, t1)*** at this client site. This change can be modeled as an operation ***O(C1, seq1)*** on ***DOM(C1, t1)***:

$$DOM(C1, t1) = DOM(C1, t0) * O(C1, seq1)$$

Each client site maintains a counter to generate a sequential number for identifying operation. The sequential number of different site maybe same, but the pair of ***(client, sequence)*** should be unique, that is the reason why we use ***O(C1, seq1)*** to identify the operation in above case. In the following sections, the model of operation (***O***) will be extended to accommodate more complex situations, and we just keep it as simple as possible here to start from.

According to *rule 1*, an operation executed in server side is validated and irrepealably, so ***O(C1, seq1)*** is vulnerable before it is applied to server side. To fulfill this purpose, client ***C1*** will request server to do the same change on ***DOM(S)*** by sending the operation ***O(C1, seq1)*** via network. This operation arrives in server side at time ***t2***, and will be executed by server on ***DOM(S, t0)***:

$$DOM(S, t2) = DOM(S, t0) * O(C1, seq1)$$

In the above equations, we use asterisk (*) to strand for an operation is executed, or applied, on a specified ***DOM*** state. Now, ***DOM(S, t2)*** is equal with ***DOM(C1, t1),*** because the initial status and applied operation are same. In other words, client ***C1*** is consistent with ***S*** now. In order to keep others clients synchronized, an executed operation ***EO(C1, seq1, exeseq1)*** will be sent to each client. Server also maintains a counter to generate a sequential number (***exeseq1***) for identifying each executed operation. In our case, when this executed operation is received by the operation original client (e.g. ***C1***), no other actions need to be performed at this site. On another hand, when this executed operation is received by other client sites (e.g. ***C2*** at time ***t4***), the corresponding operation must be applied on current document model. We can describe it as following equation:

$$DOM(C2, t4) = DOM(C2, t0) * EO(C1, seq1, exeseq1)$$

The difference between ***O(C1, seq1)*** and ***EO(C1, seq1, exeseq1)*** is the additional attribute which denote the execution sequence in server side. When it is applied in client ***C2*** at time ***t4***, we can infer that each sites keeps a consistent document model:

$$DOM(S, t4) == DOM(C1, t4) == DOM(C2, t4)$$

Figure 5 illustrates the basic system model of this sequence.

What's more, we can infer that the document model at server site from ***t2***, is equal with the document model at client sites from ***t4***, so we can use

DOM(EO(C1, seq1, exeseq1)) to represent such unified document status in our deeper discussion.

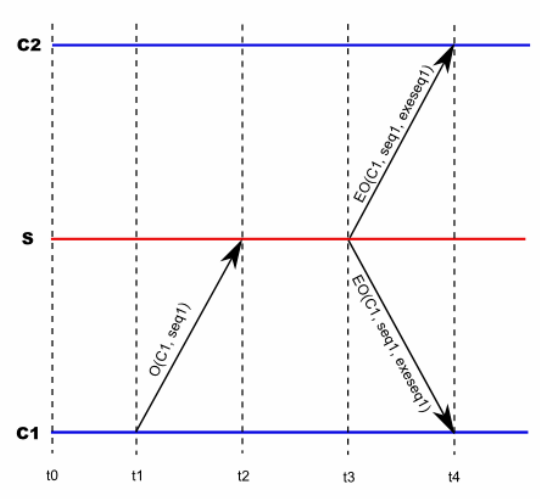

Figure 5 basic system modeling for operation executing

Algorithm on Client Site

Consider a more complex scenario in a real collaborative editing system which is illustrated in figure 6. Both client and server start from the consistent status ***DOM(EO0)***, and the user in client ***C1*** and ***C2*** perform some operations sequentially. In order to avoid jumping into too complex discussion directly, let's start from the case in which all of these operations can be executed at server side successfully.

