International Journal of Multimedia and Ubiquitous Engineering
Vol. 10, No. 3 (2015), pp. 45-58
http://dx.doi.org/10.14257/ijmue.2015.10.3.05

Research in Distributed Evolution based on Real-time Collaboration Technology

Shanshan Wang, Liping Gao, Sizhen Zhu and Chunxue Wu

University of Shanghai for Science and Technology
Shanghai Chine
wangshan14@126.com; lipinggao@fudan.edu.cn; zhusz@usst.edu.cn
andtyfond@126.com

Abstract

Evolution design takes the advantage of the evolution technology into the art design, and designers can achieve satisfactory products by performing evolutionary operations. It's a powerful method in the procedure of the art design. However, the traditional evolution running on single site leads to the low efficiency and the less diversity. In addition, the products just represent the aesthetics of the single site. Based on these problems, this paper aims at using distributed evolution technology to complete the collaborative evolution, broadcasting evolutionary operations among distributed sites and executing evolutionary operations on each site. Then, the acquired products could represent the aesthetics of most sites due to its rich diversity. The paper mainly solves the problems of the operation conflicts which come from different sites, builds a new hybrid document model designs several algorithms based on Operation Transformation (OT) to solve the conflicts, and describes the procedure of the consistency maintenance. The proposed solution has been theoretically verified for its correctness in hybrid model.

Keywords: *Collaborative design, Evolutionary Design, Consistency maintenance, Hybrid document model*

1. Introduction

Evolutionary design [1-4] is a new design process, which aims at meeting the demand of new products and preserving the excellent characteristics of the original products. It stimulates the steps of biological evolution process [5], and new products are obtained from the original products by evolution operations [6] (including Crossover, Selection, Mutation *etc*.,). Evolutionary design is based on the products, and new products can also act as the original products for the next evolutionary operation, so the process of evolution is a cyclical process. This process can promote the reuse of the productions and can improve the efficiency of the design to a large extent. However, in the traditional evolutionary design [1-4], the design of a production is finished by one or more users in dispersed sites in which every user designs his product desperately, and then the excellent products can be selected according to the centralized comparison and evaluations. Because each of the designers differs on aesthetics standards and design conceptions, the productions are always quite different under this detached pattern. Therefore, it takes a lot of human and material resources to complete the final design of excellent products.

Real-time collaboration technology [8-10] provides us with a new method, by which the designers distributed all over the world can cooperate and communicate conveniently through the Internet. If different users in different places are allowed to execute the Crossover, Mutation and Selection operations at the same time in the process of the evolutionary design, the speed and quality of the evolutionary design can be improved significantly by ensuring the consistency of the document that located at distributed sites. However, because of the simultaneous releases of the different operations from different

ISSN: 1975-0080 IJMUE

sites, conflicts will occur frequently and if no strategy is adopted to guarantee the consistency of the document, the consistency of the distributed sites cannot be achieved. Another problem is the network, if every site transports the data from local site to the central server, the data may be delay as the network limit. Therefore, in order to reduce strain on networks, central server will not be adopted and a replicated model has been used in our research, every site just need to transport the operation order to remote sites and the remote sites execute operation on local site.

The rest of the paper is organized as follows. In Section 2, we introduce the related works of real-time collaboration technology and evolution technology. In Section 3, we define a new document model of the evolution design environment and a new set of operations (including Select, Crossover, Mutation and Score) for the evolutionary design. In Section 4, we analyze the conflict problems among the operations. And in Section 5, we design a consistency maintenance algorithm based on Operation Transformation (OT) algorithm [11-13] to resolve the conflicts. The efficiency analysis of the algorithm and the example are given in Section 6.

2. Tree-based Coding Method

The object of the evolution environment is represented as an expression of a mathematical function string, with the shape of the function representing the outline of the arts. In order to support the evolution design, the function strings are coded as binary trees and genetic operators [5, 6] (including Crossover, Mutation, Selection *etc*.,) are operated on the binary tree. The binary tree [1] is a finite set that is composed of the mathematical operation and operators. For example, if we have an expression:

$$y = \log\left(x^2 + \sin(x)\right) + x^{\cos\frac{x}{2}}$$

we can obtain the corresponding binary tree shown in Figure 1.

