Concurrent Video: Operational Extensions[1]

Oleg Proskurnin

University of St.Petersburg, Russia
olegpro@acm.org

Abstract. Today, non-linear video editing technology became ubiquitous, having spread from the film industry to almost every home computer. Dozens of multimedia authoring applications offer a wide range of functional capabilities, yet there is an evident lack of support for collaborative activities among them. To change this situation, a novel data model called concurrent video has been recently proposed. Since it was substantially based on formal aspects of cooperative transactions, it has provided a solid foundation for consistent exchange and sharing of information between co-workers in collaborative environments. The main objective of this work is to extend the concurrent video model by introducing advanced editing operations, which should permit more flexibility within the authoring process and enable a higher level of concurrency among users' actions. Besides, this extension is going to be performed in such a way that major transactional properties of the original proposal will be preserved.

Keywords. Concurrent video, video modeling, collaborative multimedia authoring, CSCW, cooperative transactions, consistency, CoAct

1. Introduction

It is a well known fact that the possibility to collaborate in multimedia authoring environments is going to bring considerable advantages into the process of content creation. With reasons ranging from tight project deadlines to distribution of tasks among editors, support for joint activities could noticeably improve video post production technology in all related industries [15].

Recently, a technique for collaborative authoring of media streams, such as video or audio, has been presented in the form of a concurrent video data model [11]. Based on the cooperative activity framework CoAct [13, 18, 9], this model has enabled consistent sharing of information in the face of concurrent modifications, which is often regarded as the key task of multi-user applications design.

The background is that co-authors' actions in the CoAct framework are represented via activity histories, which are basically sequences of editing operations constituting individual working processes. As a result, consistency of information exchange is achieved by means of the history merging mechanism [18] that takes into account compatibility relations of participants' actions in order to detect conflicts in their joint efforts.

[1] This work is partially supported by the Russian Foundation for Basic Research under grant No. 04-01-00173

To avoid unnecessary conflicts, editing operations offered within the concurrent video model were especially designed to have good commutativity properties. In fact, according to the concepts of timeline-based video authoring [4], these operations allowed participants to work concurrently on different clips, which in the end formed the desired video material.

The goal of this paper is to improve the original data model by providing support for non-conflicting cooperative activities within the same clips. However, most transactional aspects of the previous proposal are going to be kept intact.

The rest of this paper is organized as follows. After giving a brief overview of the related work, the improved version of the data model is introduced. Next, transactional properties of concurrent video are discussed and finally conclusions and further work are presented.

2. Related Work

This section first gives a brief review of the CoAct framework and then presents a summary of previous work on video modeling and collaborative multimedia authoring.

2.1. The Cooperative Activity Model

The cooperative activity framework CoAct [13] defines a generic transactional model that enables consistent sharing and exchange of information in various collaborative scenarios, like multi-user document authoring environments [14].

To support joint activities, the CoAct framework assigns a personal isolated workspace containing copies of shared data to each user involved in the cooperative effort. Also, a separate common workspace is established to represent the current state of team work. Thus, participants are able to manipulate their private versions of shared objects independently from each other, while the common database stores at first initial data, then intermediate and lastly final results of the joint effort.

At the user level, cooperation is achieved by explicit exchange of the contents of workspaces, which can be performed either directly or through the common database by means of dedicated operations. But conceptually, these processes are controlled by the history merging mechanism [18, 9], that takes into consideration semantics of the application domain, ensuring correctness of information exchange.

The point is that participant's actions in CoAct are modeled by an activity history, which is basically a sequence of operations carried out by a certain user in his workspace. Similar to conventional concurrency control [20], a semantic conflict test is defined for each possible pair of operation invocations. Such compatibility predicate is often symmetric and is typically specified in terms of state-independent commutativity relations. Since the order in which commuting operations are executed is irrelevant, the compatibility property is used in CoAct as a basis for merging activity histories. In other words, two operation invocations coming from different users are both allowed to be present in the merge only if they are compatible. Otherwise, the controlling user has to resolve the conflict by discarding unnecessary actions.

Consequently, the merging mechanism of the CoAct model enables semantically consistent incorporation of individual activity histories into a single one that is implicitly formed in the common database and represents the results of cooperative work.

