Journal of Universal Computer Science, vol. 17, no. 14 (2011), 1887-1912
submitted: 15/05/11, accepted: 28/9/11, appeared: 1/10/11 © *J.UCS*

Achieving Transparent & Real-time Collaboration in Co-AutoCAD Application

Liping Gao
(University of Shanghai for Science and Technology, Shanghai, PR China
Shanghai Key Laboratory of Computer Software Evaluating and Testing, Shanghai, PR China
lipinggao@fudan.edu.cn)

Tun Lu
(Fudan University, Shanghai, PR China
lutun@fudan.edu.cn)

Abstract: In order to support the real-time collaboration between geographically distributed designers, the single-user application AutoCAD is required to be transformed transparently into groupware system by adopting fully replicated architecture. As the core issue to maintain the consistency of the distributed replicas, traditional consistency maintenance algorithms (such as Operation Transformation and Address Space Transformation algorithms), however, support only linear data model, and may lead to low algorithm efficiency and small operation types when adapted to the collaborative design field. In this paper, a novel layered document model is proposed to abstract the document model of AutoCAD, and the AST algorithm is adapted according to the model to achieve transparent & real-time collaboration. Moreover, the Update conflicts resolution based on child-precedence strategy, and the database listening technique used to grasp the semantics of interface operations to realize operation adaptation are also presented. Efficiency analysis of the Layer-AST algorithm is given, showing the improved performance of the algorithm. Finally, the system architecture of Co-AutoCAD using this strategy is detailed to guide the application.

Keywords: Layered document model, Data model transformation, Operation adaptation, Consistency maintenance, Conflict resolution
Categories: J.6

1 Introduction

Traditional consistency maintenance algorithms (such as operation transformation (OT) technology [Ellis and Gibbs,1989;Sun et al.,1998,2006; Li and Li,2004;Shao et al.,2010] and address space transformation (AST) technology [Gu et al.,2005; Yang et al.,2005] in replicated groupware environments[Ellis and Gibbs,1989; Sun et al.] usually assume the document model as linear structure, which use the index value of the object node as the unique identifier of the object. With this model, transparent adaptation (TA) technology [Xia et al. 2004; Sun et al. 2005] provides the possibility to transform current off-the-shelf single-user applications into groupware systems without modifying the applications' source codes. TA technology makes use of the API (Application Programming Interface) of the applications to interpret and decompose the interface instructions, maps the application document model into linear structure, and translates the remote operations into ones that can be

implemented by evoking the applications' APIs. TA technology provides a solution to bridge different applications with unified engine layer, thus improving the reuse of the consistency maintenance algorithm.

TA technology is mainly used in text editor field, with linear structure as the data model of its engine algorithm (i.e. OT algorithm), leading to the low efficiency and neglecting the logical data models of specific applications. Sun et. al have discussed the TA technology in text editor field in detail [Xia et al. 2004; Sun et al., 2005; Lin et al. 2007; Shen et al., 2007]. They propose to map the document model of the editor field into linear structure, use OT technology to maintain the document consistency, and analyse the API interface of the applications and provides strategies to interpret interface instructions.

In order to adapt this technology to the CAD environment, however, three key issues must be re-discussed: data model transformation, consistency maintenance algorithm and operation adaptation, since the logical document model and the APIs in CAD environment are quite different with those of text editor field, where formatting all documents models into linear structure will lead to low algorithm efficiency. Data model transformation is responsible for mapping the CAD document model into the layered document model in order to improve the algorithm efficiency and enrich the operations types the engine algorithm can support. Consistency maintenance is the core module of the engine layer which needs to be modified according to the new data model. Meanwhile, operation adaptation is responsible for intercepting the instructions released from the user interface of AutoCAD and changing their formats so as to be adapted to the engine algorithm.

This paper analyses the logical document model of AutoCAD, proposes to abstract it into a layered document model and modifies AST algorithm to support this new model. Based on this model, the paper proposes to use database listening technique to intercept user instructions and backup technology to access the parameters of the Update operations by comparing the old and new document states.

The rest of this paper is organized as follows. Section 2 surveys the related work of collaboration between applications. Section 3 gives a brief description of TA technology. Section 4 describes the novel data model transformation process from CAD document model to the layered document model. Section 5 describes how the AST algorithm is adapted to the new data model. Section 6 elaborates on the operation adaptation process and introduces child-precedence strategy to solve Update conflicts in the layered document model. Section 7 discusses the Co-AutoCAD system architecture and Section 8 summarizes the paper and gives an outlook for the future work.

2 Related work

There have been many efforts in the research field of cooperation between applications. The DistEdit [Knister and Prakash, 1990] provides a set of primitives to be added to the applications to provide collaboration support, without dealing with distributed system issues. Although the primitives provided by toolkits are generic enough to support editors with different user interfaces, it adopts not the replicated architecture but the client/server one, which means it can only support loosely-coupled collaboration but not tightly-coupled collaboration. Dewan and A. Sharma

(1999) have done series of experiments in order to export protocols, techniques and lessons as for the interoperation between multiple users and proposed to use a language-independent system such as CORBA to allow different systems to communicate with each other. However, this method can only be used between applications that use the compatible communication protocols. The PSI [Palfreyman et al., 1999] combines a simple data model with active sharing mechanisms to construct an active sharing infrastructure to support information sharing across a number of distributed applications. The platform is realized as an active layer on top of an extended space. The Placement Documents [LaMarca et al.,1999] is somewhat similar with PSI in that it also needs a data repository to store the edited documents and a middleware to monitor the change of the common data. Both PSI and Placement Document perform well in loosely-coupled environment. On the contrary, in tightly-couple environment such as the CAD engineering process, where the control objects' granularity should be as small as possible, they don't work very well. Besides that, because of the common data repository, the replicated architecture cannot be supported naturally. The Disciple framework [Krebs et al., 2003], presents a framework for building collaborative applications for clients with different display and processing capabilities. However, its main purpose is to reduce the influence of the slow and/or unreliable connections of different clients and its structure is also based on client/server architecture. Li and Li(2002) proposes intelligent collaboration transparency technology, which is based on application black box assumption to transform single-user applications into collaborative groupware by series of steps, such as event capture, event reduction, consistency maintenance, event reproduction and event replay. This method can solve the conversion of some relatively easy application with simple function set. However, to realize the powerful functions of commercial off-the-shelf applications by event capturing is almost impossible. Li and Lu (2006), later, modified the former method by replacing the event capture strategy with series of document state accesses and state playbacks. However, not all applications support the CTRL +A, CTRL +C, CTRL +V or CTRL+SHIFT+HOME, CTRL+SHIFT+END key-board shortcuts.

Since many mature commercial applications have provided plenty of APIs, Sun et. al (2004) first introduces the idea of transparent leverage of single-user application to multi-user application. However, in order to use the OT framework [Xia et al. 2004; Sun et al., 2005; Lin et al. 2007; Shen et al., 2007, Zheng et al., 2009], he formats all the document models both in text editor field and graphical editor field into linear structures, thus reducing the algorithm efficiency greatly. Our previous work [Gao et al., 2008] tries to adapt the consistency maintenance technology from text editor field to CAD groupware field; however, the main focus in that work is to deal with the heterogeneity of different applications, also neglecting the algorithm optimization.

3 TA technology

TA technology [Xia et al. 2004; Sun et al. 2005] doesn't need to modify the source codes of the application. It realizes transparent adaptation by interacting with the API interface and performing series of operation decompositions and reconstructions. TA technology is composed of three layers: Single-user Application Layer (SA), Collaboration Adaptor (CA) and Generic Collaborative Engine (GCE), with SA

indicating current single-user applications, CA bridging SA and GCE by smoothing the differences of data models and operation formats between them as the middle ware, so as to reach the purpose of transparent adaptation (see Fig. 1).

Since different applications have different APIs and different data models, it's hard to construct a common middleware as the CA components for all applications. It must be designed according to different APIs, different data models and different algorithms the engine layer adopts. In this paper, the document model of the CAD environment is first analyzed and then mapped into layered structure in order to achieve high efficiency of consistency maintenance algorithm. Based on the layered document model, AST algorithm is modified to support this new model and database listening technology is adopted to get the necessary parameters of the operations.