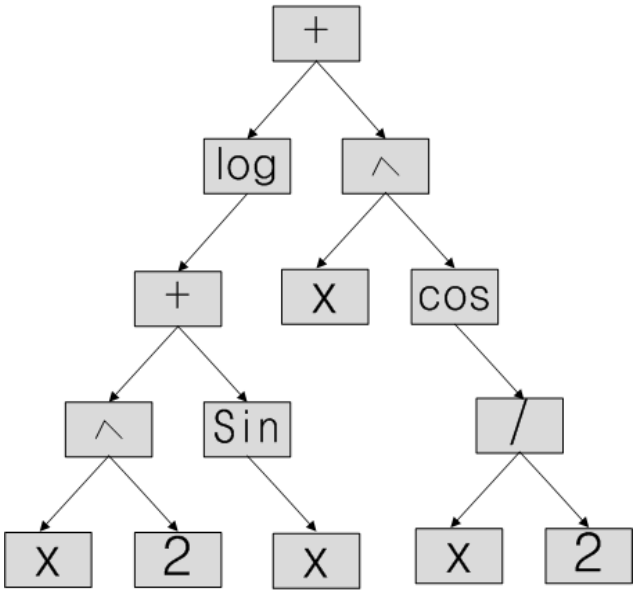


Figure 1. The Binary Tree of the Expression

After traversing the binary tree of the Figure 1, we can obtain the original expression. All of our operations are based on the binary tree, and several operations (including the crossover, mutation, selection) which are defined on the binary tree.

The introduction of crossover, mutation and select we can reference [7].

3. Hybrid Document Model

3.1. The Hybrid Document Model

We can reference method [14] as our document model, suppose that we have an expressions as below:

$$y_1 = \log(x_1^2 + \sin(x_1)) + x_1^{\cos\left(\frac{x_1}{3}\right)}, y_2 = x_2^2 * (x_2 + \cos(x_2)), y_3 = \frac{1+x_3}{x_3}, \ldots, y_n$$

We can define a model as Figure 5:

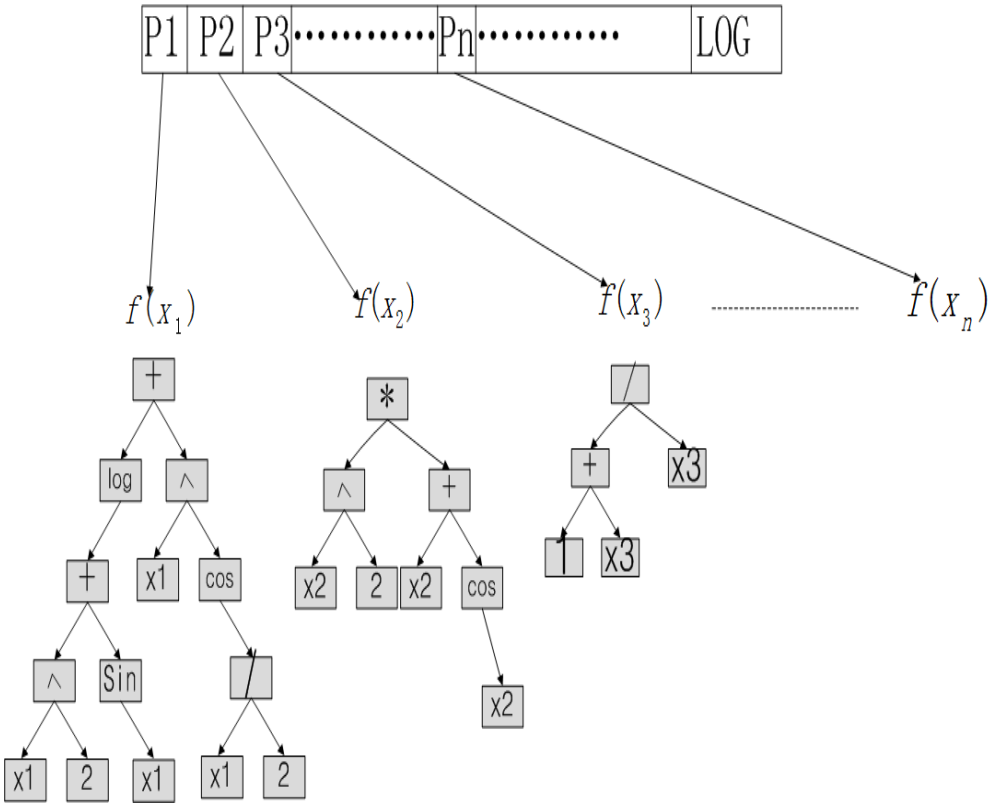


Figure 5. The Linear between Different Binary Trees

P1, P2..... Pn form a linear structure, and Pi represents the unique identifier of every binary tree. Every node of the linear structure Pi is composed of two parts P_Tree and Score, denotes as Pn = <P_Tree, Score>.