2.2. Collaborative Video Authoring

Although a variety of multimedia authoring tools are available in the market, e.g. [2, 16], none of them supports concurrent video editing and the related research proposals are also very few.

Some of these works investigate cooperative activities based on locking mechanisms [3, 17], while other approaches consider collaboration in real-time groupware systems only [22]. In contrast to the concurrent video model [11], these proposals do not cover multimedia modeling issues and do not provide consistency guarantees in a transactional sense.

However, there exists a wealth of research works examining various aspects of video data management and some of them have common features with the presented investigations.

In particular, operations on media segments defined in video and stream algebra [21, 10], as well as updating mechanisms in other proposals [1, 6] partially resemble editing facilities of the model described below. Moreover, previously introduced hierarchical data structures [1, 21] and video objects [12] also have certain correlations with the presented approach.

Nevertheless, the major distinction between concurrent video and former models lies in the suitability of the new proposal for the basic needs of both video authoring systems and cooperative environments, which has not been achieved ever before.

The point is that in the considered authoring systems the movie is created by altering and assembling source clips which are treated as independent units of raw material. These clips along with other media objects are placed on the timeline, which naturally represents the time flow. After the inclusion, objects can be edited, special effects and transitions can be applied and the resulting material can be arranged into a final video production. In addition, since such editing implies just referencing the original files, which are not actually modified, a special process called rendering is required to preview or export the produced movie.

Concurrent video takes into account these basic concepts of timeline-based authoring and provides an efficient tree-based abstraction for referencing a sequence of media clips. It also presents a set of high-level non-destructive editing operations allowing users to insert and delete, alter and temporally combine video material to form the final production.

To support joint activities, operations on different clips are considered to be commutative, i.e. different users can work concurrently on different clips without any conflicts. Moreover, hierarchical structure that references video data also stores activity histories, providing elegant support for cooperation mechanisms of CoAct.

The above features clearly distinguish concurrent video model [11] from other research proposals in the area of cooperative authoring. In particular, this investigation significantly differs from recent works [8] and [5], which present tree-based data structures for synchronous collaborative document editing.

3. Enhanced Concurrent Video Model

This section outlines improved concurrent video model that provides advanced intra-clip editing operations and is entirely based on its already presented counterpart [11].

3.1. Video Segments

In the considered model video segments represent an abstraction over independent units of video data, which are used as building blocks within the authoring process.

Basically, each video segment references a contiguous part of raw material via a frame sequence and has its own set of attribute-value pairs which describe the proper interpretation of the underlying media at the presentation or rendering level. Frame sequences reflect the stream-like nature of video data, while attributes support implementation of non-destructive editing operations as well as specifications of desired transitions and special effects for the segment.

Actually, both the definition and the purpose of video segments in this work stay the same as in the original proposal [11], however, we quote some facts here for convenience:

Definition 1 (frame sequence) A frame sequence is a finite non-empty sequence $(f_1, \ldots, f_N)$ of video frames f_i, referring to a contiguous block of video data. N is called the length of a video sequence F and is denoted by $Length(F)$.

Definition 2 (video segment) A video segment is a pair (F, A), where F is a frame sequence and A is a possibly empty set of attribute-value pairs $\{a_i:v_i\}$, storing various additional information about the containing segment.

Besides, two trivial operations – concatenation and extraction – can be defined for frame sequences [11], however they won't be referred in this work explicitly and thus are omitted.

3.2. Video Activity Tree

Concurrent video employs a single hierarchical data structure, called video activity tree, for modeling both underlying media and cooperative transactions.

Basically, the structure of an activity tree in this work remains the same as before: leaf nodes, called valid, are associated with video clips that authors place on a timeline, dead nodes stand for previously deleted clips and the rest nodes hold related parts of the activity history. Also, activity history elements are marked whether they are private or shared to indicate what part of the tree is present in the common database and what part exists only in the considered local workspace, as illustrated in figure 1.

However, unique identifiers assigned to video clips, which have enabled the development of editing operations with state-independent commutativity relations in [11], are not sufficient for the purposes of this work. In order to provide intra-clip operations with similar commutativity properties, unique values should be assigned to all frames constituting the video stream. Fortunately, such task can be performed

by combining clips' identifiers with frames' indices within these clips. This approach is actively exploited by novel editing operations and is reflected below in the redefinitions of activity tree's node structures.

