Towards Collaborative Video Authoring[1]

Boris Novikov and Oleg Proskurnin

University of St.Petersburg, Russia
{borisnov,olegpro}@acm.org

Abstract. It's a long time since video post production became digital, yet the issue of collaborative video authoring has not been seriously investigated so far. In this paper we tackle this problem from a transactional point of view, that allows us to ensure consistent exchange and sharing of information between authors by taking into account application semantics. Our main contribution is the development of a new data model called concurrent video, especially intended for cooperative authoring environments. We demonstrate that the presented model provides efficient means of organizing and manipulating video data, at the same time enabling direct use of merging mechanisms, which constitute a formal basis for collaborative scenarios. Moreover, since the proposed approach is mainly media-independent, we argue that the results of our work are applicable to other types of stream data as well.

1 Introduction

Today, computers are widely used to support the work of individuals. However, people tend to work in teams, whose members may be distributed over different locations. In addition, cooperation is inevitable in large-scale development and joint efforts help to meet tight deadlines in shorter projects. Thus, computers are required to support team activities just as well as individual tasks.

In case of video authoring collaboration often becomes an essential part of the process. This is especially true for the post production stage of movie-making when several people may be constantly involved in editing, adding visual effects and assembling a set of clips coming from common footage. Undoubtedly, appearance of adequate tools for multi-user video authoring is going to encourage closer cooperation among editors and hence enhance the quality of work that is carried out in both entertaining and broadcasting industries [9].

Since the goal of team work on a shared resource is to increase development productivity, cooperating users should not be disturbed with various difficulties that may arise from parallel actions of other participants. Therefore, ensuring consistency of shared data in the presence of concurrent actions becomes the key aspect in collaborative applications design. Actually, the latter happens to be the classical problem of transaction management, except that the traditional paradigm of competition for resources has to be replaced with an appropriate cooperation strategy.

[1] This work is partially supported by the Russian Foundation for Basic Research under grant No. 01-01-00935.

Such technique was presented in the cooperative activity model CoAct [15], which provides basic transactional support for interactive multi-user environments. In CoAct, each user operates in its own workspace that contains private versions of shared objects. User's actions are modeled via an activity history, which can be treated as a sequence of operations representing an individual working process. To achieve cooperation, co-workers can explicitly incorporate the results of their actions into the common workspace, which reflects the current state of collaborative effort. This incorporation can be viewed as semantically correct merging of private activity histories [20], guided by compatibility relations of constituent operations and involving the responsible user in case of a conflict. Thus, user transactions are executed concurrently in the context of a single cooperative transaction, enabling joint work and ensuring consistency of shared data.

In this paper we utilize formal aspects of the CoAct framework, which functionality corresponds to the needs of collaborative authoring environments [16, 1], in the field of cooperative video editing. For this purpose we develop the concurrent video data model, which provides an efficient tree-based abstraction for the underlying media stream and defines basic editing operations to support the authoring process. Furthermore, the model enables adequate maintenance of activity histories, facilitating utilization of the CoAct merging algorithm in the considered applications.

The rest of the paper is organized as follows. After giving a brief overview of related work, in section 3 we introduce the concurrent video data model. Next, in section 4, we discuss the transactional aspect of our proposal. Finally, we present conclusions and further work.

2 Related Work

In this section we first describe basic concepts underlying non-linear video editing and then briefly review previous approaches to cooperative multimedia authoring. Lastly, we summarize current research on video modeling.

2.1 Video Authoring

During the last decade video post-production techniques have moved from analog tape-to-tape editing to the world of digital non-linear editing systems. Computers brought enormous advantages to the video industry, leaving behind the drawbacks of work with sequential storage media.

Though there are various approaches to digital multimedia authoring [6], in this paper we will refer to well-known timeline-based systems to understand the basics of non-linear editing. We will keep in mind the functionality offered by representative commercial tools such as Adobe Premiere [3] and Ulead MediaStudio Pro [17] to identify the most general concepts underlying the video authoring process.

In applications we consider, the movie is created by altering and assembling source clips that are treated as independent units of raw material. These clips along with other graphical and audio objects are placed on the timeline, which naturally represents the time flow [18]. After the inclusion of necessary objects, they can be edited,

special effects and transitions can be applied and the resulting material can be arranged into a final video production. Thus, the authors are likely to operate on clips as a whole, rather than on particular video frames. In any case, such editing implies just referencing the original files, which are not actually modified, and hence a special process called rendering is required to preview or export the produced movie.