3.2. Data Operation

Suppose that all the operations are legal. There are two types of operation in our model: the Tree Level Operation (TLO) and Unit Level Operation (ULO). TLO are the operations for the whole binary tree (linear document operation). ULO are the operations for the part of the binary trees (binary tree operation). TLO consists of Insert, Delete, Set-score, Update (Update operation of TLO is defined as a ULO operation). There are many researches about the consistency maintenance of the Insert and Delete operations of the linear document (*e.g.*, adOPTed [10], GOTO [8], COT [13] *etc.*), so the consistency maintenance of TLO can be tackled successfully with these strategies. As for the Update operation, it can be divided into several operations like Selection, Crossover, Mutation, Delete *etc*. In this paper we focus on solving the problem of consistency maintenance of these operations.

According to the document model that we have described in Section 3, the definition of Insert, Delete, Set-score, Select, Crossover and Mutation operations are described as follows.

Insert: Insert a tree expression f(x) to the nth position of the array. E.g., Insert $(n, f(x))$ $(0 \leq n \leq N,$ N is the total number of binary tree).

Delete: Delete a binary tree f(x) from kth position of the array, *e. g.*, Delete(Pk,f(x)) $(0 \leq k \leq N)$. In order to support Undo operation [16, 17], the information of the to-be-deleted tree is also stored in the Delete operation. Pk is a pointer which points to a binary tree, f(x) is the binary tree that is to be deleted. We also define Undo and Redo operations for this operation. The Pre-condition of delete is that Pk must exist.

Set-score: Set-score operation is used to set the score of each binary tree by every site. The ID of the site and the score will be saved. E.g. Set-score (Pk,Sn,Numx), with 'Pk' indicating the ID of the binary tree ,'Sn' indicating the ID of the site, and 'Numx' indicating the score. The Pre-condition is that Pk must exist already.

Select: Select (param): with param indicating the proportion of the selection. For example, 0.8 means that the 20% of individuals will be eliminated according to the score. The rule of calculating the score of an individual is like this: if the scores number is larger than three, remove the maximum score and the minimum score, and then average the rest of the scores as the final score, otherwise average scores directly. Heapsort will be used to eliminate the products with the low scores. After selection, we should clear the Log, Queue and Status on all sites. Select operation aims to select excellent individuals from the group. Before executing the Select operation, we must confirm that all sites have been paused (Pause operation should be referenced to the definition 3.

The process of Select is as below, and its pre-condition is that all sites must be paused.

Crossover: Crossover(N,Pi,Ng,Pj,Nk)($i \in N$, $j \in N$, $i \neq j$, N is the total number of the binary trees) means that we select nodes of Ni and Nj from Pi and Pj respectively, and swap two sub-trees between Ni and Nj. Because the objects of Crossover operations cannot be the same one, a rule is made that $i \neq j$. During the execution process of the operation, two target trees are firstly copied and inserted to the last of the array, then the two sub-operations are swapped on the new trees. The strategy of operate on the new trees can avoid the conflict between Cross operation and other Insert, Delete operations. The Pre-condition of Crossover is that Pi and Pj must exist.

The execution process of Crossover is described as bellows, we could call Crossover(N,Pi,Ng,Pj,Nk)

Input:

N: the total number of the binary trees

Pi: the pointer of Pi

Ng: the node of Pi tree

Pj: the pointer of Pj

Nk::the node of Pj tree

Output: A new binary tree array

Method:

```
if i=j then return;
else
        Flag = 0;
        copy Pi and insert to the last of the tree array
        copy Pj and insert to the last of the tree array
```


if Ng.ID=1 then {Pi.P_Tree=Nk;Nk.PointFront = null;Nk.ID=1;}

else

if Ng.ID& 1=0 then {Ng.PointFront.PointLeft = Nk;}

else{Ng.PointFront.PointRight = Nk;}

if Nk.ID=1 then {Pj.P_Tree=Ng;Ng.PointFront = null;Ng.ID =1;}

else

if Nk.ID& 1=0 then {Nk.PointFront.PointLeft=Ng;}

else{Nk.PointFront.PointRight =Ng;}

Flag = 1;

Mutation: The execution of Mutation is somewhat similar with that of Crossover. Before executing the replacement of the sub-tree, a new tree is also copied to the last of the array, and then mutation is executed on this new tree. The Pre-condition of Mutation is that Pi must already exist. There are several types of mutation, and in this paper, we only discuss two types.

Replace the STR of a node with a new STR.

Replace the sub-tree with a new tree.

