

# Maintaining Semantic Intention of Step-wise Operations in Replicated CAD Environments

Liping Gao, Bin Shao, Tun Lu and Ning Gu

Department of Computing and Information Technology, Fudan University, Shanghai, P.R. China  
{lipinggao, binshao, lutun, ninggu}@fudan.edu.cn

## Abstract

Two kinds of operations are discovered in CoAutoCAD system: immediate one and step-wise one. The execution of the former (such as Line, Box etc.) is completed in one step while the latter (such as Mirror, Array) takes some continuous or discrete steps and the number of the steps can even not be predicted in some scenarios (such as Copy-Paste). So, during the execution process, the intention of the step-wise operations may be violated by concurrent operations coming from remote sites. This paper discusses the phenomenon and proposes to maintain the semantic intention of operations by introducing a novel approach called CLPF (Check Last Predict Future) to solve the violation happened before while predicting the future using the newly-updated information. In order to get the execution format of causally-ready operations, GOTO algorithm has been extended with VT function reflecting the influence of one operation on the other one.

**Keywords:** CoAutoCAD, intention of step-wise operation, constrain maintenance

## 1. Introduction

CoAutoCAD is developed to support a group of designers editing shared documents from different sites over the Internet simultaneously. In order to achieve high responsiveness to support unconstrained collaboration, the replicated architecture is adopted [1,2], in which local operations are executed immediately after their generation while remote ones are first transformed to include or exclude the effects of corresponding operations in the history buffer (HB) [2]. Many issues are studied in this field including consistency maintenance [1, 2], group undo [3], conflict resolution [4] etc. All the issues mentioned above are based on the consistency models [2] which are introduced to guarantee the correctness of a replicated groupware system, within which the CCI model of Chengzheng Sun is the most famous[2]. CCI is the abbreviation of Convergence, Causality-preservation, and Intention-preservation with convergence ensuring that after the executing of the same operations at all sites, all copies of the shared document are identical,

causality-preservation ensuring that for any pair of operations  $O_a$  and  $O_b$ , if  $O_a \rightarrow O_b$ ,  $O_a$  must be executed before  $O_b$  at all sites and intention-preservation ensuring that for any operation  $O$ , the effects of executing  $O$  at all sites are the same as the intention of  $O$  and the effect of executing  $O$  doesn't change the effects of independent operations.

In order to meet the requirements of the consistency model, OT algorithm is proposed by Ellis and then polished by Chengzheng Sun[2], Du Li[1], Norrie[6] etc. OT algorithm tries to maintain the CCI model by transforming operations' position parameters. Up to now, however, only five types of operations including Insert, Delete [2], Update [5], Group/Ungroup [6] and Undo [5] can be manipulated directly with OT. As for complex operations (such as Copy-Paste, Mirror, Array), before they are executed, they are decomposed into several primitive operations firstly and then executed in several steps, during which the semantic meaning of the complex operation may be destroyed by concurrent operations coming from other sites. Take Mirror operation as the example. Several designers are struggling to complete the design of the outlook of a building. When one designer completes the sculpture design of a window, another one launches the Mirror operation to create another window with different position. However, during the mirror process, the former finds that the height of the window is too large and shortens it. Then confusion may occur when all the operations have been executed at all sites, since the latter just wants to array two identical windows with different positions and doesn't care about the attributes of the source window. Yet, he gets two different windows on the same wall.

So, the objective of the paper is to analyze the intention violation of step-wise operations in graphical group editors (such as CoAutoCAD) which adopts the replicated architecture, explain the generation reason, and discuss the possible solutions for that.

The rest of the paper is organized as follows. The second part gives the detailed description of a typical scenario where intention violation of step-wise operations occurs. The third part analyzes the reason for the phenomenon and introduces CLPF strategy to solve it. Example analysis is given in part four. Comparison with related work is introduced in the fifth part and the

major contribution and future work of our research is summarized in the last section.

## 2. Typical Scenario

Chengzheng Sun has pointed out in [7] that “Current transformation algorithms are only capable of handing fine-grain primitive operations, such as Insert

and Delete. Useful editors, however, must offer to the end user higher level compound operations, such as Move and Replace “. Norrie [6] added group/ungroup operations in graphical editors to extend the process scope of OT. However, that’s not enough. In fact, operations can be divided into different classes according to different criteria (See Table 1).