To the best of our knowledge, none of the existing commercial applications supports concurrent video authoring and the related research works are also very few.

In [5] and [19] cooperative editing of multimedia documents based on locking mechanisms is proposed, while in [24] the prospects of exploiting operational transformations in a replicated multimedia authoring environment are investigated. However, these approaches do not consider stream data modeling issues and do not provide consistency guarantees in a transactional sense.

2.2 Video Modeling

There exists a wealth of research proposals covering various aspects of video modeling, such as annotation, content-based access and segmentation. Taking into account the course of our work, we will focus on those models that consider composition of structured multimedia data.

A set of basic operations for temporal and spatial composition of video segments is defined in video algebra [23]. These operations allow users to create multi-window presentations by constructing possibly nested algebraic expressions. A hierarchy of annotated expressions forms the algebraic video data model, providing a logical representation to the underlying raw material and enabling content-based access.

OVID database system [14] introduces video-objects as an abstraction over sets of video frames. Each object has a unique identifier and an appropriate set of descriptive attributes, which can be inherited by newly composed elements on the basis of the interval inclusion relationship.

AVIS system [2] presents a tree-based structure for modeling video and its contents. Though this abstraction is mostly designed for annotation and query processing, adequate updating mechanisms are also supported. The latter makes this work in some respects the closest to our investigations.

A recent proposal [8] to a certain extent summarizes previous approaches, constructing a framework for segmentation, annotation, querying and composition of video data. In this model, composition operators are embedded into the query language and propagate the logical structure of the source videos to the resulting ones.

However, editing operations in the above systems, as well as those in some other related works [7, 10], will unlikely be sufficient for multimedia authoring environments. The problem lies in the fact that most mentioned models do not provide explicit support for such essential editing actions as insertion and deletion of video segments or frame-level manipulations.

Actually, a model called stream algebra [13] does provide a formal basis for operating on diverse kinds of streams in the context of the exploratory data analysis. It defines some useful editing operations, but it does not consider the logical structure of the underlying media and therefore can hardly be exploited in video authoring applications without any additional abstractions.

3 Concurrent Video Model

In this section we first emphasize some important issues related to modeling video data in collaborative authoring environments and then present our solution to these problems – the concurrent video model.

3.1 Basic Requirements

Generally, the main goal of our work is to develop a data model that would be suitable for the needs of video authoring techniques (1) and that would enable execution of cooperative activities (2).

More precisely, the first requirement implies the creation of an adequate internal representation of the media stream, that would capture the nature of the editing process and would provide efficient support for manipulating underlying video. Traditionally, such task assumes the development of an appropriate abstraction that merely references raw material, being separate from it, and also contains various additional information, i.e. meta-data. In this way, instead of working with vast amounts of multimedia structures, references to videos are manipulated, supporting efficient updating mechanisms and promising some other benefits [10].

At the same time the second requirement adds a transactional aspect to our investigations. Since in the CoAct framework the compatibility property is used as a basis for merging activity histories and, similar to conventional concurrency control [22], a semantic conflict test is defined for each possible pair of operation invocations, the model should provide editing operations with good commutativity properties, thus enabling high concurrency among co-workers' actions.

A more serious problem here is maintaining activity histories, which play the central role in merging facilities of CoAct. The point is that the presence of inverse operations, such as insertion and deletion of video segments, may lead to fictitious conflicts and accumulation of large amount of redundant data in these histories, as demonstrated later in section 4. Mentioned issues are not addressed in CoAct and thus certain mechanisms should be provided to enable correct use of the existing merging algorithm in the discussed applications.

Concurrent video satisfies both of the above requirements in a uniform and efficient manner, presenting a single data structure for modeling underlying media and maintaining activity histories. According to the concepts of timeline-based authoring, the model proposes video segments of an arbitrary length as the basic units constituting the media stream. It provides a tree-based structure for referencing this raw material and supports high-level operations allowing the users to insert and delete, alter and temporally combine video clips to form the final production.

To enable cooperative efforts, operations on different segments are considered to be commutative, i.e. different users can work concurrently on different clips without any conflicts. Moreover, activity histories are stored within the same hierarchical structure that references video data, providing elegant support for merging mechanisms and eliminating all problems related to the history maintenance.