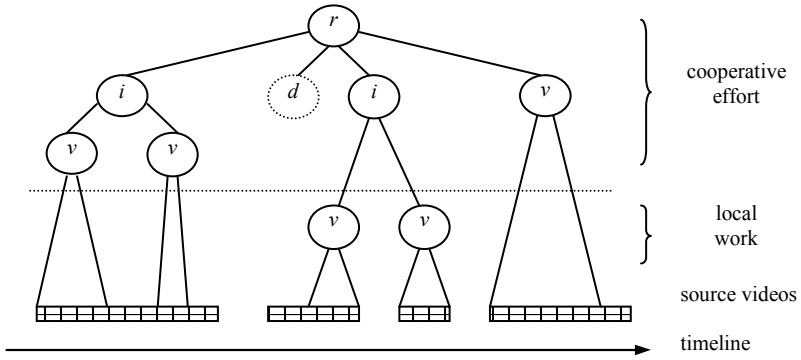

Figure 1. A sample video activity tree in a user's workspace. Characters *v*, *d*, *i* and *r* denote valid, dead, intermediate nodes and the root node of the tree respectively.

Definition 3 (video activity tree) A video activity tree T is a tuple (*Root*, *Nodes*, *Valid*, *Dead*), where *Root* is the root node of T, *Nodes* is a set of intermediate nodes of T, and *Valid* and *Dead* are disjoint sets of so-called valid and dead leaf nodes respectively.

Additionally, there exists a total order $<_t$ defined over the set of leaf nodes of T, that corresponds to the order in which associated video clips are (or were for dead nodes) located on the timeline.

Definition 4 (valid node) A valid node V of a video activity tree is a tuple (*NID*, *Range, Segment*, *History*), where *NID-Range* pair uniquely identifies V's associated video data, *Segment* is the video segment representing this data and *History* is an ordered set of operation instances related to the given node V. *Range* is represented as a pair of bounding values [*Low*, *High*].

Definition 5 (intermediate node and dead node) An intermediate node as well as a dead node of a video activity tree is merely a tuple (*NID*, *Range, History*), whose elements are intended for the same purposes as their counterparts in valid nodes (except *NID-Range* pair identifies no data).

Operation instances, mentioned above, can be treated as records describing the fact of execution of a certain editing operation by some user. They act as elementary entities for building activity histories and modeling the authoring process, being the same as in the original work [11]:

Definition 6 (operation instance) An operation instance is a tuple (*Status*, *OID*, *Name*, *Input*, *Output*), where $Status \in \{private, shared\}$ indicates whether this instance is present only in the current workspace or not, *OID* is a unique identifier of this instance (required for tracing identical instances during the merging process) and *Name* is the name of the executed operation, whose input and output parameters are reflected in *Input* and *Output* sets.

3.3. Editing Operations

The main challenge in the design of editing operations is to provide a solid support for modeling authors' actions in a collaborative environment. Thus, it is worth to remember that the most common thing cooperating authors are likely to do is altering and assembling video clips on a timeline, and what is more, they are likely to do it concurrently.

In contrast with the original proposal [11], which has provided only segment-level editing operations, in this work most attention is focused on frame-level manipulations and achieving commutativity of semantically non-conflicting actions performed within the same video segment, e.g. altering of different frames of the same clip by different users.

First of all, an initialization routine intended for fixing an initial state of the cooperative activity in the common database is provided.

Definition 7 (initialization algorithm) The initialization algorithm takes a sequence of N video clips, denoted by a collection of N video segments (vs_1, …, vs_N) as input and forms a corresponding video activity tree with an empty history:

1. Create N valid nodes V_i: $\forall\ i,\ 1 \le i \le N$: $V_i := (NewID(), [1, Length(vs_i.F)], vs_i, \varnothing)$,
   where $NewID()$ is a function generating a new unique identifier,
   see [11] for how it may work.
2. Construct a video activity tree T: $T := (Root, \varnothing, \{V_i \mid 1 \le i \le N\}, \varnothing)$, such that:
   $\forall\ V_i$: $Parent(V_i) = Root \wedge \forall\ i, j$: $1 \le i < j \le N \Leftrightarrow V_i <_t V_j$.