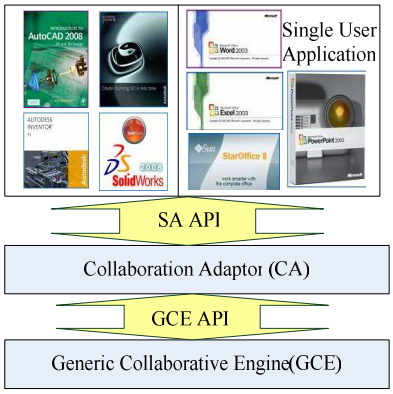


Figure 1: TA transformation framework [Xia et al. 2004; Sun et al. 2005]

4 Data model transformation

This section describes the logical document model of CAD, gives the definition of the layered document model, and describes the mapping process from the logical document structure of CAD to the layered structure.

4.1 The layered document model

The document of AutoCAD application is composed of multiple layers (see Fig. 2). Every **Document** comprises multiple **Layers**. Every **Layer** comprises one or several simple or complex **Entities** and **Blocks**, with each **Block** comprising one or several simple or complex **Entities**. Simple **Entity** is composed of multiple **Vertexes**, whose information can be got by retrieving the attributes information of the **Entity**. Designers can release Insert, Delete and Update operation targeted at each layer node object and the interface operations can be decomposed into multiple operations with target objects being distributed over different layers. For instance, after a **Trim** operation is executed by a **Pline** object upon another one, the latter **Pline** object is

decomposed into two **Pline** ones. Thus , the **Trim** operation should be divided into the Update and Delete operations of the Vertex object as well as the Insert operation of a new **Pline** object (Fig. 3).

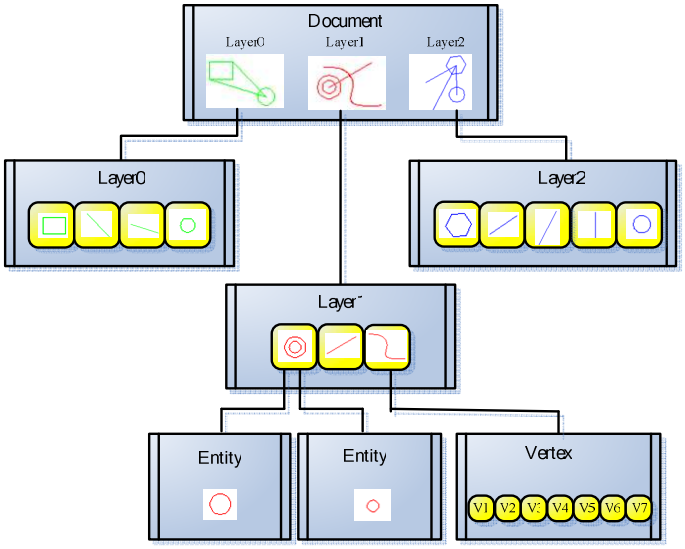


Figure 2: The logical document structure of AutoCAD

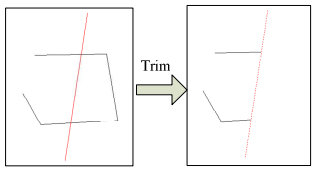


Figure 3: The Pline object is decomposed into two objects by the Trim Operation

The paper introduces to map the logical document model of CAD into the layered document model (Fig. 4). In this model, every ***Node*** is composed of one or several ***Elements***, with each ***Element*** attached with a linear operation history list indexed by its timestamp. The unique identifier of every Node ***Element*** is defined as $<P_1,P_2,\ldots,P_i>$ ($1 \le i \le 5$), with P_j ($1 \le j \le i-1$) indicating the index value of the Node's parent Node Elements at layer j, P_i indicating the index value of the Element at current Node. As can be seen in Fig. 4, P_1 indicates the index of the ***Document*** layer, P_2 indicates the index of the ***Layer*** layer, P_3 indicates the index of the ***Block*** layer, P_4 indicates the index of the ***Entity*** layer and P_5 indicates the index of the ***Vertex*** layer. For example, the Node Element Vertex <2,1,1,2,4> of Fig. 4 indicates the 2th Document's 1th Layer's 1th Block's 2th Entity's 4th Vertex.

It must be pointed out that the identifier of the Node Element is relevant with specific document status. Node Elements may have different identifiers in different contexts. For example, if a Delete operation is executed on Entity <2,1,1,1> (Fig. 4),

the identifier of Entity<2,1,1,2> will change to Entity<2,1,1,1>, and the child Node Elements of Entity<2,1,1,2> Vertex<2,1,1,2,1>, Vertex<2,1,1,2,2>, Vertex<2,1,1,2,3> and Vertex<2,1,1,2,4> will change to Vertex<2,1,1,1,1>, Vertex<2,1,1,1,2>, Vertex<2,1,1,1,3> and Vertex<2,1,1,1,4> respectively.

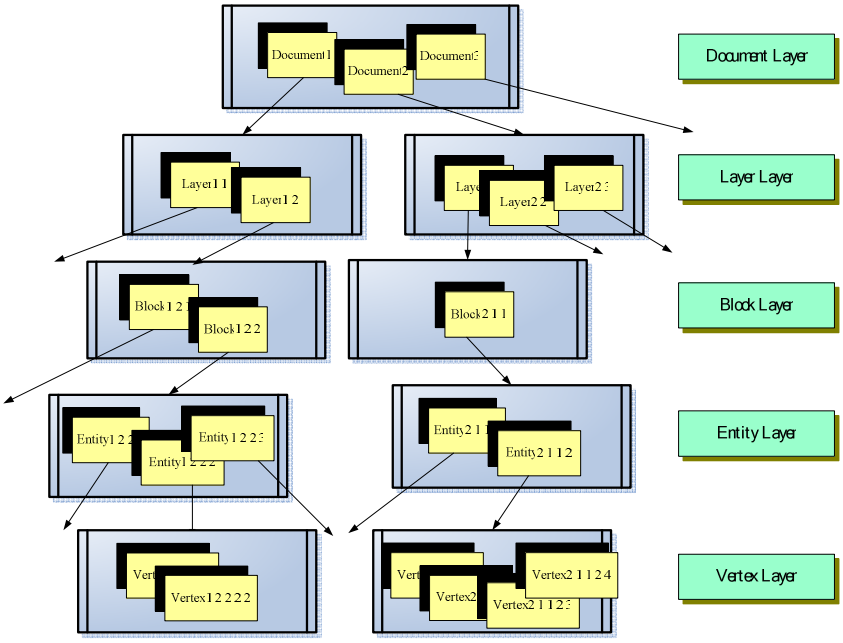


Figure 4: The layered document model

The **Node** definition is described in Definition 1 and the Layered Document Model definition is given in Definition 2.

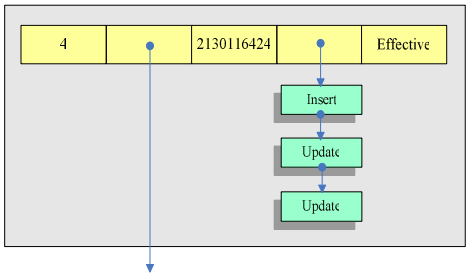


Figure 5: The Node Element's Structure

Definition 1: Node

In the layered document model (Fig. 4), Node N is the linear structure $\{E_1, E_2, \ldots, E_n\}$, with every Element E=<level, childPtr, entityID, historyPtr ,mark>:

1) level: Presenting the layer index of N, with level = 1, 2, 3, 4 and 5, indicating Document, Layer, Block, Entity and Vertex respectively.

2) childPtr: Indicating the pointer of the direct child Node of the current Node Element.

3) entityID: The unique identifier of Element E in the local session.

4) historyPtr: Indicating the operation list targeted at current Node Element.