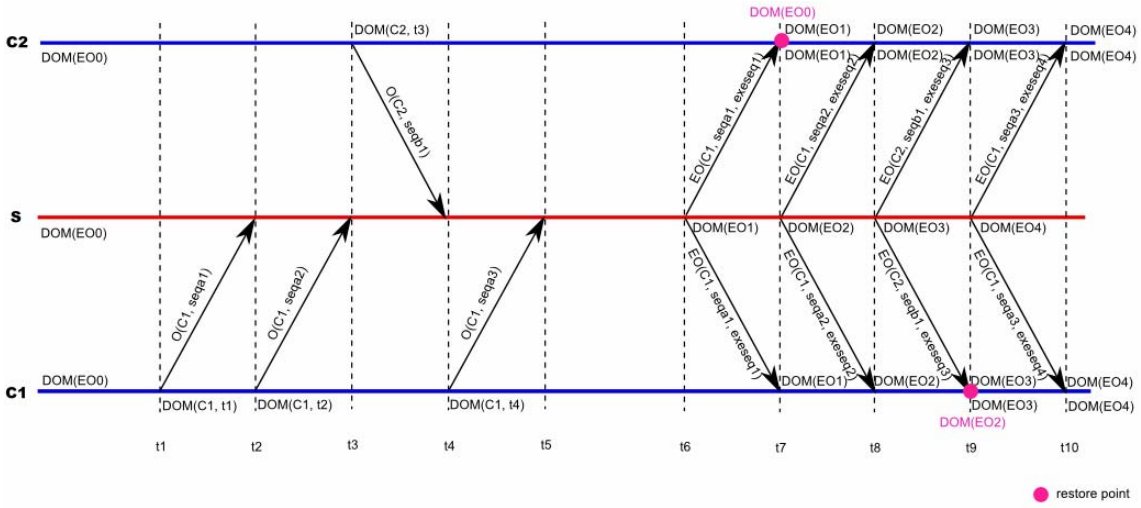

Figure 6 collaborative editing session between two clients

In our case, after the operation ***O(C1, seqa1)*** is received by server at time ***t2***, a piece of processing time is needed before this operations is really executed by

server at time ***t5***. After that, the document model at server side will change to ***DOM(EO1)***, ***EO1*** is the abbreviation for ***EO(C1, seqa1, exeseq1)***. Subsequently, the executed operation ***EO(C1, seqa1, exeseq1)*** will be sent to ***C1*** and ***C2*** at time ***t7***. At client site ***C2***, there is an ***unexecuted operation***, ***O(C2, seqb1)***, from its perspective because the ***EO(C2, seqb1, exeseq3)*** is not arrived yet at time ***t7***. For this reason, ***C2*** must apply the operation ***EO(C1, seqa1, exeseq1)*** on ***DOM(EO0)*** but ***DOM(C2, t3)*** because the intermediate document model ***DOM(C2, t3)*** is not authoritatively. In other words, the ***DOM(C2, t3)*** will be discarded, and ***C2*** will restore its document model to ***DOM(EO0)*** before applying ***EO(C1, seqa1, exeseq1)***. At client side ***C1***, there is a list of unexecuted operations including ***O(C1, seqa1)***, ***O(C1, seqa2)*** and ***O(C1, seqa3)*** at time ***t7*** because none of their corresponding executed operations has been handled by ***C1***. By comparing the first item of this list (***O(C1, seqa1)***), and the arrived ***EO(C1, seqa1, exeseq1)***, ***C2*** can tell that the ***O(C1, seqa1)*** has been executed successfully by server, so the only thing that need to do is removing ***O(C1, seqa1)*** from the unexpected operations list. We can use bellowing pseudo-code to describe such process:

```
dom := document model at client site
domeo := last consistent document model at client site
uo :=   unexecuted operations list at client site
eo :=   executed operation from server

let o = uo.firstItem()
if o != eo
then
   dom.restoreTo(domeo)
   dom.applyOperation(eo)
end
uo.removeFirstItem()
domeo = domeo * eo   // update last consistent document model
```

For more details, the definitions of an operation is equal (denoted as "==") or not equal (denoted as "!=") with an executed operation are:

Definition1: ***i) O == EO iff***
O.client == EO.client and O.seqence == EO.sequence
ii) O != EO iff
O.client != EO.client or O.seqence != EO.sequence