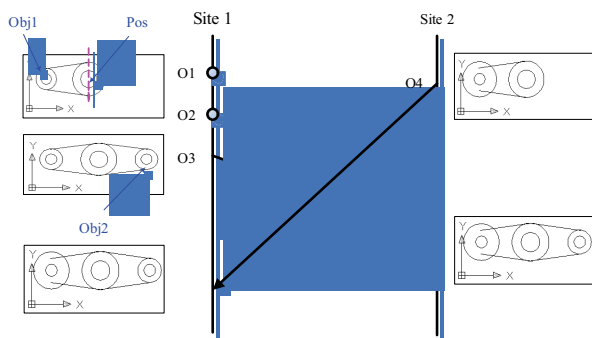
Criteria	Class name	Class description
Execution location	Local-exe operation	Operations which are executed at local sites and don’t need to be synchronized with other sites. Such as copy, select, pointer move etc.
	All-exe operation	Operations which both local and remote sites should execute and synchronize. Such as Create, Delete, Update, Paste, Move etc.
Complexity	Primitive operation	Operations which can be manipulated directly by OT algorithm. Such as Insert, Delete, Update, and Undo.
	Complex operation	Before their execution, they must first be decomposed into one or several primitive operations. Such as Copy-Paste, Move, Mirror, Array etc.
Execution time	Single-step operation	Operations which need only one step to be executed. Such as Create, Delete, Update, Undo etc.
	Step-wise operation	Operations which need several steps to be executed. Such as Copy-Paste, Move, Offset, Mirror etc.

**Table 1 classification of operations**

Step-wise operations can be divided into three phrases: objects select (we call objects here source objects), parameters enact and objects create or update (we call objects here target objects). Current process of different kinds of operations of OT framework [2] is as follows:

1) Local-exe operations are executed at local sites and don’t need to be broadcast to other sites. Only local-exe operations can influence all-exe operations but the vice versa is not the case;

2) Step-wise operations are decomposed into primitive ones and then broadcast to remote sites and during the decomposition process, no information indicating the relationship of original operations and primitive ones is saved. For example, Mirror will be decomposed into Select, Parameter\_set and Insert operations. Because of the lose of semantic meaning of the original operation, the relationship between source objects and target objects can not be maintained. So the intention of step-wise operations may be violated with operations generated simultaneously from remote users. Look at the following scenario.



**Fig.1 The intention violation of Mirror operation**

**Scenario1:** Four operations are named as  $O_1$ ,  $O_2$ ,  $O_3$

and  $O_4$  where  $O_1$ ,  $O_2$  and  $O_3$  are the decomposed operations of a  $Mirror(obj1, pos, obj2)$  operation with  $O_1=Select(Obj_1)$ ,  $O_2=Select(Pos)$ ,  $O_3=Insert(Obj_2)$  and  $O_4=Update(Obj_1, old\_pos, new\_pos)$ . At site 1, when  $O_4$  comes, it will be transformed against  $O_3$  and we get the result  $O_4'=O_4$ . After all operations have been executed at both sites, the result violates the intention of user 1 to construct a symmetrical object of  $Obj_1$  according to  $Pos$ .

In the above scenario, although two sites have reached the same results, the intention of user 1 is violated because the constraint between the two objects implied by the mirror tool can not be maintained by conflict solving strategy based on OT. In order to solve the problem, there are two points that must be maintained.

- 1) The constrain relationship between the source objects and target ones should be maintained;
- 2) Local-exe operations should be updated with newly coming All-exe operations.

## 3. CLPF strategy

Based on the framework of OT [2], we add HB of Original Op and Local\_exe Op to store the relationship between original operation and decomposed ones (Fig. 2). The Local\_exe operation will be transformed and re-executed when All-exe operations destroy the content or the attributes of the target objects of Local\_exe one. Further more, VT function is introduced to transform an operation against the one that is manipulating the objects it refers to. And undo-vt-do-vt-redo procedure is introduced to control the execution of remote operations. The following part gives the detailed discussion of the strategy.