For the former case, the Mutation operation is defined as: Mutation(N,Pi,Nk,Str), with N indicating the total number of the binary trees, 'Pi' representing the pointer of the to-be-operated binary tree, 'Nk' representing the node of the tree, and 'Str' indicating the new string for Nk.

Mutation(N, Pi, Nk, Str):

Input:

N: the total number of binary trees

Pi: the pointer of Pi

Nk:: the node of Pj tree

Str: the content for mutation

Output: A new binary tree

Method:

Flag is execute mutation operation

Copy Pi and insert to the last

Replace Pi.Nk.STR by str

Close the flag

Mutation(N,Pi,Nk,f(x))

Input:

N: the total number of binary trees

Pi: the pointer of Pi

Nk:: the node of Pj tree

f(x): the content for mutation

Output: A new binary tree

Method:

Flag is execute mutation operation

Copy Pi and insert to the last

If Pi.Nk.ID& 1=0 Pi.Nk.PointFront.PointLeft = f(x)

Else Pi.Nk.PointFront.PointRight = f(x)

Close the flag

For the latter case: the Mutation operation is defined as: Mutation(N,Pi,Nk,f(x)) , with N indicating the total number of the binary trees, 'Pi' representing the pointer of the to-be-operated binary tree, 'Nk' representing the node of the tree, and f(x) representing the binary tree which is used to replace the sub-tree of Nk.

Since the resolution of the conflicts of the two cases of mutation is similar, in the following content we only use the case one to illustrate the process.

In order to support the operation transformation which will be introduced in the following part, the above problems are saved into the log of the site except the operation of Set-score. The priority of the operations are defined as: P(Set-score) > P(Crossover) = P(Mutation)=P(Delete) > P(Pause) > P(Select). When two or several operations are conflict, they are forced to be executed by the sequence of their priorities.

4. Consistency Maintenance Strategy

One problem that I have discussed in [9], it will be disappeared in this paper, because crossover and mutation operations are contain two processes. If other operation is execute between the two processes, the conflict will be occurred. So the same method be adopted in this paper to void the conflict.

4.1. Conflict Analysis

Table 1. Conflict Table

	Set-score	Select	Crossover	Mutation	Delete
Set-score	⊙	⊙	⊙	⊙	⊘
Select	⊙	⊘	⊙	⊙	⊙
Crossover	⊙	⊙	⊘	⊘	⊘
Mutation	⊙	⊙	⊘	⊘	⊘
Delete	⊘	⊙	⊘	⊘	⊘

The conflict relationship of different operations is critical to the design of the consistency maintenance algorithm. Thus we define the conflict relations among the five operations by reference to [15] (see Table.1).

Definition1. Conflict Relation:

Two operations, O_1 and O_2 are conflict (denoted as $O_1 \oslash O_2$), iff different execution orders of O_1 and O_2 result in different document states [15].

Definition2. Compatible Relation:

Two operations, O_1 and O_2 are compatible(denoted as $O_1 \odot O_2$), iff they don't have a conflict relation [15].

In order to handle the conflicts described in Table 1, an auxiliary operation is

introduced here, which is named as Pause operation. Defintion 3 gives the definition of it.

Definition3. Pause Operation:

Pause operation means that the current user is not allowed to do any operation until Select operation finishes. If one site needs to do Select operation, Pause operation must be done before the execution of the Select operation for we hope that the selective objects are the selective result of all sites. The Pause operation will be broadcast to all the other sites after its generation. When the other sites receive remote Pause operations, they will do the Score operation and broadcast its Pause operation. The Select operation will be executed only when all sites have confirmed that they have executed the Pause operations. After executing the Select operation, the flag that indicated the Pause status will be reset. By using Pause operation, Select and other operations will have no conflicts anymore.

The Pause operation is described in the following.

4.2. Consistency Maintenance of the Genetic Operation

The Algorithm for Solving the Conflict：

According to the analysis of the conflicts, we establish the corresponding resolution algorithm for each pair of the conflicts.

The introduction of Pause operation guarantees that the final operations in the queue can only be one or several Select operations, since the site can't create other operations after Pause have been executed.

Two operations in LO_1 and LO_2 can transform directly only when they have the same created document status, otherwise we need to transform them by COT algorithm in advance. LO_1 are the operations in queue and the LO_2 are the operations in Log [15].