According to above algorithm, when the executed operation ***EO(C2, seqb1, exeseq3)*** arrives in client ***C1*** at ***t9***, the last consistent document model in this site is ***DOM(EO2)***, the local document model is ***DOM(C1, t4)*** as it didn't do any restore operation when handling previous two executed operations, and the unexecuted operations list contains one item, ***O(C1, seqa3)***. By comparing ***EO(C2, seqb1, exeseq3)*** and ***O(C1, seqa3)***, it is obvious that ***C1*** should restore local document model to ***DOM(E02)***, apply action ***EO(C2, seqb1, exeseq3)*** on local document model, remove ***O(C1, seqa3)*** from unexecuted operations list, and

update the last consistent document model to ***DOM(EO3)*** by applying ***EO(C2, seqb1, exeseq3)*** on ***DOM(EO2)*** respectively.

Algorithm on Server Site

Now we will move to the design of server side. In order to conform rule 1, server can be simply implemented to sequentially process the operations coming from the client sites to avoid the concurrency problems. There is no limitation for client to decide when to send operation to server, so we can use a pending ***operation queue*** at server site to queue all operations need to be handled. Suppose each operation handled by server need the same time ***T***, just as we illustrated in figure 7.

We need some more definition before go on with more discussion. One of them is whether an operation (***O***) is compatible with (denoted as "$\in$") a specified document object model (***DOM(S)***):

Definition 2: ***O □ DOM(S) iff***
i) O can be applied to DOM(S)
ii) DOM(S) * O still in properly status

Another definition is whether two operations are compatible (denoted as "$\square_{DOM}$") on a specified document object model (***DOM(S)***):

Definition 3: ***O1 $\square_{DOM(S)}$ O2 iff***
DOM(S) * O1 * O2== DOM(S) * O2 * O1

We also give the definition of change set (***CS***). If a sequence of operations (***O1, O2, … On***) can be applied on an source document model (***DOM(S)***), and get the target document model (***DOM(T)***), then the change set is represented as the difference between these two models (denoted as "-"):

Definition 4: ***CS(DOM(T), DOM(S))***
= DOM(T) – DOM(S)
= O1 * O2 * … * On

An operation executed by server site may compatible with (also denoted as "$\in$") a change set or not, according to the following definition:

Definition 5: ***Om $\in$ CS(DOM(T) - DOM(S)) iff***
i) Om $\in$ DOM(S)
ii) CS(DOM(T), DOM(S)) = O1 * O2 .. * On
ii) Π (i=1, n) Om $\in_{DOM(S)* O1 * ..Oi-1}$ Oi

A deduction can be deduced using the mathematical induction:

Deduction: ***if O $\in$ DOM(S)***
and
O $\in$ CS(DOM(T) - DOM(S))
then
O $\in$ DOM(T)

It is the time to extended our operation definition to ***(client, sequence, domeo)*** in which the ***domeo*** can identity the last consistent document model when this

operation is sent by client site. In figure 7, when the operation ***O(C2, seqb1, EO0)*** is processed by ***P4*** at server site, current consistent document model in server is ***DOM(EO3)***, but this operation happens on previous consistent document model ***DOM(EO0)*** in client side at ***t4***., For this reason, server should check the compatibility of operation ***O(C2, seqb1, EO0)*** on ***DOM(EO3)*** to decide whether this operation can be executed or not. We can use bellowing pseudo-code to describe such process:

```
oue := operation under execution at server site
domeo := last consistent document model at server site

// check compatibility
if oue.domeo != domeo and not oue ∈ CS(domeo – oue.domeo)
then
    reject = new RejectOperation(oue)
    sendRejectedOperation(oue.client)
    return
end

// propagate messages
domeo = domeo * oue
exeseq = exeseq + 1
eo = new ExecutedOperation(oue, exeseq)
sendExecutedOperation(ALL_CLIENTS)
```