5) mark: Indicating the validity of current Node Element, which can be set as "effective" or "ineffective".

Definition 2: Layer-Doc

A Layer-Doc is composed of n Nodes (n is a definite integer). It can be null (n=0), or be composed of one root Node and k child Layer-Docs, which are not intersected with one another. We use Layer-Doc to present the unique identifier of a layer document, use Layer-Doc (P_1, P_2, …, P_i) to present the unique identifiers of a Node at layer i+1 (1 ≤ i ≤4), with P_j (1 ≤ j ≤ i) indicating the index value of its parent Node Element at layer j. For example, the identifier of the Node which includes Block<1,2,1>, Block<1,2,2> is Layer<1,2>. The Layer-Doc model has the following characteristics:

1) Every Node Element has one and only one Insert operation;

2) The Insert operations' timestamps of the child Nodes Elements are always larger than those of their parent Nodes, which means that only after the generation of a parent Node can its child Nodes Elements be generated;

3) Before deleting a parent Node Element, its entire child Node Elements must be deleted previously.

4.2 Mapping the document structure of AutoCAD to the layered structure

The AutoCAD document is stored in its inner database (See Fig.6). In order to transform it into the layered document structure, the Layers, Blocks, Entities and Vertexes of it must be mapped into Nodes (which are composed of lists of Elements). Section 3.2.1-3.2.3 describes the mapping processes of the Layer, Block, Entity and Vertex objects in AutoCAD document respectively.

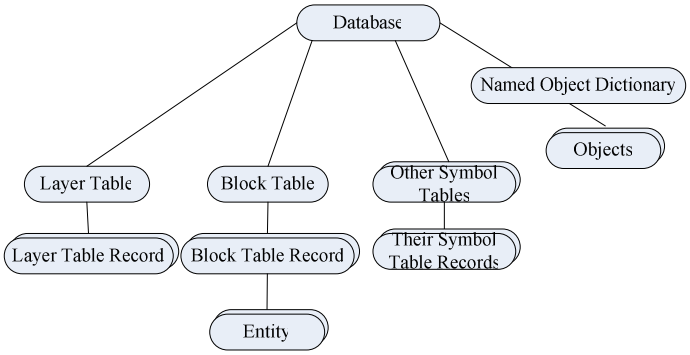


Figure 6: The document structure of AutoCAD

4.2.1 Mapping the Layer objects into the Nodes Elements

The ***Layer*** information is stored in the AcDbLayerTable. Fig. 7 describes the adding process of ***Layers*** in this table. Since Insert records are appended to the AcDbLayerTable according to the chronological order, we have no methods to maintain identical AcDbLayerTables in distributed sites if Insert operations arrive in different order. Take the case in Fig. 8 as an example, after two operations O_1 and O_2 released from site 1 and site 2 concurrently are executed in both sites(see Fig.8), the document states are different from each other (see Fig. 9). Therefore, we cannot make use of the index values of the Insert records in the AcDbLayerTable as the unique identifiers of different Layers. Additional data structure must be used to maintain the unique identifiers of different Layers. Fig. 10 describes the process to identify different Layers by adding an extra linear structure.

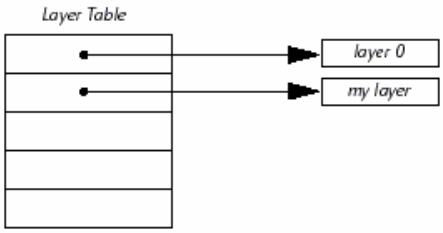


Figure 7: the AcDbLayerTable in AutoCAD

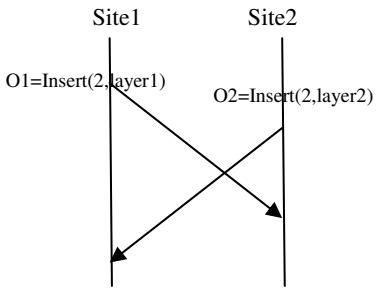


Figure 8: the Execution orders of Layer Insert operations

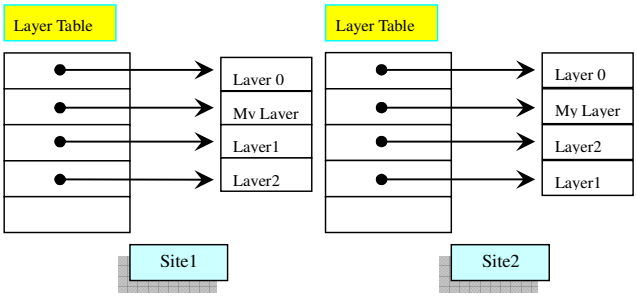


Figure 9: the Execution results of the Layer Insert operations

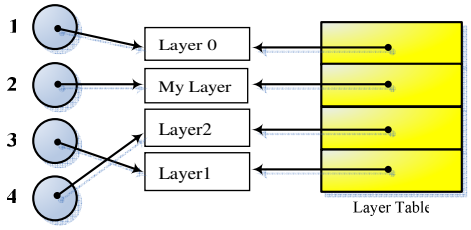


Figure 10: Use the index value of the Layer Insert record in the linear structure as the Layer's unique identifier

4.2.2 Mapping the Block and Entity objects into the Nodes Elements

The ***Block*** information is stored in the AcDbBlockTable, which provides AcDbBlockTableRecordIterator to achieve the information of every Block. Each Block contains one or several Entities or nested Blocks. Here we still use the index value of every Insert record of Blocks in the linear structure as the unique identifiers of the objects (see Fig. 11).

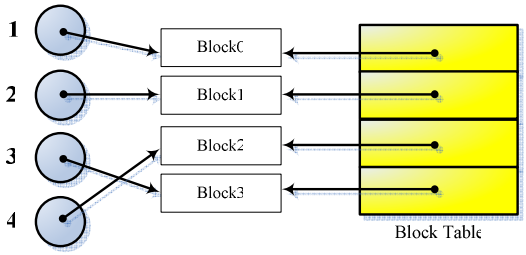


Figure 11: Use the index value of the Block Insert operation in the linear structure as the Block's unique identifier

The ***Entity*** information is stored in MODEL_SPACE of AcDbBlockTableRecord, which provides AcDbBlockTableRecordIterator to get access to each Entity. Entities

have one or several attributes, with Color, Linetype, Linetype scale, Visibility, Line weight, and so on as the common ones and one or several specific ones. For example, Circle Entity has the attributes of Center and Radius etc. and Line entity has the attributes of StartPoint, EndPoint and Thinkness, etc.

4.2.3 Mapping the Vertex objects into the Nodes Elements

The ***Vertex*** belongs to the geometrical feature of the Entity, which is not stored in tables (like the Layer, Block and Entity objects). AutoCAD provides the getSubentPathsAtGsMarker function to get the AcGi pointer so that Vertex information can be accessed by using this pointer. When executing Update operation, the unique identifier of the Vertex of the target object is accessed by using the getSubentPathsAtGsMarker function. The getSubentPathsAtGsMarker function is declared as following:

pEnt->getSubentPathsAtGsMarker(AcDb::kVertexSubentType, marker, pickpnt, xform, numIds, subentIds);

Parameter AcDb::kVertexSubentType indicates the sub-entity types with desirable values of Vertex, Edge, etc.; Pickpnt and Xform provide some extra information for some Entities types such as mplines etc.; numIds stores the number of the vertexes which are to be inquired, and subentIds points to the handler of the Vertexes. The following procedure describes the process to get access to the Vertex identifiers and how to map them into the linear structure.