Next, an operation for editing video clips as a whole or for modifying their parts is presented:

Definition 8 (editing algorithm) The editing algorithm takes as input an identifier id of the clip and a range r of frames within it, which have to be replaced with a new video segment vs representing the results of performed modifications. The algorithm splits the affected node of the video activity tree T (if necessary) and accordingly updates its activity history:

1. Find a node $V \in T.Valid$: $V.NID = id \wedge r \subset V.Range$,
   Report failure if such node does not exist.
2. *If* $r = V.Range$, *Then* modify V:
   $V := (V.NID, [1, Length(vs.F)], vs, \{O \in V.History \mid O.Status = Shared\})$,
   Append an instance ($private$, $NewID()$, $Edit$, $\{id, r, vs\}$, $\varnothing$) to $V.History$,
   Report success.
   Else construct an intermediate node $V' := (V.NID, V.Range, V.History)$,
   And go to step 3.
3. Construct valid nodes V_i (at least two of them with non-empty frame sequences):
   $V_1 = (V.NID, [V.Range.Low, r.Low - 1], LeftSegment, \varnothing)$,
   $V_2 = (NewID(), [1, Length(vs.F)], vs, \varnothing)$,
   $V_3 = (V.NID, [r.High + 1, V.Range.High], RightSegment, \varnothing)$,
   where the contents of video segments $LeftSegment$ and $RightSegment$ is illustrated in figure 2.
4. Adjust T: $T := (Root, T.Nodes \cup \{V'\}, T.Valid \cup \{V_i\} \setminus \{V\}, T.Dead)$, such that:
   $Parent(V') = Parent(V) \wedge \forall\ V_i$: $Parent(V_i) = V' \wedge \forall\ i, j$: $i < j \Leftrightarrow V_i <_t V_j$.
5. Append instance ($private$, $NewID()$, $Edit$, $\{id, r, vs\}$, $\varnothing$) to $V_2.History$.

Evidently, the above algorithm provides an efficient means of modeling frame level manipulations within a particular video clip and at the same time supports operations applied to the whole segment, as already proposed in [11]. Since intra-segment editing is based on the node splitting technique, the affected valid node is replaced with a subtree, whose leaves are arranged to correspond to the resulting video material, as illustrated in figure 2. Moreover, propagation of the original node identifier to the node's respective sub-segments formed after splitting enables concurrent intra-segment manipulations in non-overlapping clip regions.

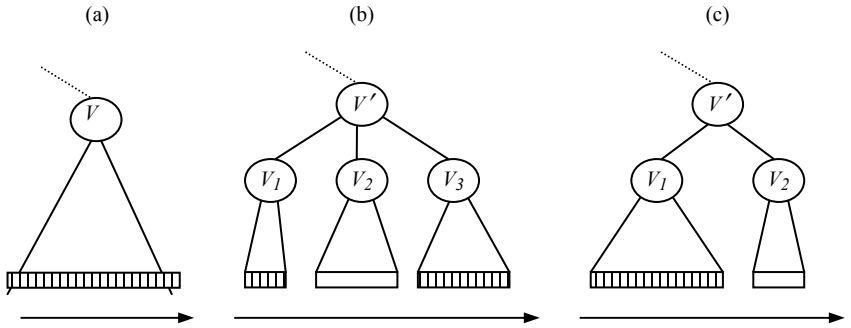

Figure 2. Node splitting carried out during intra-clip editing: (a) – an original valid node with the associated clip, (b) – split node after editing inside of the clip, (c) – split node after right-edge editing.

At this point it is worth to mention that editing algorithms make no modifications in the source video material – only references to the raw data are actually manipulated. Implicitly, such approach assumes the possibility of random access to the frames contained in the underlying video clips. However, this may be unfeasible (at least directly) when clips are coded with compression algorithms exploiting motion estimation, i.e. dependencies between adjacent frames. To overcome this difficulty it can be assumed that video material is accessed from an appropriate starting point, such as the nearest previous key frame, if necessary.

Next, operations for including new material into the video production and removing unnecessary parts from it are described. The insertion algorithm is left almost unchanged [11], while the deletion method now supports intra-clip removals.