LOT(LO_1,LO_2)

Input:

LO_1: One operations set

LO_2: One operations set

Output: A new operations set

Method:

var LO_1'=[]

Repeat until LO_1=[]

if LO_2=[] then { LO_1:= LO_1'+ LO_1;return;}

else{

remove O_x from LO_1,where O_x is the first element in LO_1

removeO_y from LO_2,where O_y is the first element in LO_2

If $O_x \oslash O_y$ then{LO_x:=T(O_x,O_y);LO_y:=T(O_y,O_x);}

else{LO_x:=O_x;LO_y:=O_y;}

LOT(LO_x,LO_2);

LO_1':=LO_1'+LO_x;LO_2:=LO_y+LO_2;}

return LO_1';

T_{O1O2}(O_1,O_2)

Input:

O_1: Operation O_1 (can be Set-score(), Delete(),Select(),Crossover() and Mutation())

O_2: Operation O_2 (can be Set-score(), Delete(),Select(),Crossover() and Mutation())

But both of O_1 and O_2 can't be Crossover or Mutation at the same time;

Output:

O: the operation after transform

Method:

transform two operation and make them no conflict;

return the last operation;

For example if O_1= Set-score(P_i,S_n,Numx) and O_2= Delete(P_j), we can through the follow method to solve the conflict:

if i>j then return [Set-score(P_{i-1},S_n,Numx)];

else if i=j then return [];

else return [Set-score(P_i,S_n,Numx)];

if O_1= Crossover(N,P_i,N_g,P_j,N_k) and O_2= Delete(P_n,f(x)), we can solve the conflict by this:

if n<i then return [Crossover(N-1,P_{i-1},N_g,P_{j-1},N_k)]

elseif n>i and n<j then return [Crossover(N-1,P_i,N_g,P_{j-1},N_k)]

elseif n=i or n=j then

return [undo(Delete(P_n,f(x)), Crossover(N,P_i,N_g,P_j,N_k),redo(Delete(P_n) ,f(x)];

else return [Crossover(N,P_i,N_g,P_j,N_k)];

The other operations we can also solve the conflict by the above method.

If O_1 and O_2 are Crossover or Mutation at the same time, we can use the follow method to solve the conflict.

COT-Tree(O,DS)

Intput:

O: the current operation

DS; the current document status

Output:

Null

Method:

if O= Crossover(N,P_i,N_g,P_j,N_k) **then**{

f(X_i)=Copy(P_i);f(X_j)=Copy(P_j);

O_i=Insert(P_{N+1},f(X_i));O_j=Insert(P_{N+1},f(X_j);

/**N is the total number of the binary tree when insert is executed*/*

Execute(COT-DO(O_i,DS)) }

*/*As Crossover is a transaction operation consisting of two CopyInsert and a Cross operation, so O_j and Cross are executed orderly without conflict*/*

elseif O= Mutation(N,P_i, N_k,Str) **then**

$f(X_i)$=Copy(P_i); O_i=Insert(P_{N+1},$f(X_i)$);

Execute(COT-DO(O_i,DS))

/**As Mutation is a transaction operation that contains a CopyInsert and a Mutation, so there is no conflict in mutation*/

It's important to note that when two conflict operations intend to insert objects to the same position, a rule is made that the operation with the smaller ID is inserted first.

In replicated architecture, each site maintains a local copy of the shared document, and we use COT algorithm to maintain the consistency of the standard operations (such as Delete, Insert *etc.*,). Besides that, we design a set of rules to solve the conflicts of operations such as Set-score, Select, Crossover and Mutation in the evolution environment, which is quite different from text editor field.

The process of operation transform：

Each site runs four processes named as Create_Operation process, Broadcast_Operation process, Receive_Operation process and Execute_Operation process. Create_Operation is responsible for handling the release of local operations, Broadcast_Operation is used to broadcast operations from local sites to the remote one, Receive_operation is responsible for receiving operation from other sites and saving them. And Execution_Operation is designed to tackle the execution process of operations which are causally-ready. The operation that has been executed will be saved into the log of the site, and the operation that has not been executed will be saved into the queue of the site. The following gives a description of these operation processes.

CreateOperation()

Input: null
Output: a new operation

Method:

/*Create the operation when Pause is 0*/

if Pause = 0 then

/*if the operation is Select then create Select and Pause operations*/

if Operation is Select then Create Select and Create Pause operations;

else Create Operation;/*Create operation directly if the operation is not Select*/

else return;}/*Don't create operation when Pause is not 1*/

BroadcastOperation()

Input: null;

Output:null

Method:

Spread the operations to the other Sites;

ReceiveOperation()