Procedure 1 Map2Linear()
Map the Vertexes of Entities from the AutoCAD storage structure into the linear structure

```
1:AstNode *vertex=new AstNode[100];
2: int marker;
3:AcGePoint3d pickpnt;
4:AcGeMatrix3d xform;
5:int numIds;
6:AcDbFullSubentPath *subentIds;
7:pEnt->getSubentPathsAtGsMarker(AcDb::kVertexSubentType, marker, pickpnt, xform, numIds, subentIds);
8:for (int i = 0;i < numIds; i++) {
9:     AcDbEntity *pEntCpy = pEnt->subentPtr(subentIds[0]);
10:    AcDbObjectId objId;
11:    vertex[i]=objId;
12: }
```

5 Improved consistency maintenance algorithm based on AST

AST [Gu et al.,2005] is a consistency maintenance algorithm used in text editor. The AST strategy retraces the document's address space to the state at the time of the operation's generation when concurrent operations are executed. In that state, the index of the object of the operation can be discovered immediately. The retrace

process does not affect the order of the objects in the linear document, but determines the position of the target object of a new operation.

The AST algorithm assumes a document as a linear structure composed of characters. Each character may have one or several operations targeting at itself while each operation targets at only one character. The operation together with its timestamp is saved in the character's node of the linear structure. Apart from the information of the characters and corresponding operations, effective/ineffective mark information is also appended to every node in the linear structure. In order to adapt the AST strategy to the layered document model, modification of it must be proposed.

5.1 The Layer-AST algorithm

In the Layer-Doc model, the retracing process of AST strategy doesn't need to cover all the objects of the document, but only the parent Nodes and the current Node of the target object of an operation O. The purpose of the retracing process is to retrace the state of the parent Nodes (from layer 1 to level(NODE)-1) to the state where O is generated. Only one Node at each layer needs to be concerned with during the retracing process, thus improving the algorithm efficiency dramatically. If one Node Element is marked as 'Ineffective', all of its children Node Elements are marked as 'Ineffective' implicitly. The purpose of the retracing process is to mask the execution effect of concurrent Insert and Delete operations of operation O. The Update operation is not considered here since it doesn't influence the index value of the Node Elements. Suppose the timestamps of the Insert and Delete operations are SV_{ins} and SV_{del} respectively, the retracing process judges the validity of Node N according to the following conditions [Gu et al.,2005]:

1) If $SV_{ins} \geq SV_o$, N is set as Ineffective;

2) If $SV_{ins} \leq SV_o$, and there is not Delete operations targeted at Node N, N is set as Effective;

3) If $SV_{ins} \leq SV_o$ and $SV_{del} \geq SV_o$, N is set as Effective;

4) If $SV_{ins} \leq SV_o$ and $SV_{del} \leq SV_o$, N is set as Ineffective.

As for the Insert and Delete operation attached with Element E, SV_{ins} and SV_{del} must satisfy $SV_{ins} \leq SV_{del}$, therefore, the conditions of $SV_{ins} \geq SV_o$ and $SV_{del} \leq SV_o$ don't exist. If several Delete operations are targeting at N, N is set to Ineffective if one of the Delete operations satisfies that $SV_{del} \leq SV_o$. It is easy to conclude that the operations after SV_o don't influence the mark of the Node. Therefore, if SV is constant, the mark of the Node Element is also definite.

The retracing process of the Layer-Doc is described in procedure 2, with Fig. 12 describing an example of two concurrent Update and Delete operations.

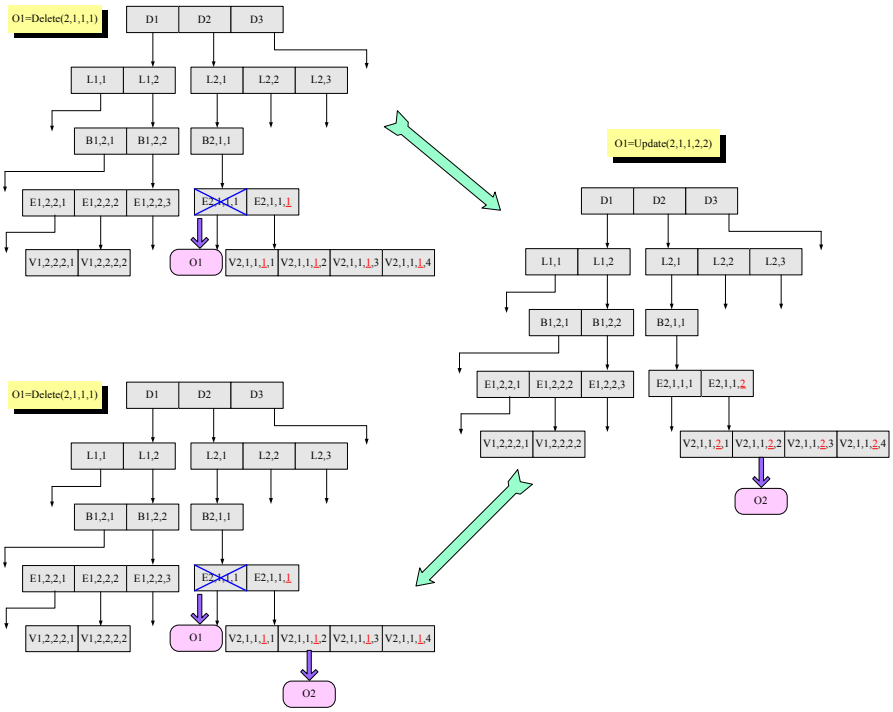


Figure 12: The retracing process of an Update operation

Procedure 2 Retracing_LayerDoc (Layer-Doc, level, SV_o, O), Retrace the document status of Layer-Doc to status when O is generated

// Check the validity of every Element at every Layer Node from up to down of the Layer-doc
//Suppose O.pos={p_1,p_2,…,p_{n-1}, p_n}
1: If level==length(O.pos)
2: Return;
3: for every element E_i of Layer-Doc {
4: Set E_i ineffective;
5: Consider the Insert O_{ins} of E_i, if O_{ins} is timestamped by SVO_{ins} and $SVO_{ins} \leq SV_o$ {
6: set E_i effective;
7: }
8: For each Delete O_{del} of E_i {
9: if O_{del} is timestamped by SVO_{del} and $SVO_{del} \leq SV_o$ {
10: set E_i ineffective
11: }
12: }
13: Count the effective Elements from left to right of the Layer-Doc to find out

```
the p[level] Element, and indicate it as E[p[level]];
14: Layer-Doc=E[p[level]].childPtr;
15: Retracing (Layer-Doc, level+1, SV_o, O )
16: }
```

In Fig.12, O_1 and O_2 are two concurrent operations released from two sites. After O_1 is executed at site1, the object E<2,1,1,1> is set to be ineffective, so that the identifiers of E<2,1,1,2> and its child Nodes Elements V<2,1,1,2,1>, V<2,1,1,2,2>, V<2,1,1,2,3> and V<2,1,1,2,4> turn to be E<2,1,1,1>, V<2,1,1,1,1>, V<2,1,1,1,2>, V<2,1,1,1,3> and V<2,1,1,1,4> respectively. If no strategy is adopted to guarantee the consistency, the object (V<2,1,1,2,2>) of operation O_2 cannot be targeted at correctly.

5.2 Efficiency analysis of the improved algorithm

Procedure Retracing_LayerDoc (Layer-Doc, level, SV_o, O) process retraces the document status to SV_o. The retracing process doesn't need to cover all Elements of the Layer-Doc, but cover only one Node of each layer. Therefore, the algorithm efficiency is improved dramatically.

Suppose the average length of very Node is h, which implies that there are h Elements that need to be checked to be set the 'Effective' or 'Ineffective' mark. Suppose the operation number of every Node Element is d, which implies that the check cost of every Element is O(d). Thus, the efficiency of retracing a Node can be defined as O(d·h) and the whole efficiency of the retracing process is O(h·level·d), where the value of level is decided by the layer of operation O.

The retracing process used in AST needs to check all the Elements at the bottom of the layered document. According to the assumption of the length of every Node, the Elements of the bottom layer is $h^{level-1}$, therefore, the retracing process of AST is $O(h^{level-1} \cdot d)$, which is definitely higher than that of the retracing process of Layer-AST.

6 Operation adaptation

This section describes the definitions of primitive operations of CAD environment, the extension of Layer-AST algorithm to support Update operation, and the conflict definition and solution in the layered document model. This section also describes the procedure to get the parameters of the primitive operations based on the database listening technique.