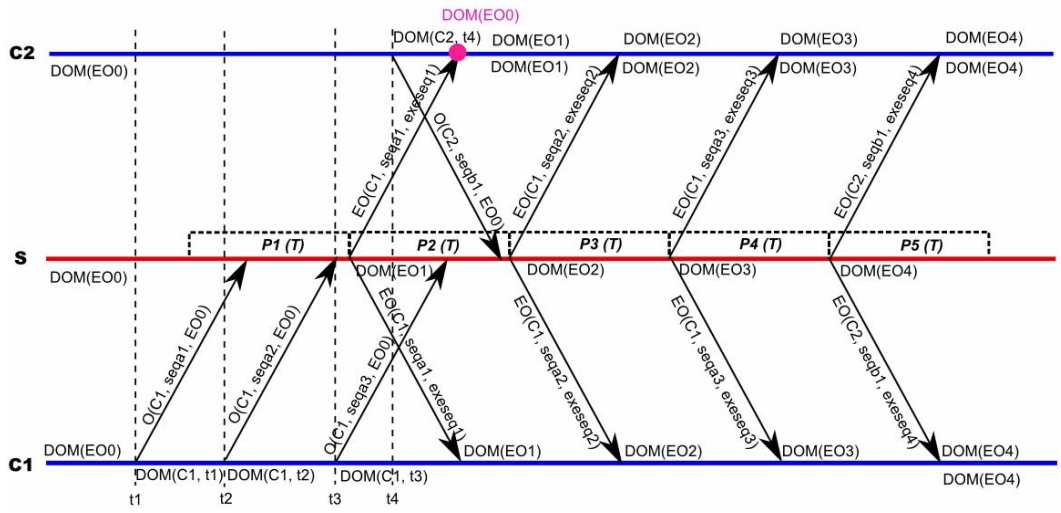

Figure 7 operations compatibility checking

Web 2.0 Group Document Editor Implementation

We implemented a web based group editor that uses the algorithm described above whose screenshot is presented in figure 8. The left side bar in this

screenshot illustrates there are two people working on the same document, who have different color identities. The color in the content area means different ownership, and we found it to be clear and helpful to know who contributed to which changes.

In details, the on-line editor hosted in browser composed by a) editing component handles user's editing operations, and b) synchronizing component synchronize, execute and propagate operations on shared documents. Editor has the capability to capture user's operations, and then transform it to synchronization messages that will to be executed in other peer sites. There are two kinds of messages to be executed; one comes from local user, and one from other sites.

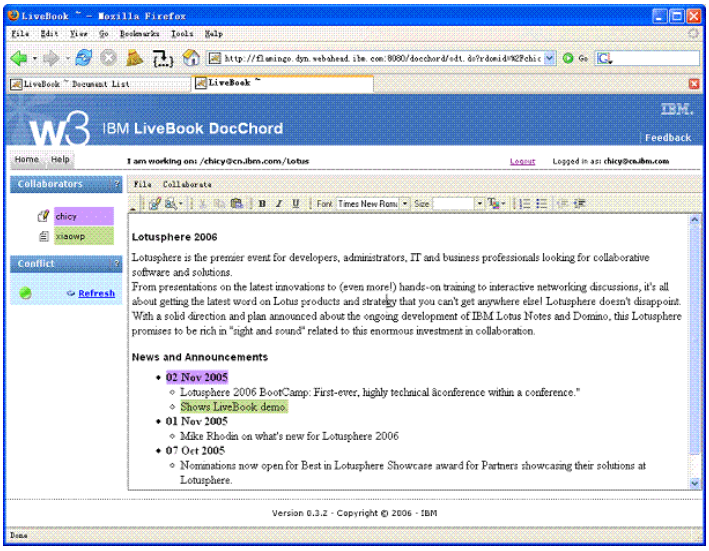

Figure 8 on-line collaborative Editor UI

Messaging is used in this editing system that can be executed in each participant's document model. According to the specification, standard DOM implementation will fire different events when data model is changed. Consequently, if we want to support web based collaborative applications, the collaborative event object should be well defined and contains sufficient information for collaboration.