Input: operation

Output: put the operation into the queue

Method:


```
    /*executing the operation immediately if the operation is Set-score*/
    if the Operation is Set-score then Execute(Set-score) after it arrived
    else append the Operation to the wait queue of the site
ExecuteOperation()
Input: operation
Output: null
Method:
    /*Set Score directly when operation is Set-score*/
      if Operation is Set-score then Execute(Set-score);
      else/*transform the operation*/
            while Flag = 0{sleep(1);} /*the flag of the transaction*/
            if Operation is Select then
/*judging the condition whether meet to execute the select condition*/
                  while JudgeStatus()=0 {sleep(1)};
                  Execute(OT-Select())}
      else
            Execute(LOT(O,Log));/* Introduced in algorithm 2*/
            Log =Log+O;
```

5. Integrated Example

Example 1:

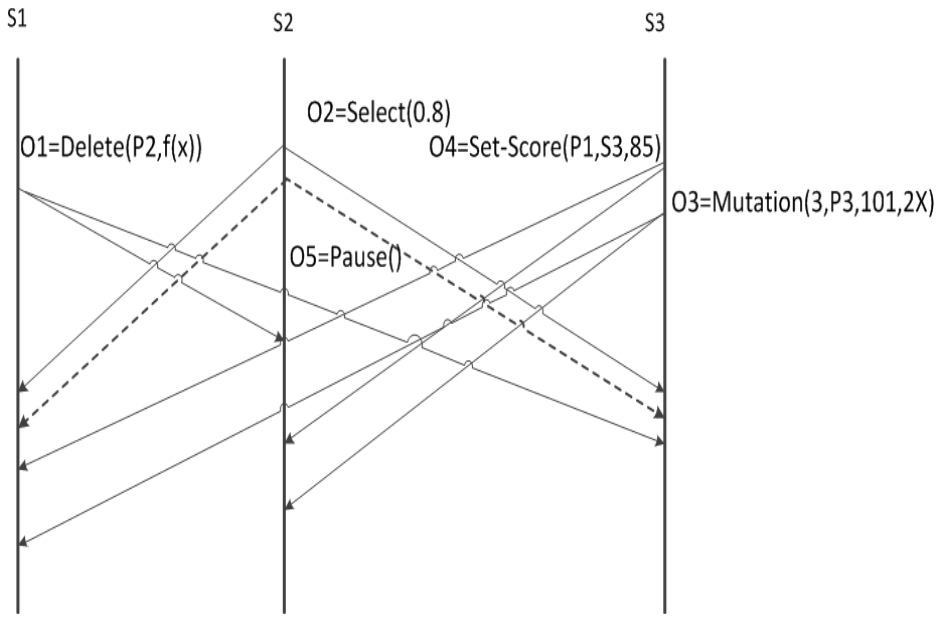


Figure 8. Scenario 2 for Concurrent Operations

The original status of the document is the same as that in example 1. There are also three sites: S_1, S_2 and S_3. Operations are O_1 = Delete(P2,f(x)), O_2 = Select(0.6), O_3 = Mutation(3,P_3,101,x), O4 = Set-score(P_1,S_3,90), O_5 = Pause() respectively. Suppose the default scores of the three functions are {<original,40>}, {<original,50>}, {<original,70>}. The analysis of the example two is as below, The execution process of site2 is described.

Site2:

1.O_2 is a Select operation and the pause statuses of all sites don't satisfy the execution condition of Select. thus the Select operation is moved to the queue of the site2 and O_5= Pause() is created. Then Pause() is executed and spread to other sites. The status flag of Site2 is set to 1, and then Site2 cannot create operation before executing Select operation.

2.O_1 arrives and executes as-is result in P_1, P_2'. P_2 is deleted, resulting in $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}, y_2 = \frac{1+x_2}{x_2}$.

3.O_4 =Set-score(P_1,S_3,90) arrives: Since P_1 exists , it isn't conflict with other operations according to the second part of V. O_4 will be executed immediately, and insert the <Site2,90> into the score of P_1.

4.When O_3 arrives: 1) because the documents statuses are the same when O_1 and O_3 are generated, LOT([O_3],[O_2,O_1,O_4]) is invoked. Since $O_3 \oslash O_1$, $O_3 \odot O_2$ and $O_3 \odot O_4$, T_{mutdel} is called resulting in [O_{31}, O_{32}, O_{33}] = [Undo(Delete(P_2,f(X_2)), Mutation(3,P_3,101,2X_3), Redo(Delete(P_2,f(X_2))]. 2) Execute Undo(Delete(P_2,f(X_2)) resulting in P_1,P_2,P_3. Execute Mutation(3, ,P_3,101,2X_3): Create the copy of P_3 and insert it into the array as P_4, mutate P_4 resulting in $y_4 = \frac{1+x_4}{2x_4}$ and the score of P_4 is set to default score(suppose the default score is 50). Execute Redo(Delete(P_2,f(X_2)) and result in P_1,P_2',P_3'(P_2' is the original P_3 and P_3' is the mutation result of P_4). Thus , the final result at site2 is: $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}, y_2 = \frac{1+x_2}{x_2}$, $y_3 = \frac{1+x_3}{2x_3}$ and the scores of them are {<original,40>,<Site2,90>}, {<original,70>}, {<original,50>} respectively.