6.1 Primitive operations of CAD

All the objects in the CAD environments can be mapped into Nodes Elements of the layered document model, with $<P_1, P_2, \ldots, P_n>$ as the unique identifier of the object. Based on this address space, three primitive operations are defined as follows:

1) Insert(P, type, param): Insert an object at the P^{th} position, with ***type*** indicating the object type, and ***param*** indicating the parameters of the objects;

2) Delete(P, type, param): Delete the object at the P^{th} position, with ***type*** indicating the object type, and ***param*** indicating the parameters of the objects;

3) Update(P, key, old_val, new_val): Modify the value of the key of the P^{th} position from old_val to new_val, with ***key*** indicating the attribute type, such as Color, Position, etc. and ***old_val*** and ***new_val*** indicating the attribute values before and after the Update execution respectively.

Here, '.' is used to indicates the parameter of a certain operation. For instance, O.P indicates the position parameter of operation O; O.type indicates the type of operation O. The above definitions can support both Do and Undo operations, since the type, param, old_val, and new_val are stored in the Delete and Update operations, providing necessary information to define the inverse operations of those operations. The inverse operations of the three operations are defined as follows:

1) $\overline{Insert}$ (P, type, param)=Delete(P, type, param);
2) $\overline{Delete}$ (P, type, param)=Insert(P, type, param);
3) $\overline{Update}$ (p, key, old_val, new_val)=Update(p, key, new_val, old_val).

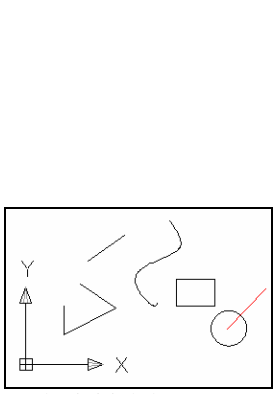
6.2 Conflict resolution of the Update operation

After a remote Update operation arrives, if it satisfies the causally- ready precondition, Retracing (Layer-Doc, 1, SV_o, Update) process will first be called, and the target object is found according to O.P (Fig. 13). Only Insert and Delete operations are handled in Retracing process, where the intention conflict of Update operations is not detected, So Update conflict detection and resolution technique must be discussed in addition. Firstly, let's have a look at the conflict definition in OT framework [3,4].

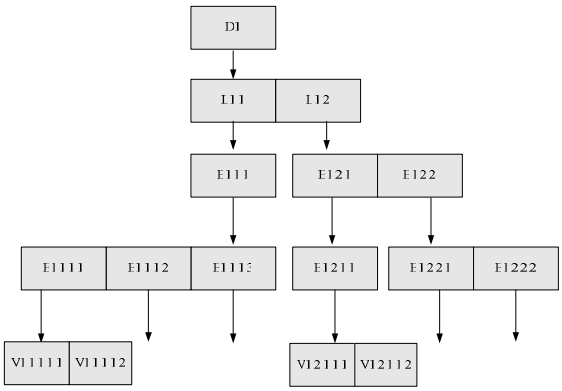
Definition 3: Conflict Operation ($\otimes$)[Sun and Sun,2004]

Two Update operations U_a and U_b conflict with each other (indicates as $U_a \otimes U_b$), iff: (1) $U_a \parallel U_b$; (2) U_a and U_b are manipulating the same objects ($U_a.P=U_b.P$); (3) U_a and U_b are modifying the same attribute type ($U_a.key = U_b.key$).

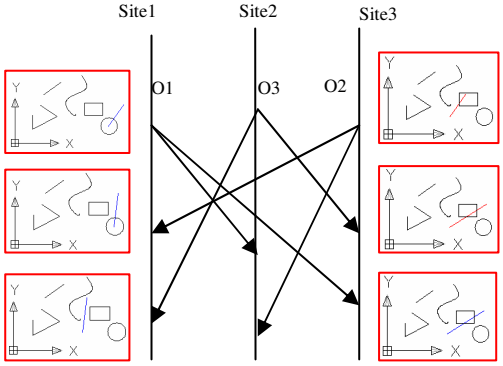
All the objects in the OT framework are located at the same layer. Therefore, the execution of one operation cannot influence the attributes of other objects. However, in the Layer-doc model, the operation's execution of the parent node object can also influence the status of its child nodes. In Fig. 13, O_1, O_2, O_3 are concurrent with one another (that's $O_1 \| O_2 \| O_3$). Since they are not targeting at the same object, according to definition 3, they do not conflict with each other and can be executed in any order. However, as you can see from Fig. 13, different execution orders lead to different execution results. The Move operation targets at E<1,2,1,1> also changes the position attribute of V<1,2,1,1,2>. Therefore, different execution orders of operation O_2 and O_3 result in different execution results (see Fig. 13). Thus, new conflict definition must be given in the Layer-doc model.



(a) The initial document status



(b) The initial layered document model



O_1= Update(Pos(1,2,1,1),Color, Red, Blue);
O_2= Update(Pos(1,2,1,1), Move, (-10,0,0));
O_3=Update(Pos((1,2,1,1,1), Position, (140,44,0),(120,59,0));

(c) Execute Orders of different operations at three sites

Figure 13: Update conflict of the Layer-Doc Model

Definition 4: Conflict Operation of the Layer-Doc

Two Update operations U_a and U_b conflict with each other (indicates as $U_a \otimes U_b$), iff: (1) U_a is concurrent with U_b (i.e $U_a \parallel U_b$); (2) one of the following condition occurs: (a) U_a and U_b are targeting the same object's same attribute (i.e U_a.P=U_b.P and U_a.key=U_b.key); (b) one of the operations U_a and U_b (without generalization, we take U_a as the example) is targeting at an object which is the parent Node Element of another operation (i.e U_a.P locates at the parent node of U_b.P) and the semantic meaning of U_a.key covers that of U_b.key(for example, the Move operation and Update(Position) operation). As for the two conditions, we still use $U_a \otimes U_b$ to indicate the first one, while use $U_a \vec{\otimes} U_b$ to indicate the second one.

The paper introduces to use *Child-precedence* strategy to solve the conflict of the second condition. If two Update operations U_a and U_b satisfy that $U_a \vec{\otimes} U_b$, we define that the priority of U_b is always higher than U_a. Since U_b needs to be executed in the

context where U_a hasn't been executed, $\overline{U_a}$ must be executed firstly to exclude the execution effect of U_a, and then U_b and U_a are executed sequentially. The Conflict resolution process is described in procedure 3.

Procedure 3 Execute(Layer-Doc, level, SV, U),
Execute Update operation U with layer level and status SV.

```
//Suppose U.pos={p1,p2,...,plevel-1, plevel }
// Retraceing_LayerDoc procedure is called firstly to retrace the document status to
the moment where U is generated, so that the execution effects on the position of all
concurrent Insert and Delete operations are hidden.
1: Retracing(Layer-Doc, level, SV, U);
//Judge whether there are operations targeting at the parent Nodes Elements of U
that are conflict with it. If there are, a temp structure is defined to save these
operations;
2: OperationNode *tempUpdate[100];
3: int counter=0;
4: for (int i=1;i<level;i++) {
5:     Node-Element ele=Layer-Doc[U.pos[i]];
6:     for each operation Up of ele {
7:       if (Up ⊗ U) {
8:         if (Up.element<>tempUpdate[counter].element) // indicate that there are
many conflict operations of U. So that, the latest Update operation should be
replaced
9:           counter++;
10:          tempUpdate[counter]=&Up;
11:       }
12:     }
13: }
14: if (counter==0) {  //if there are no conflict operations of parent Node
15:   Execute(Layer-Doc, U);
16:  } else
     {  //Undo all the conflict operations from top to down of the Layered-doc
17:    for (int j=1; j<=counter; j++) {
18:      Execute(Layer-Doc, tempUpdate[j]); // Execute the reverse operation of
tempUpdate[j];
19:    }
20:  Execute(Layer-Doc, U);
21:  for (j=counter; j>=1; j++) {  //Execute Update operations from top to down
22:    Execute(Layer-Doc, tempUpdate[j]);
23:  }
24;  }
```