We define a XML based collaborative event protocol and convert DOM event to our semantic synchronization event at client site when user's action is detected and send to server. A similar message will be received by each other client site after the operations has be processed at server site. The receiver will convert the collaboration event to DOM actions and perform corresponding operations in local document model. One message can contain several operations that reflect user's atomic intention.

At client side, the user can freely edit the document without any limitations. Upon receiving a remote operation, the receiving site will verify it first for causal readiness. If the composite operation is not causally ready, the operations in the

history stack will rollback to original state, and then execute the remote operations, it's very similar as a UNDO operations.

We have done some preliminary performance testing for the whole system. We simulated clients to send synchronization events at same time, when the number is less than 20, the response time is less than 5 seconds. The system is currently running as a pilot system that supports more than 100 active users.

Related Work

The important characters of real-time groupware systems are "highly interactive" and "real-time" (Ellis and Gibbs 1989). Locking manipulated objected isn't a good user experience. Operation serialization and operation-based approach are different approach to solve concurrency control problem in real-time groupware system within lock-free property.

The Operational transformation algorithm dOPT (Ellis and Gibbs, 1989), and sub-sequential extend algorithms, like David (1989), Sun (1998) have been used in many group editors, which focus on plain text document model. It means the document model is plain text without rich styles, and can be located by linear address mechanism. Basic OT algorithm also is extended to handle different document model. treeOPT (Ignat and Norri, 2002, 2003) models the text document using a hierarchical structure as tree representation of the document, which has several levels of granularity: document, paragraph, sentence, word and character corresponding to the common syntactic elements used in natural language. Because the document is a generalization of the linear representation, so the treeOPT algorithm can extend the existing OT algorithms.

CoWord (Xia, 2004) proposed a transparent adaptation approach to adapt the single-user application-programming interface to the data and operational models of the underlying collaboration supporting technique. It access all data objects of a Word document by their positional references in a linear address space from an adapted Word API, which meets the data modeling requirement of OT. The assumption is the shared application's API should be adaptable. CoWord is one example that converts Word to a real-time collaborative word processor.

To consider XML markup language as document model, Davis (2002) provided an extended OT algorithm to handle general XML, "The major operational transformation editors use an abstract data model of a single linear sequence of content data. Such a flat sequence can model flat text". Ignat presented an operation-based approach to merger XML document for an asynchronous editor (Ignat and Norri, 2006a).

Greenberg S. and Marwood (1994) examine locking, serialization, and the degree of optimism etc. concurrence control methods, which have quite different impacts on the interface and how transitions are shown to, and perceived by group members. There is a general concurrency control method to fulfill different

application domains. GROUPKIT gives groupware developers the power to choose a concurrency scheme that fits the nuances of their application.

Qiu X. (2003) investigate a general approach to restructure W3C DOM based applications as web service, and then apply a general approach to making web services collaborative. Java SVG application is used to demonstrate the idea. .

Conclusions and future work

Within Web 2.0 era, people are eager for rich editing experiences in on-line collaborative environments. On the other hand, how to keep document model consistency is another major issue of Web based group editor systems while keeping the processing load on the browsers small and maintaining a good interactive response. This paper presents a consistency maintenance algorithm that is tuned for the online environments where browsers are used by each user to collaborate on the same document and a central server to keep the document's state synchronized and consistency. In this algorithm, W3C standard document object model (DOM) is extended for distributed environments in order to identify the right address when synchronous operation is executed on remote sites.

As the computation capability is a major limitation in our target environments, such algorithm is designed to simplifying some of the issues in pure peer-to-peer systems and reducing processing requirements on the browsers. The principle of server controlling execution sequences, and all participants executing with the same order differs from previous work in terms of the rules used preserving consistency properties.

Our preliminary work uses this algorithm to develop an on-line text editor. Because the basic Web programming model is based on the DOM to develop kinds of applications, our future work is extending our work to other on-line co-authoring system which base on XML or XHTML, for example SVG drawing. As a next step, we also want to do more user studies to improve the system usability.