Site1:

1.O_1 is generated at the local site and then executed as-is result in P_1,P_2'. The original P_2 has been deleted and result in $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}, y_2 = \frac{1+x_2}{x_2}$.

2.When O_2 = Select() arrives, it will be inserted it into the queue of site1. After that Pause() operation is executed on site1 and broadcasted to other sites (We didn't show all of it in Figure 7, because the lines were too complex). The flag of Site1 is set to 1 and then no operations can be created from Site1 until the flag turns to 0.

3.When O_4 and O_3 arrive: The transformation process of it is somewhat like that on site2. And we don't give detailed description here. The result is: $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}, y_2 = \frac{1+x_2}{x_2}, y_3 = \frac{1+x_3}{2x_3}$. And the scores of them are {<original,40>,<Site2,90>}, {<original,70>}, {<original,50>} respectively.

Site3:

1.O_4 and O_3 are executed as-is result in: $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}$, $y_2 = x_2^2 * (x_2 + \cos(x_2))$, $y_3 = \frac{1+x_3}{x_3}, y_4 = \frac{1+x_4}{2x_4}$,and the scores of them are:

{<original,40>,<Site2,90>}, {<original,70>},{<original,50>,{<original,50>}.

2.When O_2 = Select() arrives: 1) It is firstly inserted into the queue of site3, Pause() operation is executed on site3 and broadcasted to other sites just as the operation of O_5 (reference part 2 of Site1). The flag of Site3 is set to 1 nd then no operations can be created from Site3 until the flag turns to 0.

3.When O_1 arrives: Since $O_1 \odot O_4$, $O_1 \odot O_3$, $O_1 \odot O_2$, LOT([O_1],[O_4,O_3,O_2]) is first invoked and result in:O_1' = O_1. Execut O_1 resulting in $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}$, $y_2 = \frac{1+x_2}{x_2}, y_3 = \frac{1+x_3}{2x_3}$, and the scores of them are: {<original,40>,<Site2,90>},{<original,70>}, {<original,50>}.

At last: the final document status are the same, and the Pause() operation on all sites have been executed. Since JudgeStatus()=1, OT-Select() is invoked. Since there is only one Select operation in the queues, execute this Select operation directly on all sites. Calculate the efficient scores resulting in 65, 70, 50, and eliminate products that the scores are low in 40 percent (eliminate:$y_3 = \frac{1+x_3}{2x_3}$). Get the last document status as: $y_1 = \log\left(x_1^2 + \sin(x_1)\right) + x_1^{\cos\left(\frac{x_1}{3}\right)}, y_2 = \frac{1+x_2}{x_2}$, clear all queues, logs on the sites and all scores are set as defaulting score.

6. Conclusions and Future Work

The efficiency of evolution on one single site is too low. The communication on traditional collaboration evolution can't meet the demand of real-time collaboration. Besides, traditional collaboration strategy in text editor field doesn't provide corresponding solutions for evolutionary operations. Although OT algorithm can provide consistency maintenance for operations on the linear document model, it can't satisfy the operations in the evolution environment. Based on the characteristics of the evolution environment, the document model is first be modeled as hybrid document model, which formats all the objects in the design environment into linear structure, and represents the details of a single object with a binary tree. New operations are defined and a consistency maintenance algorithm is given to guarantee the identity of the shared document in the replicated architecture. The algorithm is based on the operation transform algorithm (including OT, Undo/Redo, COT *etc.*), and the algorithm can maintain the consistency evolutionary operation on theory.

The work of this article doesn't allow the modification of the single object (that's the tree structure). Other ongoing and future work is to add the consistency maintenance of this Update operation.

Acknowledgments

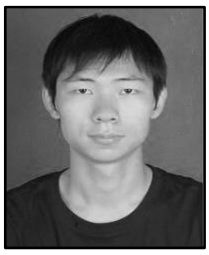
The authors would like to thank the anonymous reviewers for their constructive comments and suggestions on improving the presentation of this work. This work is supported in part by the National Science Foundation of China under Grant No. 61202376, "Chen Guang" project sponsored by Shanghai Municipal Education Commission and Shanghai Education Development Foundation under Grant No.10CG49, and Innovation Program of Shanghai Municipal Education Commission under Grant No. 13YZ075.