We take the scenario described in Fig. 14 as the example to describe the conflict resolution process. In Fig. 14, there are four Update operations: U_1, U_2, U_3 and U_4, which satisfy that $U_1 \rightarrow U_3$, $U_2 \| U_3$, $U_1 \vec{\otimes} U_4$, $U_3 \vec{\otimes} U_4$ and $U_4 \vec{\otimes} U_5$. Then the Execute () process is described as follows:

1) tempUpdate construction process：

(1) i=1:

Variable ele points to E_1, and all the operations of E_1 will be compared with U_5 to judge the conflict relationship;

Since $U_1 \vec{\otimes} U_5$, tempUpdate[1]=&U_1;

Since $U_3 \vec{\otimes} U_5$, and U_3 and tempUpdate[1] points to the operation of the same Node element, so that tempUpdate[1] is replaced with &U_3;

(2)i=2:

Variable ele points to E_2, and all operations of E_2 will be compared with U_5 to judge the conflict relationships;

Since $U_4 \vec{\otimes} U_5$, tempUpdate[2]=&U_4;

2) Undo process of tempUpdate:

(1) j=1: Execute ($\overline{U_3}$);

(2) j=2: Execute ($\overline{U_4}$);

3) Execute U_5;

4) Re-execute tempUpdate:

(1) j=2: Execute (U_4);

(2) j=1: Execute(U_3);

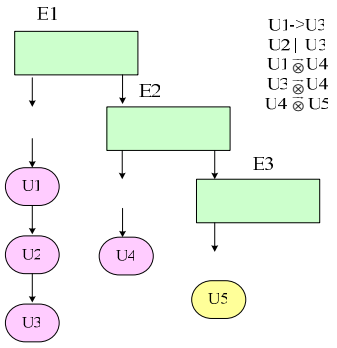


Figure 14: Scenario of conflict Update operations

During the construction process of tempUpdate, only one element of tempUpdate can point to one operation of every Node Element. If multiple operations satisfy the conflict condition, only the last operation can be appended to the tempUpdate list, since the execution effects of other Update operations with the same key have already been reflected within the last operation.

6.3 Capture of interface operations

TA technology depends on the API interface of the application to capture the local operations and complete execution of remote operations. AutoCAD provides twelve types of reactors to capture message: AcApDocManagerReactor, AcApLongTransactionReactor, AcDbDatabaseReactor, AcDbEntityReactor, AcDbLayoutManagerReactor, AcDbObjectReactor, AcDbSummaryInfoReactor, AcEdInputContextReactor, AcEditorReactor, AcRxDLinkerReactor,

AcRxEventReactor, AcTransactionReactor. Different reactors are responsible for listening to different states of AutoCAD. For example, AcDbDatabaseReactor is used to listen to the changes of the entities in the database, including objectAppended, objectModified, objectErased; AcEditorReactor is used to listen to the interface operations, including commandWillStart, commandEnded, commandFailed, commandCancelled, etc. TA technology mainly uses these two reactors.

At the beginning of the collaboration session, every application instance will be attached with a command reactor. After the reactor captures the *commandWillStart* message, a database reactor will be invoked so as to get the detail information about the operation, such as the operation type (Insert, Delete, Update), operation parameters (entity types, values), etc. If an operation is captured as an Update operation, all the entities' information of the layer of current BlockTable is copied in order to get the old_val parameter of the Update operation (see Fig. 15).

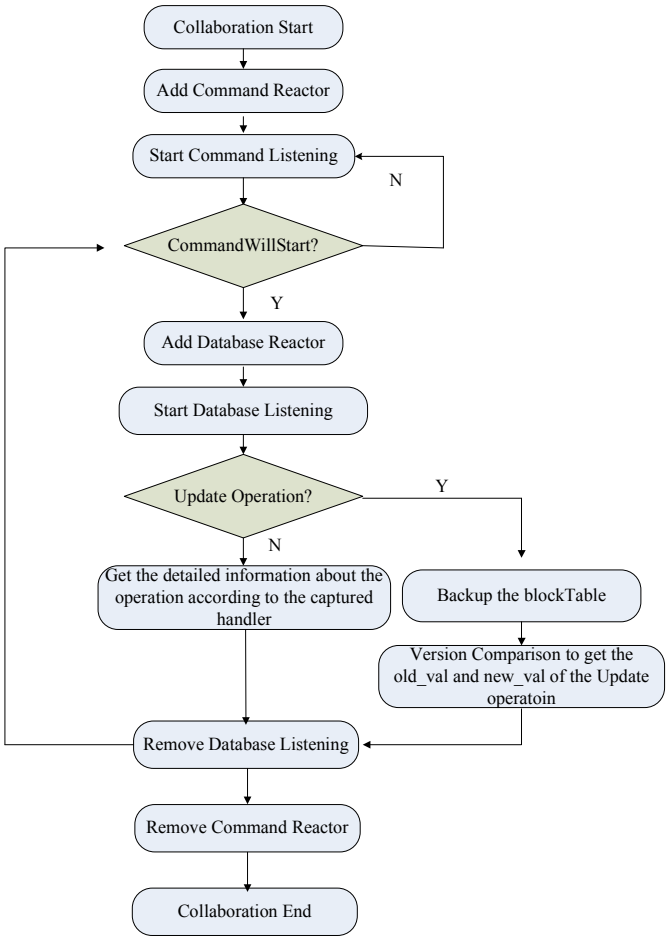


Figure 15: Control flow of database listening process

The database reactor is responsible for listening to the message types (*objectAppended*, *objectModified* and *objectErased*). If the message is *objectAppended* or *objectErased*, the entity types information got from *commandWillStart* message is achieved and we make use of AcDbObject pointer pObj to get the attributes values of the inserted entities (Take LINE as the example, startPoint, endpoint and Color attributes will be got); If the message is *objectModified,* the old_ val of the Update operation will be got from the block copy, while the new_val of it will be got from the current block.

7 The Co-AutoCAD system

The Co-AutoCAD system is designed based on the TA technology and its corresponding key aspects discussed in Section4, 5 and 6. The system is composed of two components: CoAutoCADServer and CoAutoCADClient(see Fig. 16), where the former is responsible for document storage, session management (including session startup, document upload and download etc.) and the latter responsible for the coordination between the CoAutoCADServer and the client applications, the control of the document upload and download process, startup of the client applications and load of CoDesign Kernel module etc. During the editing session, the CoAutoCADClient manages the collaboration process through its sub-module CoDesign Kernel, which completes the following functions: interception, decomposition, execution and broadcast of user interface messages as well as reception, transformation and execution of remote operations.

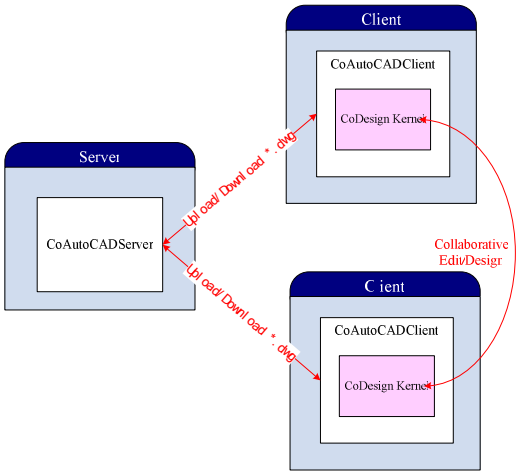


Figure 16: The system architecture

7.1 Interaction between CoAutoCADServer and CoAutoCADClient

The CoAutoCADServer manages all the design documents and provides the file upload and download services for CoAutoCADClient. It is also responsible for the

coordination of each client since only when all the participants are ready for the collaboration can the CoAutoCADServer start up the edit session.