Definition 9 (insertion algorithm) This algorithm takes as input a new video segment *vs* and its desired location on the timeline, specified by a destination point *pos* within a clip having given identifier *id*. The affected node is split and the activity history is accordingly updated:

1. Find a node $V \in T.Valid$:
   $V.NID = id \wedge (pos \in V.Range \vee (pos = 0 \wedge V.Range.Low = 1))$,
   Report failure if such node does not exist.
2. Construct valid nodes V_i (at least two of them with non-empty frame sequences):
   $V_1 = (V.NID, [V.Range.Low, pos], LeftSegment, \varnothing)$,
   $V_2 = (NewID(), [1, Length(vs.F)], vs, \varnothing)$,
   $V_3 = (V.NID, [pos + 1, V.Range.High], RightSegment, \varnothing)$,
   and split the node V in a way similar to splitting in the editing algorithm.
3. Append instance $(private, NewID(), Insert, \{vs, id, pos\}, \varnothing)$ to $V_2.History$.

Note, that in contrast to [11], insertion operation instance is stored in the newly created node and not in the split one. Along with propagating of the original segment identifier to its remainders, this enables commutativity of insertions occurred at different positions in the same clip.

Definition 10 (deletion algorithm) This algorithm takes as input an identifier *id* of the clip and a range *r* of frames within it, which need to be removed. The algorithm splits the affected node (if necessary), forms a new dead node and accordingly updates its activity history:

1. Find a node $V \in T.Valid$: $V.NID = id \wedge r \subset V.Range$,
   Report failure if such node does not exist;
   Coordinate with previous deletions and undo local insertions, like in [11].
2. Construct valid nodes V_i (at least two of them with non-empty frame sequences):
   $V_1 = (V.NID, [V.Range.Low, r.Low - 1], LeftSegment, \varnothing)$,
   $V_2 = (V.NID, r, \varnothing)$, this should be actually a dead node,
   $V_3 = (V.NID, [r.High + 1, V.Range.High], RightSegment, \varnothing)$,
   and split *V* in case its part is removed, otherwise *V* becomes a dead node itself.
3. Append an instance (*private*, *NewID*(), *Delete*, {*id*, *r*}, ∅) to $V_2.History$.

Similar to [11], the deletion algorithm can be designed to support commutativity between any two removals, even overlapping within the same clip. Also, it can act as an inverse for some local insertions, which are not shared by other users and are not affected by subsequent operations. The latter can be seen as an example of history reduction at the user's side.

In addition to the presented editing operations it turns out to be useful to provide a dedicated moving algorithm, which would enable moving of selected contiguous blocks of the media stream to the specified locations. Though at first sight it may seem that such method is unnecessary because it looks the same as an appropriate sequence of deletion and insertion, in fact it is required since delete-insert pairs are in conflict with concurrent editing of the moved block. Moreover, moving and editing activities over a single segment can be regarded as compatible from a semantic point of view and thus the following algorithm has to be provided.

Definition 11 (moving algorithm) This algorithm takes as input an identifier *mid* of the clip and a range *r* of frames within it, which has to be moved. Their target location is specified by *pos* and *id* parameters, which are similar to their counterparts in the insertion algorithm. The algorithm is especially designed in a way to ensure compatibility of moving operation with concurrent manipulations within the moved segment, such as editing or insertion activities. To achieve this, it is insufficient to move a single valid node like in [11], but it is necessary to move a proper valid (if it exists) or the nearest intermediate node with a certain part of its subtree, which stands for concurrent manipulations mentioned above (if any). Throughout this process, the identifier of the moved clip should be kept intact, as outlined below:

1. Find a node $M \in T.Valid \cup T.Nodes$: $M.NID = mid \wedge r \subset M.Range$; or fail.
2. Apply steps 2, 3 of deletion algorithm to *M*, using *r* and *Move* operation instance.
3. Apply insertion algorithm with *M.Segment* (if *M* is valid node), *r*, *id*, and *pos* as input, such that $V_2 = (mid, r, M.Segment, \varnothing)$, and use *Move* operation instance.
4. Re-execute operation instances which were located in the nodes below *M* (if any); This step is required only when *M* turns out to be an intermediate node.