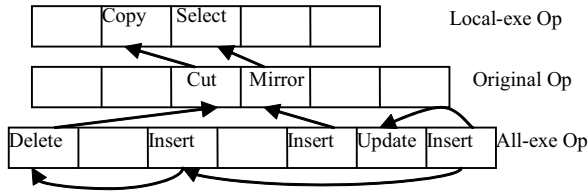


Fig.2 History buffer

### 3.1 Modifying the Operation Generation Process

When an operation is released by user interface, the operation generation process will pack the format of the operation to the one that can be processed directly by OT algorithm. If it is a primitive one, it will be executed immediately and added to the All-exe Op HB; if it is a step operation of a complex operation, two conditions may exist: if it is a Local-exe Op, the Op will be executed immediately and the packed operation will be appended to the end of the Local-exe Op HB; if it is an All-exe one, the local-exe Op of the ordinal Op will first be re-executed to refresh the execution environment of the newly generated Op and then the packed Op will also be appended to the All-exe HB (See Function 1).

```

Function 1 Generate Oa: O
{
  If Oa is a primitive Op {
    O=Pack(Oa);
    Execute(O);
    For each Oi in Local-exe Op HB
      Oi=IT(Oi, O);
    Append(All-exe HS, O); }
  Else if Oa is a Local-exe Op {
    O=Pack(Oa);
    Append(Local-exe HS, O); }
  Else {
    Olocal=O.OriginalOp.Local-ExeOp;
    Re-execute(Olocal);
    O=Pack(Oa);
    Execute(O);
    For each Oi in Local-exe Op HS
      Oi=IT(Oi, O);
  }
}

```

### 3.2 Introducing VT Function

VT(Virtual Transformation) is somewhat similar to IT [7] in the sense that the impact of one operation will be reflected on the other. However, it modifies not the parameters of the obj of the first operation but that of the referenceobj. The following gives the pre-/post-conditions of the VT function.

VT(O<sub>a</sub>, O<sub>b</sub>):O<sub>a</sub>'

Precondition for input parameters: O<sub>b</sub>  $\parallel_c$  O<sub>a</sub> //O<sub>b</sub> is constrain concurrent with O<sub>a</sub>;

Postcondition for output: O<sub>b</sub>  $\parallel_c$  O<sub>a</sub>', and the attribute values of referenceObj referenced to by O<sub>a</sub> and manipulated by O<sub>b</sub> is replaced with the updated ones.

Table 2 gives the possible conditions where VT function may be needed and Function 2 to 6 gives the detailed description of it.

O <sub>a</sub> \ O <sub>b</sub>	Insert	Delete	Update
Insert	IT, ET, VT	IT, ET, VT	/
Delete	IT, ET, VT	IT, ET, VT	/
Update	IT, ET	IT, ET	VT

Table 2. Possible transformations of O<sub>a</sub> according to O<sub>b</sub>

```

Function 2. VT_II(Oa, Ob):Oa' {
  Oa.referenceObj.Range.HigherBound += Ob.Obj.Range.Length;
  Oa.Obj=GetContent(Oa.referenceObj.Range);
  Return Oa;
}
Function 3. VT_ID(Oa, Ob): Oa' {
  Oa.referenceObj.Range.HigherBound -= Ob.Obj.Range.Length;
  Oa.Obj=GetContent(Oa.referenceObj.Range);
  Return Oa;
}
Function 4. VT_DI(Oa, Ob): Oa' {
  Oa.referenceObj.Range.HigherBound += Ob.Obj.Range.Length;
  Return Oa;
}
Function 5. VT_DD(Oa, Ob): Oa' {
  Oa.referenceObj.Range.HigherBound -= Ob.Obj.Range.Length;
  Return Oa;
}
Function 6. VT_UU(Oa, Ob): Oa' {
  Oa.new_value= ReConstruct(Oa.old_value, Ob.new_value);
  Return Oa;
}

```

### 3.3 Defining the Constraint Concurrent Relation between Operations

**Definition 1.** Constraint Concurrent Relation ' $\parallel_c$ ': O<sub>a</sub> is constraint concurrent with O<sub>b</sub>, expressed as O<sub>a</sub>  $\parallel_c$  O<sub>b</sub> or O<sub>b</sub>  $\parallel_c$  O<sub>a</sub> if: (1) O<sub>a</sub>  $\parallel$  O<sub>b</sub> [3]; (2) O<sub>b</sub> is one of the decomposed operation of a step-wise one with a series of operations expressed as <O<sub>1</sub>, O<sub>2</sub>, ..., O<sub>k</sub>, O<sub>k+1</sub>, ..., O<sub>n</sub>> where O<sub>1</sub>...O<sub>k</sub> are the Local-exe operations of source objects for the following operations O<sub>k+1</sub>, ..., O<sub>n</sub> and O<sub>b</sub> is one of O<sub>k+1</sub>, ..., O<sub>n</sub>; (3) The change of Obj<sub>a</sub> in O<sub>a</sub> will lead to the change of Obj<sub>b</sub> in O<sub>b</sub> because of the dependence relationship of Obj<sub>a</sub> with referenceObj<sub>b</sub> and that of Obj<sub>b</sub> and referenceObj<sub>b</sub>.