As can be seen from Fig. 17, user name and password are first required to be transmitted to the server and only the valid user can enter the waiting list. Before the collaboration begins, collaboration participants can query the file list of editable documents from the server and download one of them. The server traces the file download requests and constructs the participants' list according to these requests (all the users downloading the same document are classified as one work team). Every participant can query the information of other ones by interacting with the Server (*getclientlist*). After one participant assumes that all the other participants are ready, it releases *negotiate* request to the server. And after the server receives this message, it denies the download requests from any client, so that new users are not allowed to enter into the collaboration session. The Server delays the response of the negotiate request (*clientinfolist*) until it receives all the *negotiate* requests of all participants. After all participants receive the *clientinfolist* response, they construct the P2P network of participants, load the CoDesign Kernels and switch on the collaborative design process. After the design session ends up, participants send *endedit* message to the Server, which responds with *commit* message to the last participant notifying it to upload its local document.

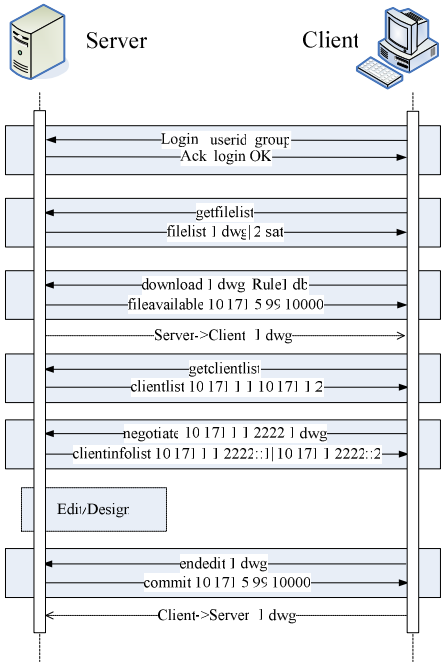


Figure 17: Interactions between the Server and the Client

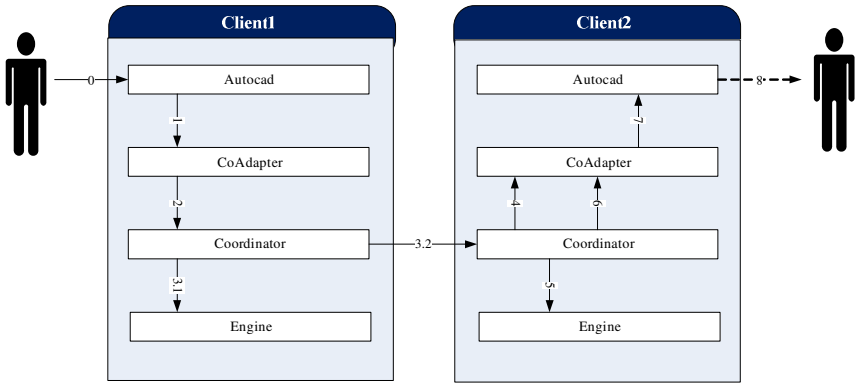
7.2 The CoDesign Kernel

The CoDesign Kernel is the core module of the Co-AutoCAD system, which is

responsible for managing the collaborative design process. It's composed of three layers: CoAdapter, Coordinator and Engine layers.

The CoAdapter layer is responsible for intercepting the user interface messages by using COM/ObjectARX interface of AutoCAD application, decomposing these messages into primitive ones, mapping their address formats into those that can be used by the Layer-AST algorithm, and completing the actual execution of remote operations on the local shared document.

The Coordinator layer is devised to append the timestamps to all the transformed operations and broadcast them to all the other participants, to receive remote operations from the network, to forward these operations to the Engine layer so that Layer-AST algorithm can be called to retrace the document to the state where these operations are released, and to transform them into those that can be executed by calling the COM/ObjectARX interface of AutoCAD. Figure 18 gives a brief description of the interaction process of the three layers.



Interaction process
0 User A releases design/edit command X, for example *Line XXX*
1 The CoAdapter layer intercepts the document modification message, such as *ObjectModified* message
2 The CoAdapter layer transform the document modification message to primitive operations (*UPDATE/CREATE/DELETE*) and transfers them to the Coordinator layer
3 The Coordinator layer performs the following operations
3.1 Notify the Engine layer to execute and handle these primitive operations
3.2 Broad these primitive operations to other collaborative participants
4 The Coordinator layer locks the CoAdapter layer to prevent new operations releases
5 The Coordinator layer transforms the remote operations into those that can be executed in current document status by calling corresponding interfaces of the Engine layer
6 The Coordinator layer notifies the CoAdapter layer to execute the transformed operations
7 The CoAdapter layer transforms the primitive operations into those that can be executed on the local document and calls the APIs of AutoCAD to complete the operations actual execution
8 User B sees the execution effect of operations released from user A

Figure 18: The architecture of the CoDesign Kernel

7.2.1 The CoAdapter layer

When the CoAdapter layer intercepts the user interface messages, it transforms them into primitive ones and transfers the transformed ones to the Coordinator layer, which broadcasts them to the other participants through the network. As a sequence, the

Coordinator layer transfers those operations to the Engine layer so that these operations can be executed on the layered document model. Besides that, the CoAdapter layer is responsible for executing the remote operations transferred from the Coordinator layer, so that the execution effect of other participants' operations can be reflected on the local document state.

The transformation objects of the CoAdapter layer on the user interface operations include the following:

1) Command set. AutoCAD has a rich set of commands, including create a circle, create a line, change color, modify line and so on. The Engine layer, however, has relatively limited operation types. For example, the Create operation in the Engine layer is formatted as Create:<*ADDRESS*>;<*TYPE*> where parameter <*TYPE*> is used to indicate the object type what is to be created. In order to adapt the AutoCAD's command set to the Engine's operation set, map matrix must be constructed between them by the CoAdapter layer.

2) Address space. AutoCAD uses random space (a long integer or a string) to differentiate every entity. The modification of the entity is also based on this address space. The Engine layer, however, uses the position information of every entity in the layered document as the unique identifier. So that, the CoAdapter layer must complete the transformation between these two address spaces.

The external interfaces and their descriptions the CoAdapter layer provides are as follows:

Interface 1 bool Exec(const std::vector<tstring>& ops, tstring timestamp) Execute remote operations *ops*. After the Coordinator layer monitors remote operations *ops*, it forwards them to the Engine layer so that current document status can be retraced to that when these operations are released. And after that, the Coordinator layer calls this function to complete the actual execution on the local shared document.
Parameters： ***ops:*** the operations which the Coordinator layer has received and which the Engine layer has transformed against current document status. ***timestamp:*** the timestamp attached to the operations *ops*. **Return value：** Return value "True" indicates that it has been executed successfully on the local site.

Interface 2 bool Lock() Lock the user interface of AutoCAD so as to prevent new operations' releases When the Coordinator layer receives causally-ready operations from remote sites, before it forwards these operations to the Engine layer, it must call *Adapter.Lock* to lock the local user interface so as to guarantee the invariance of the operations' execution environment.
Parameters: Null **Return value:** Return value "True" indicates that the user interface has been locked successfully.

7.2.2 The Coordinator layer

The Coordinator layer is responsible for performing the following operations as for the primitive operations transmitted from the CoAdapter layer:

1) Call the corresponding function in the Engine layer to execute these operations on the layered document model and return the current timestamp.

2) Attach the current timestamp to these operations and broadcast them to other sites.

The external interface functions and their descriptions provided by the Coordinator layer are as follows:

Interface 3 tstring Broadcast(const std::vector< tstring >& ops) It is called after the CoAdapter layer transformed the user interface commands (local operation) into primitive ones. Primitive operations *ops* are executed in the Engine layer, and then the operations attached with current timestamp will be broadcasted to other participants.
Parameter: *ops*：Primitive operations to be broadcasted to remote sites. **Return value：** The current timestamp which indicates the document status when the user commands are released.

7.2.3 The Engine layer

The Engine layer is responsible for consistency maintenance and it is implemented with the Layer-AST strategy. The main functions of the Engine layer include the following:

1) Execute the operation on the layered document model, and return current timestamp.

2) Judge whether a remote operation can be executed or not.