Acknowledgments

We thank our colleagues Robert Flavin for insightful introduction to this area, Apratim Purakayastha, Hui Su, and Frank Fu for support during this period of design and development. We thank all the anonymous referees for their time in reading this paper.

References

Davis, A.H., Sun, C., J. (2002). 'Generalizing operational transformation to the standard general markup language'. *Proceedings of CSCW 2002*, New Orleans, Louisiana, USA, pp.56-87

Ellis, C.A, and Gibbs, S.J. (1989). 'Concurrency control in groupware systems', *Proceedings of the ACM SIGMOD Conference on Management of Data*, pp.399-407

David A. Nichols, Pavel Curtis, Michael Dixon, and John Lamping (1985). 'High-latency, low-bandwidth windowing in the Jupiter collaboration system', *Proceedings of the ACM 1995 Symposium on User Interface Software and Technologies*, ACM, November 1995, pp.111–120.

Sun, C., Yang, Y., Zhang, Y., and CHEN, D. (Sun 1996). 'A consistency model and supporting schemes for real-time cooperative editing systems', *Proceedings of the 19th Australian Computer Science Conference*, Melbourne, pp.582-591

Sun, C., Jia, X., Zhang, Y., Yang, Y., Chen, D. (1998). 'Achieving Convergence, Causality-preservation, and Intention-preservation in Real-time Cooperative Editing Systems'. *In ACM. Trans. on Computer-Human Interaction 5, 1(March) 1998.* pp.63-108.

Sun, C., Ellis C.A. (1998). 'Operational Transformation in Real-Time Group Editors: Issues, Algorithms, and Achievements', *in proceeding of CSCW'98 (Nov. 1998), ACM Press*, pp.59-68

Greenberg S. and Marwood D. (1994). 'Real time groupware as a distributed system: concurrency control and its effect on the interface'*, in proceeding of ACM conference on Computer Supported Cooperative Work*, pp.207-217

Xia, S., Sun, D., Sun, C., Chen. D., Shen, H. (2004). 'Leveraging Single-user Applications for Multi-user Collaboration: the CoWord Approach'*, in proceedings of the CSCW'04*, Chicago, Illinois, USA

Sun D., Xia S., Sun C., Chen D. (2004). 'Operational Transformation for Collaborative Word Processing*'*, *in proceedings of the CSCW'04*, Chicago, Illinois, USA

Qiu X. (2003). 'Internet Collaboration using the W3c document Object Model', *proceedings of International Conference on Internet Computing*, Las Vegas

Ignat, C.L. and Norrie, M.C. (2002). 'Tree-based model algorithm for Maintaining Consistency in Real-Time Collaborative Editing Systems', *The Fourth International Workshop on Collaborative Editing Systems, CSCW 2002*

Ignat, C.L. and Norrie, M.C. (2003). 'Customizable Collaborative Editor Relying on treeOPT Algorithm', *in proceedings of the 8th ECSCW'03*, Helsinki, Finland, pp.315-334

Ignat, C.L. and Norrie, M.C. (2004). 'Grouping in Collaborative Graphical Editors', *in proceedings of the CSCW'04*, Chicago, Illinois, USA, pp.447-456

Ignat, C.L. and Norrie, M.C. (2006a). 'Flexible Collaboration over XML Documents', *in proceedings of the 3rd International Conference on Cooperative Design, Visualization and Engineering, CDEV' 06*, Mallorca, Spain.

Ignat, C.L. and Norrie, M.C. (2006b). 'Draw-Together: Graphical Editor for Collaborative Drawing', *in proceedings of the CSCW'06*, Banff, Alberta, Canada, November, pp447-456

Document Object Model Level 1 Specification. W3C Recommendation, http://www.w3c.org/ TR/REC-DOM-Level-1/1998

XHTML 1.0 specification, W3C Recommendation, http://www.w3c.org/TR/2002/REC-xhtml1-20020801/

AJAX technology: http://en.wikipedia.org/wiki/Ajax_%28programming%29

FCKEditor, http://www.fckeditor.net

Google Docs, http://docs.google.com

Web 2.0, http://oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html