**Property:** The constraint concurrent relation of the two operations is transitive, that's to say if O<sub>a</sub>  $\parallel_c$  O<sub>b</sub> and O<sub>b</sub>  $\parallel_c$  O<sub>c</sub>, then O<sub>a</sub>  $\parallel_c$  O<sub>c</sub>. And here we indicate it as O<sub>a</sub>  $\Rightarrow$  O<sub>b</sub> to express the relationship between O<sub>a</sub> and O<sub>c</sub>. For the length of this paper, the proof of the property is omitted here.

### 3.4 Get the Execution Form of operations and Their Execution Context

In GOTO algorithm [2, 7], the execution form of O (indicated as EO) and the execution context EC(O) can

be got by performing IT and ET transformation on HB. It transforms HB(O) into such an equivalent HB(O') that all operations causally preceding O are positioned before independent operations in HB(O'). Let HB(O')=HB(O)'.left+ HB(O)'.right, where HB(O)'.left is the sub-list of causally preceding operations, and HB(O)'.right is the sub-list of independent operations. Then O will be inclusively transformed against the list of independent operations in HB(O)'.right and that EC(O')=HB(O'). However, in order to ensure the constrain relationship between constrain concurrent operations: 1) the constrain following operation must first be undone if it has been in the HB to get the execution context of O; 2) the constrain concurrent following operation must execute VT transformation against the preceding one to reflect the effect of the former. So that, the HB(O') is like that HB(O')=HB(O)'.left+ HB(O)'.middle+ HB(O)'.right where HB(O)'.left is the sub-list of causally preceding operations, HB(O)'.middle is the sub-list of constrain concurrent preceding operations as well as the concurrent operations while HB(O)'.right is the sub-list of constraint concurrent following operation and that EC(O')=HB(O)'.left+ HB(O)'.middle. The extended algorithm is described in function 7 and 8.

Function 7 GOTO2(O,L):EO,M  
O: a causally-ready operation  
L: the list of operations [EO<sub>1</sub>,EO<sub>2</sub>,...EO<sub>n</sub>] in EC(O).  
EO: the execution form of O.  
M: the execution context index.

1. Scan L[1,m] from left to right to find the first operation EO<sub>k</sub> such that EO<sub>k</sub>||O. If no such an operation is found, then return EO:=O.
2. Scan L[k,m] from left to right to find all the operations causally preceding O. If no such operation is found, goto 4
3. otherwise, let L1=[EO<sub>c1</sub>, EO<sub>c2</sub>,...EO<sub>cr</sub>] be the list of operations in L[k,m] which are causally preceding O,  
for each i from 1 to r do  
LTranspose(L[k+i-1,ci]);
4. Scan L[k+r,m] to construct the constrain following operations tree EOms=FindFollowing(O,L[k+r,m]). If no such operation is found, goto 6
5. otherwise, for each j from n to 1 do  
LTranspose(L[k+r+pj,m-n+j])
6. for each l from k+r+1 to m-n do  
If L[l]  $\xrightarrow{C}$  O then  
O=IT(VT(O,L[l]));  
Else O=IT(O,L[l]);
7. M=n-m;

Function 8 FindFollowing(O,L):TF  
O: a concurrent operation  
L: the list of operations [EO<sub>1</sub>,..., EO<sub>n</sub>] where EO<sub>i</sub>||O  
TF: the tree of operations[EO<sub>p1</sub>, EO<sub>p2</sub>,..., EO<sub>pn</sub>] which satisfies O  $\Rightarrow$  EO<sub>pi</sub> and that only when operation is  $\xrightarrow{C}$  the other, the former can be the ancestor of the latter.  
Scan L[1,m] to find all the operations which satisfies that O  $\xrightarrow{C}$  EO<sub>k</sub> if no such operation is found, return null;  
otherwise, suppose R=[EO<sub>c1</sub>, EO<sub>c2</sub>,...EO<sub>cr</sub>] be the search result.  
Let LF.append(R);  
L.delete(R);  
For i=1 to r do {

```
R=FindFollowing(EOci,L);
LF.append(R);
}
```

### 3.5 The control-algorithm of the execution of remote operations


The execution process can be described as undo-vt-do-vt-redo process (See Procedure 1 and 2).