3) As for remote causally-ready operation, control the retrace process of the document and perform corresponding operation transformation.

The external interface functions and their descriptions provided by the Engine layer are as follows:

Interface 4 std::basic_string<TCHAR> Exec(const std::vector< std::basic_string<TCHAR> >& ops) As for local operation, this interface is invoked in function Coordinator.Broadcast. Execute the operation *ops* on the layered document model and return the current timestamp.
Parameters: *ops*: Operations, which have been transferred by the Coordinator layer and which are to be executed in the Engine layer. **Return value:** The current timestamp which indicates the document status when the user commands are released.

Interface 5 bool Executable(std::basic_string<TCHAR> stamp) Before the Coordinator layer executes a remote operation, it calls this function to judge whether the operation is causally-ready. Judge whether a remote operation attached with timestamp *stamp* satisfied the execution condition. It realizes this function by comparing the stamp with current timestamp. If one and only one component of the former is larger by 1 than that of the latter and all the other components are little than those of the latter, such operation with this timestamp is thought to be ready for execution.
Parameters: *stamp*: Timestamp attached to the to-be-judged remote operation. **Return value:** Return value "True" indicates that the operation is ready for execution.

Interface 6 std::vector< std::basic_string<TCHAR> > Trans(const std::vector< std::basic_string<TCHAR> >& ops, std::basic_string<TCHAR> stamp) Before the Coordinator layer executes remote operations, the operations should first be transformed into the formats that can be executed in current document status. Transform remote operations according to its timestamp *stamp* against current document status so that the transformed operation can be executed in current document status.
Parameters: *ops*: remote operations which are to be transformed. *stamp*: the timestamp of remote operations *ops*. **Return value:** The transformed operation.

8 Conclusions and Future Work

The paper analyses the data model of the AutoCAD application and puts forwards to mapping it into the layered document model so as to reduce the number of the nodes to be retraced, thus improving the algorithm efficiency. Update conflict definition and resolution strategy in layered document model are also discussed. As for the issue of operation adaptation, the paper introduces using database listening technique and comparison of old and new document to capture the user interface operations and their corresponding parameters. Besides that, the paper gives detailed description about the Co-AutoCAD system architecture which uses the TA technology and the whole strategy discussed in this paper.

In the future, more research work will be done to tackle with the distributed consistency of three-dimensional objects of the CAD applications. New consistency model and algorithm will be discussed in order to be adapted to the new environment.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (NSFC) under Grant No.60803118 and No. 60736020, "Chen Guang" project supported by Shanghai Municipal Education Commission and Shanghai Education Development

Foundation under Grant No.10CG49, the 2010 Special Fund for Cultivate Research of Shanghai College to Select Outstanding Young Teacher under Grant No. SLG10007, the 2010 Special Project for Manufacturing Informatization of Shanghai "Science & Technology Innovation Action Plan" under Grant No. 10dz1125600 and Key Laboratory Funding of Science & Technology Commission of Shanghai Municipality under Grant No.10DZ2272600.

References

[Dewan and Sharma, 1999] Dewan, P., An experiment in interoperating heterogeneous collaborative systems. In: Proceedings of the Sixth European Conference of Computer Supported Cooperative Work (ECSCW'99), 1999: 371–391.

[Ellis and Gibbs, 1989] Ellis, C. A. and Gibbs S. J., Concurrency control in groupware systems. In: Proceedings of the 1989 ACM SIGMOD international conference, 1989: 399-407.

[Gao et al., 2008] Gao, L. Shao, B., Zhu, L. Lu, T. and Gu, N., Maintaining Time and Space Consistency in Hybrid Engineering Environments: Framework and Algorithms. In Journal of Computers in Industry, Volume. 59 (2008), Issue. 9 : 894–904.

[Gu et al., 2005] Gu, N. Yang, J., Zhang, Q., Consistency maintenance based on the mark & retrace technique in groupware systems. In: Proceedings of the international ACM SIGGROUP conference on Supporting group work, 2005: 264-273.

[Gutwin et al., 2008], Gutwin, C., Greenberg, S. Blum, R., Dyck, J., Tee, K. and McEwan, G., Supporting Informal Collaboration in Shared-Workspace Groupware. In Journal of Universal Computer Science, vol. 14, no. 9 (2008): 1411-1434.

[Kell, 2008] Kell, S., A Survey of Practical Software Adaptation Techniques, In Journal of Universal Computer Science, vol. 14, no. 13 (2008): 2110-2157.

[Herskovic et al., 2011] Herskovic, V., Ochoa, S., Pino, J.A., and Neyem, A., The Iceberg Effect: Behind the User Interface of Mobile Collaborative Systems, In Journal of Universal Computer Science, vol. 17, no. 2 (2011): 183-202.

[Knister and Prakash, 1990] Knister, M.J. and Prakash, A., DistEdit: A distributed toolkit for supporting multiple group editors, in: Proceedings of ACM Conference on Computer Supported Cooperative Work, 1990: 343–355.

[Krebs et al., 2003] Krebs, A.M., Ionescu, M. Dorohonceanu, B., Marsic, I., The DISCIPLE System for Collaboration over the Heterogeneous Web. In: Proceedings of the 36th Annual Hawaii International Conference on System Sciences, 2003: 11-21.

[LaMarca et al., 1999] LaMarca, A., Edwards, W.K., Dourish, P., Lamping, J.,Smith, I., Thornton, J., Taking the work out of workflow: Mechanisms for document-centered collaboration. In: Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work (ECSCW'99), 1999: 1–20.

[Li and Li, 2004] Li, D., Li, R., Preserving operation effects relation in group editors. In: Proceedings of the ACM Conference on Computer Supported Cooperative Work (CSCW'04), 2004: 457–466.

[Li and Li, 2002] Li, D., Li, R., Transparent sharing and interoperation of heterogeneous single-user applications. In: Proceedings of the ACM Conference on Computer-Supported Cooperative Work, 2002: 246-25.

[Li and Lu] Li, D., Lu, J., A Lightweight Approach to Transparent Sharing of Familiar Single-User Editors. In: Proceedings of ACM Conference on Computer-Supported Cooperative Work, 2006: 139–148.

[Lin et al., 2007] Lin, K., Chen, D. Xia, S., Sun, C., API design recommendations for facilitating conversion of single-user applications into collaborative applications. In: International Conference on Collaborative Computing: Networking, Applications and Worksharing, 2007: 309-317.

[Palfreyman et al., 1999] Palfreyman, K., Rodden, T., Trevor, J., PSI: A platform for shared interaction. In: Proceedings of the Sixth European Conference of Computer Supported Cooperative Work (ECSCW'99), 1999: 351–370.

[Shen et al., 2007] Shen, H., Sun, C., and Zhou, S., Leveraging Single-user OpenOffice Writer for Collaboration by Transparent Adaptation. In: the 8th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, 2007: 15-20.

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

[Sun and Sun, 2006] Sun, D., Sun, C., Operation context and context-based operational transformation. In: Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work, Banff, 2006: 279–288.

[Sun et al., 2006] Sun, C., Xia, S., Sun, D., Chen, D., Shen, H. and Cai, W., Transparent adaptation of single-user applications for multi-user real-time collaboration. In ACM Transactions on Computer-Human Interaction(TOCHI), 2006, 13(4): 531-582.

[Xia et al., 2004] Xia, S., Sun, D., Sun, C., Chen, D. and Shen, H., Leveraging Single-User Applications for Multi-User Collaboration: the CoWord approach. In: Proceedings of the 2004 ACM conference on Computer supported cooperative work, 2004: 162-171.

[Yang et al., 2005] Yang, J., Zhang, Q., Gu, N., Yang, G., Liu, Z., The multi-version and single-display strategy in undo scheme. In: The Fifth International Conference on Computer and Information Technology, 2005: 290–296.

[Zheng et al., 2009] Zheng, Y., Shen, H., Sun, C., Leveraging single-user AutoCAD for collaboration by transparent adaptation. In: Proceedings of International Conference on Computer Supported Cooperative Work in Design, 2009: 78-83