4. Achieving Cooperation

According to the principles of the CoAct transaction model, two distinct types of commutativity, namely forward and backward, are utilized for manipulating activity histories [9, 19].

In particular, backward commutativity is intended for determining dependencies between user's actions within a single history. The point is that the behavior of editing operations may be influenced by previously executed methods, for example, any modifications of a particular video clip depend on the preceding insertion of this clip into the media stream. Consequently, no operation instance can be exchanged between any workspaces without its relevant predecessors. And backward commutativity allows us to identify closed subhistories [9] having no external dependencies and thus representing consistent units of work, which can be exchanged between cooperating users separately from each other.

Actually, such dependencies are naturally reflected by the hierarchical structure of the original as well as the extended concurrent video model – dependent operation instances can be found "above" the given one within the tree. Hence, already developed algorithm for subhistory extraction [11], which is almost as simple as subtree selection, remains valid for the given proposal, providing an efficient way for identifying consistent units of work.

After extraction of consistent units of work from users' workspaces, another type of commutativity – forward – is exploited for detecting conflicts between operations coming from different users, serving as a basis for semantically correct merging of the results of individual activities. Forward commutativity properties for the methods of the data model presented in this paper are summarized below.

Operations	*Edit*(*ID*, *R*, *VS*)	*Insert*(*VS*, *ID*, *POS*)	*Delete*(*ID*, *R*)	*Move*(*MID*, *R*, *ID*, *POS*)
Edit(*id*, *r*, *vs*)	$id \neq ID \vee r \cap R = \varnothing$			
Insert(*vs*, *id*, *pos*)	$id \neq ID \vee pos \notin R$ *	$id \neq ID \vee pos \neq POS$		
Delete(*id*, *r*)	$id \neq ID \vee r \cap R = \varnothing$	$id \neq ID \vee POS \notin r$ *	*true*	
Move(*mid*, *r*, *id*, *pos*)	$(mid \neq ID \vee r \supset R \vee r \cap R = \varnothing) \wedge (id \neq ID \vee pos \notin R$ *$)$	$(mid \neq ID \vee (r.Low - 1 \neq POS \wedge r.High \neq POS)) \wedge (id \neq ID \vee pos \neq POS)$	$(mid \neq ID \vee r \cap R = \varnothing) \wedge (id \neq ID \vee pos \notin R$ *$)$	$\{mid, id\} \cap \{MID, ID\} = \varnothing$

* Actually, it is also reasonable to treat these operations as conflicting when $pos + 1 \in r$ in order to completely avoid commutativity in boundary cases

Table 1. Forward commutativity relations (symmetric).

Relations above demonstrate that editing operations of the expanded concurrent video model provide visible concurrency benefits comparing to the original proposal. It is now clear that users can freely work on the same segments until frame regions they are manipulating do not overlap and what is more, video extracts moved along the timeline can be simultaneously modified by other participants.

5. Conclusions and Further Work

In this paper, the concurrent video data model has been extended by means of the development of intra-segment editing operations.

Such extension has enabled a higher level of concurrency among cooperating users, at the same time providing more accurate support for modeling their activities in collaborative video authoring environments. Additionally, major transactional aspects of the original proposal [11] were preserved in a way to ensure efficient information sharing and exchange.

Moreover, the results of this work are general enough to be applicable to stream data in general. For instance, by renaming of frame sequences and video segments to sample sequences and audio segments, the concurrent video model can be transformed into concurrent audio, which may be found useful in the similar authoring environments.

As a possible direction of further research, investigation of versioning support for stream data on the basis of this work is considered.

Besides, development of a prototype system demonstrating feasibility of the presented approach is regarded as an essential part of the future work.

For this purpose, there is an intention to utilize the recently developed Advanced Authoring Format [7, 15] and its open source SDK. Basically, the AAF file format is an industry-driven standard especially designed for interchange of compositional meta-data between various multimedia authoring tools. At the moment of writing this paper, several video editing applications have already claimed a certain level of compliance with AAF. In particular, Adobe Premiere Pro [2] is able to export its projects in this novel form.

Acknowledgements

Special thanks are expressed to Boris Novikov for encouragement and discussions on the topic.