Procedure 1 Execute O:  
If O is not causally-ready {Queue(O);}
Else {
GOTO2(O,L,EO,M);
Undo the constraint concurrent following operations CCOs recursively from bottom to top of the layered tree;
Execute(EO);
For each O<sub>1</sub> in Local-exe Op HS
O<sub>1</sub>=IT(O<sub>1</sub>, O);
VT(EO, CCOs);
For each Os in CCOs {
Redo CCOs which are undone.
}
}

Procedure 2 VTF(O, TOs):  
TOs is the construction tree of operations [EO<sub>p1</sub>, EO<sub>p2</sub>,..., EO<sub>pn</sub>] which satisfies O  $\Rightarrow$  EO<sub>pi</sub> and that only when operation is  $\xrightarrow{C}$  the other, the former can be the ancestor of the latter.  
If TOs is null return;  
TOs.O=IT(VT(TOs.O, O));  
For each child TOs' of TOs do {
VTF(O, TOs');
}


## 4. Example Analysis

In this section, we use one example to illustrate how the whole strategy works. Scenario in Figure 1 is used here. Let's S<sub>j</sub><sup>i</sup> denotes the document state at site i after executing j operations and let's (S<sub>j</sub><sup>i</sup>) EO<sub>x</sub>○EO<sub>y</sub> denotes the application of operations EO<sub>x</sub> and EO<sub>y</sub> in sequence on S<sub>j</sub><sup>i</sup>. The execution form of each operation and the document state after its execution at different sites are illustrated as folloes.

→Site1: S<sub>0</sub><sup>1</sup> = 

(1)Execute O<sub>1</sub>, O<sub>2</sub> and O<sub>3</sub>:

EO<sub>1</sub>=O<sub>1</sub>; EO<sub>2</sub>=O<sub>2</sub>; EO<sub>3</sub>=O<sub>3</sub>;

S<sub>3</sub><sup>1</sup>=(S<sub>0</sub><sup>1</sup>)EO<sub>1</sub>○EO<sub>2</sub>○EO<sub>3</sub>= 

(2)Execute O<sub>4</sub>:

Since O<sub>4</sub>  $\xrightarrow{C}$  O<sub>3</sub>, O<sub>3</sub> will first be undone;

HB(O<sub>4</sub>)={EO<sub>1</sub>,EO<sub>2</sub>,EO<sub>3</sub>} and EO<sub>1</sub>||O<sub>4</sub>, EO<sub>2</sub>||O<sub>4</sub>,

O<sub>4</sub>  $\xrightarrow{C}$  O<sub>3</sub>, and the transformed HB will become:

HB(O<sub>4</sub>)'={EO<sub>1</sub>,EO<sub>2</sub>}; EC(O<sub>4</sub>)=HB(O<sub>4</sub>)'; EO<sub>4</sub>=O<sub>4</sub>;

EO<sub>3</sub>'=IT(VT(EO<sub>3</sub>,EO<sub>4</sub>),EO<sub>4</sub>)= Insert(obj2'); //Obj2' is

the modified object2 according to Obj1'

$$S_4^1 = (S_3^1) \overline{EO_3} \circ EO_4 \circ EO_3' = \text{Diagram}$$

And after that,  $O_1 = VT(O_1, EO_4) = \text{Select}(\text{Obj1}')$ ;

$$\rightarrow \text{Site2: } S_0^2 = \text{Diagram}$$

(1) Execute  $O_4$ :

$EO_4 = O_4$ ;

$$S_4^2 = (S_0^2) EO_4 = \text{Diagram}$$

(2) Execute  $O_3$ :

$HB(O_3) = \{EO_4\}$  and  $O_4 \parallel \xrightarrow{c} O_3$ ;  $EC(O_3) = HB(O_3)$ ;

$EO_3 = IT(VT(O_3, EO_4), EO_4) = \text{Insert}(\text{obj2}')$ ; //Obj2' is the modified object2 according to Obj1';

$$S_3^2 = (S_4^2) EO_3 = \text{Diagram}$$

From the above analysis, we can see that all sites can reach the same results. And the two editions can preserve the intentions of both Mirror operation and Update operation. Limited by the length of this paper, only a simple example is analyzed here, for more detailed description of complex examples; please consult to the following paper.