References

[1] J. Byrne, E. Hemberg, M. O'Neill, "Interactive operators for evolutionary architectural design", Proc. ACM Companion on Genetic and Evolutionary Computation, **(2011)**; Dublin.
[2] P. Bentley, "Evolutionary Design by Computers", Morgan Kaufmann Pub, **(1999)**, pp. 219-279.

[3] P. Villacorta, D. Pelta, "Evolutionary design and statistical assessment of strategies in an adversarial domain", Proc. IEEE The Conference on Evolutionary Computation (CEC'10), **(2010)**; Barcelona.
[4] E. Benkhelifa, M. Farnsworth, A. Tiwari, "Evolutionary Algorithms for Planar MEMS Design Optimisation: A Comparative Study", Nature Inspired Cooperative Strategies for Optimization, **(2010)**; Berlin Heidelberg.
[5] I. Goryanin, "Computational optimization and biological evolution", J. Biochemical Society Transactions, vol. 38, no. 5, **(2010)**, pp. 1206-1215.
[6] J. Holland, "Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence", A Bradford Book, **(1992)**.
[7] S. Wang, "Research on Consistency Maintenance Technique on Non-Linear Document Model", Shanghai: University of ShangHai for Science and Technology, **(2014)**.
[8] C. Z. Sun, C. Ellis, "Operational transformation in real-time group editors: issues, algorithms, and achievements", Proc. ACM Computer Supported Cooperative Work, **(1988)**; Seattle.
[9] S. Greenberg and E. Chang, "Computer support for real time collaborative work", Proc. Numerical Mathematics and Computing, **(1989)**; Manitoba.
[10] L. Hattori and M. Lanza, "Syde: A tool for collaborative software development", Proc. ACM/IEEE Software Engineering, **(2010)**; Cape Town.
[11] C. Ellis, S. Gibbs, "Concurrency control in groupware systems", Proc. ACM Special Interest Group on Management of Data, (**1989)**; Oregon.
[12] C. Sun and D. Chen, "Consistency maintenance in real-time collaborative graphics editing systems", J. Transactions on Computer-Human Interaction, vol.9, no.1, **(2002)**.
[13] D. Sun and C. Sun,"Operation context and context-based operational transformation", Proc. ACM Computer Supported Cooperative Work, **(2006)**; Canada.
[14] S. Wang, C. Wu, L. Gao, "Research on Consistency Maintenance Technique on Evolutionary Design Base on Tree Model", Journal of Chinese Computer Systems, **(2014)**.
[15] C. Sun and D. Xu, "Operational transformation for dependency conflict resolution in real-time collaborative 3D design systems", Proc. ACM Computer Supported Cooperative Work, **(2012)**; Bellevue.
[16] A. Prakash, M. J. Knister, "A framework for undoing actions in collaborative systems", J. Transactions on Computer-Human Interaction, vol.1, no.4, **(1994)**, pp. 295–330.
[17] M, Ressel and R. Gunzenhäuser, "Reducing the problems of group undo", Proc. ACM Supporting Group Work, **(1999**); Phoenix.

Authors



Shanshan Wang (1988-) graduated from University of Shanghai for Science and Technology, China with a master degree in 2014 in Technology of Computer Application. And he obtained his BSc degree in 2011 at Hubei University of Arts and Science. His current research interests include CSCW, collaborative design and collaborative computer.

Liping Gao(1980-) graduated from Fudan University, China with a PhD in 2009 in Computer Science. She received her BSc and master degree in Computer Science from Shandong Normal University, China in 2002 and 2005 respectively. She is doing her research work in University of Shanghai for Science and Technology as an assistant professor. Her current research interests include CSCW, heterogeneous collaboration, consistency maintenance and collaborative engineering.

Sizheng Zhu(1980-) China with a master degree in 2009 in Computer Science. He is doing his research work in University of Shanghai for Science and Technology as an engineer. His current research interests include Database and data mining, cloud computing, collaborative computing and access control etc.

Chunxue Wu (1964-) received the Ph.D. degree in Control Theory and Control Engineering from China University of mining and technology, Beijing, China, in 2006. He is a Professor with the Computer Science and Engineering and software engineering Division, School of Optical-Electrical and Computer Engineering, University of Shanghai for Science and Technology, China. His research interests include, wireless sensor networks, distributed and embedded systems, wireless and mobile systems, networked control systems.