References

[1]. Adali, S., Candan, K.S., Chen, S., Erol, K., Subrahmanian, V.: The advanced video information system: Data structures and query processing. Multimedia Systems, Vol.4, pages 172-186, 1996

[2]. Adobe, Adobe Premiere Pro 7.0: http://www.adobe.com/products/premiere, 2004

[3]. Borghoff, U.M., Teege, G.: Structure management in the collaborative multimedia editing system IRIS. In Proc. of the International Conf. on Multi-Media Modeling, pages 159-173, Singapore, 1993

[4]. Bulterman, D.C.A., Hardman, L.: Multimedia authoring tools: State of the art and research challenges. In Lecture Notes in Computer Science #1000, Springer-Verlag, 1995

[5]. Davis, A.H., Sun, C., Lu, J.: Generalizing operational transformation to the standard general markup language. In Proc. of the 2002 ACM conference on Computer Supported Cooperative Work, pages 58-67, New Orleans, USA, November 2002

[6]. Dumas, M., Lozano, R., Fauvet, M.-C., Martin, H., Scholl, P.-C.: A sequence-based object-oriented model for video databases. Multimedia Tools and Applications, Vol.18, Issue 3, pages 249-277, 2002
[7]. Gilmer, B.: AAF – The Advanced Authoring Format. http://www.aafassociation.org/html/pr/291_gilmer1_0702.pdf
[8]. Ionescu, M., Marsic, I.: Tree-Based Concurrency Control in Distributed Groupware. Computer Supported Cooperative Work, Vol.12, No.3, pages 329-350, 2003
[9]. Klingemann, J., Tesch, T., Wasch, J.: Semantics-based transaction management for cooperative applications. In Proc. of the International Workshop on Advanced Transaction Models and Architectures, pages 234-252, Goa, India, August - September 1996
[10]. Mackay, W.E., Beaudouin-Lafon, M.: DIVA: exploratory data analysis with multimedia streams. Conf. proc. on Human factors in computing systems, pages 416-423, Los Angeles, USA, April 1998
[11]. Novikov, B., Proskurnin, O.: Towards collaborative video authoring. In Proc. of the 7th East-European Conference on Advances in Databases and Information Systems, pages 370-384, Dresden, Germany, September 2003
[12]. Oomoto, E., Tanaka, K.: OVID: Design and implementation of a video-object database system. IEEE Transactions on Knowledge and Data Engineering, Vol.5, No.4, August 1993
[13]. Rusinkiewicz, M., Klas, W., Tesch, T., Wasch, J., Muth, P.: Towards a cooperative transaction model - The cooperative activity model. In Proc. of the 21st International Conference on Very Large Databases, pages 194-205, Zurich, Switzerland, September 1995
[14]. Tesch, T., Wasch, J.: Transaction support for cooperative hypermedia document authoring – a study on requirements. In Proc. of the 8th ERCIM Database Research Group Workshop on Database Issues and Infrastructure in Cooperative Information Systems (EDRG-8), pages 31-42, Trondheim, Norway, August 1995
[15]. Turner, B.: A new take on teamwork. Video Systems, December 2002. http://videosystems.com/ar/video_new_teamwork
[16]. Ulead, MediaStudio Pro 7.0: http://www.ulead.com/msp, 2004
[17]. Wang, K.: The design of extending individual multimedia authoring to cooperative multimedia authoring. In Proc. of the Fourth International Conference for Young Computer Scientists (ICYCS'95), Beijing, China, July 1995
[18]. Wasch, J., Klas, W.: History merging as a mechanism for concurrency control in cooperative environments. In Proc. of RIDE-Interoperability of Nontraditional Database Systems, pages 76-85, New Orleans, USA, February 1996
[19]. Weihl, W.E.: Commutativity-based concurrency control for abstract data types. IEEE Transactions on Computers, Vol.37, No.12, pages 1488-1505, December 1988
[20]. Weikum, G.: Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, Vol.16, No.1, pages 132-180, March 1991
[21]. Weiss, R., Duda, A., Gifford, D.: Composition and search with a video algebra. IEEE Multimedia, Vol.2, No.1, pages 12-25, 1995
[22]. Xiao, B.: Collaborative multimedia authoring: scenarios and consistency maintenance. In Proc. of the Fourth International Workshop on Collaborative Editing Systems, New Orleans, USA, November 2002