## 5. Comparison to Related Work

The aim of this paper is to maintain the semantic meaning of complex operations by maintaining the constraints between different objects of the document. As for constraints maintenance, there are some research works. CAB [8], SAMS [9], and CoDesign [10], CoCSE [11] are related to constraint control in collaborative environments. CAB presents an active rule based approach to modeling user-defined semantic relationships in collaborative applications. However, it doesn't mention the settlement of complicated problems such as constraint violations and that CAB's purpose is to maintain the constraints between different objects but not those between different operations. SAMS achieves semantic consistency by integrating semantic constraints to the operational transformation approach. However, SAMS is based on XML resources. Whether it can be used in other environments is not certain. Moreover, it does not ensure constraint satisfaction. CoDesign intends to achieve semantic preservation in real-time collaborative graphic systems and devises semantic expression to express constraints. However, it cannot represent temporal constraints between objects. CoGSE adopts a strategy which is able to maintain both the tree structure constraint and system consistency in the face of concurrent operations. However, it can only support the static constrain but not the dynamic one. Moreover, the above systems are all targeting to process

the constrain relationship between different objects; no one is attempting to process that between different operations.

## 6. Conclusion and Future Work

The paper first analyzes the operation classification involved in graphical group editors and finds out the fact that step-wise operation's semantic meaning may be disturbed by concurrent operations coming from remote sites since other people can not be cautious about what actions others are taking. Based on the analysis, the paper introduces the overall solving strategy involving the generation of new operation, the capture of causally-ready operation context, as well as the whole control algorithm to realize the execution of operations and examples is also simulated.

In future, more detailed research will be done to have an understanding of how to maintain the design rules by defining the constrain relationship between different operations in computer-aided design field, so that, forcing the automatic satisfaction of the rules. Also, we'll apply this new algorithm into the application of CoAutoCAD so as to get more insight into different kinds of operations in the practical application environments.

## Acknowledgements

The work is supported by National Natural Science Foundation of China (NSFC) under Grant No.90612008 and National Grand Fundamental Research 973 Program of China under Grant No. 2005CB321905.

## References

- [1] D. Li and R. Li. "Preserving operation effects relation in group editors," In *Proc. of ACM Conference on CSCW 2004*, pp. 437 - 446, Chicago, USA, Nov 2004.
- [2] C. Sun, X. Jia, Y. Zhang, Y. Yang, and D. Chen. "Achieving convergence, causality-preservation, and Intention-preservation in real-time cooperative editing systems," *ACM Transactions on Computer-Human Interaction*, 5(1):63 - 108, Mar 1998.
- [3] Matthias Ressel, Rul Gunzenhauser: "Reducing the Problems of Group Undo," In *Proc. of ACM Conference on Supporting Group Work*, pp, 131-139, Nov. 1999.
- [4] Chengzheng Sun and David Chen. "A multi-version approach to conflict resolution in distributed groupware systems," In *Proceedings. 20th International Conference on Distributed Computing Systems*, pp:316-325, 10-13 April 2000.
- [5] David Sun, Steven Xia, Chengzheng Sun and David Cheng. "Operational Transformation for Collaborative Word Processing". In *Pro. Of the ACM Conf. on Computer-Supported Collaborative Work.*, pages , Nov. 2004.
- [6] Claudia-Lavinia Ignat, Moira C. Norrie, "Grouping in

Collaborative Graphical Editors”. In Proc. Of the ACM Conf. on Computer-Supported Collaborative Work., pp: 447-456 , Nov. 2004.

[7] Chengzheng Sun, Clarence Ellis, ”Operational Transformation in real-time group editors: issues, algorithms, and achievements”. In Proc. Of the ACM Conf. on Computer-Supported Collaborative Work., pages 59-68, Nov. 1998.

[8] Li, D. and Patrao, J. (2001): ‘Demonstrational customization of a shared whiteboard to support user-defined semantic relationships among objects’, ACM GROUP’01, Sept. 30-Oct. 3, 2001, Boulder, Colorado, USA, pp. 97-106.

[9] Skaf-Molli Hala, Molli Pascal and Ostér Gerald. (2003). Semantic consistency for collaborative systems, the Fifth International Workshop on Collaborative Editing Systems Hosted by the 8th European Conference of Computer-supported Cooperative Work, Helsinki, Sept. 15, 2003.

[10] Wang, X.Y, Bu J.J and Chen, C. (2002). Semantic preservation in real-time collaborative graphics designing systems, The Fourth International Workshop on Collaborative, New Orleans, USA, 2002.

[11] Kai Lin, David Chen, Chengzheng Sun and Geoff Dromey. Maintaining constraints in collaborative graphic systems: the CoGSE approach. In Proceedings of the Ninth European Conference on Computer-Supported Cooperative Work, 18-22 September 2005, Paris, France, 185-20.