In what follows we give the definition of the concurrent video data model, while the transactional aspect of our work is discussed later in section 4.

3.2 Video Segments

We introduce the notion of a video segment to 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 in addition has its own set of attribute-value pairs which describe the proper interpretation of the underlying media at the presentation and rendering level. Frame sequences reflect the stream-like nature of video data, while attributes support implementation of non-destructive editing operations.

Following are the definitions:

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.

To refer to different elements of the above entities and other objects introduced later we will use dot notation. For example, $S.F$ will identify the frame sequence of a given video segment S.

In general, the presented definitions are similar to those that can be found in previous research works, for example see [10], and our notion of a video segment is actually very close to the notion of a video object in [14]. However, we intentionally avoid timing specifications within frame sequences and do not interpret segment's attributes as mere annotations to the underlying data.

The reason why we do not explicitly consider the temporal aspect in the definition of a frame sequence is the assumption of constancy of the frame rate within a particular block of raw video, to a portion of which the sequence is supposed to refer. The corresponding frame rate can be easily indicated aside among the attributes of the given video segment, for example as $\{FrameRate{:}30\}$, thus allowing users to change the play speed of the selected clips in an efficient way and without modifying the source material.

In a similar manner, attributes can be used to specify the spatial resolution or the color mode desired in the final production. Moreover, modeling of special effects and transitions, which are typically applied to video clips, is also possible in this way, as it will be described in more detail in section 3.4.

Below, for convenience of the subsequent discussion, we define two trivial operations on frame sequences:

Definition 3 (concatenation) For any two given frame sequences $F = (f_1, \ldots, f_N)$ and $G = (g_1, \ldots, g_M)$ their concatenation is a new frame sequence $(f_1, \ldots, f_N, g_1, \ldots, g_M)$, denoted by $Concat(F, G)$.

Definition 4 (extraction) For any given frame sequence $F = (f_1, \ldots, f_N)$ of length N, an extraction within the range $[k, m]$, where $k \geq 1 \wedge m \leq N \wedge k \leq m$, is a new frame sequence $(f_k, f_{k+1}, \ldots, f_m)$, denoted by $Extract(F, [k, m])$.

Advanced versions of these operations can be found in [8].

3.3 Video Activity Tree

At this point we introduce the core part of the concurrent video data model, defining a hierarchical structure that arranges video segments in a single media stream and at the same time enables efficient maintenance of activity histories. Basically, each video activity tree represents an abstraction over a sequence of clips which can be edited and temporally combined by cooperating users to form the final production.

The proposed tree is a properly organized collection of nodes which serve different purposes depending on their position within the hierarchy. Particularly, certain leaf nodes, called valid, are associated with video segments constituting the media stream, other leaves, called dead, stand for previously deleted segments and intermediate nodes are mainly intended for holding related parts of the activity history. Actually, there exists a one-to-one correspondence between valid nodes of the tree and video clips visible to the users, and moreover, the total order imposed on leaf nodes by the tree structure reflects the order in which mentioned clips appear on the timeline. Additionally, since we are concerned with collaborative work, nodes as well as activity history elements are marked whether they are private or shared, indicating what part of the tree and the history is present in the common database and what part exists only in the considered local workspace.

The following is a basic illustration of the above concepts:

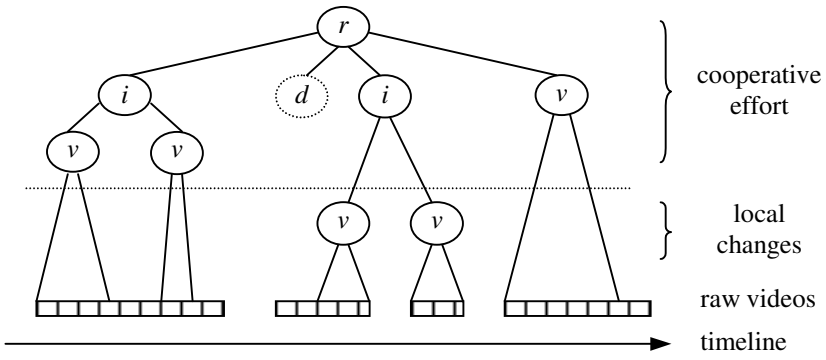

Fig. 1. A video activity tree in a local workspace. Characters *v*, *d*, *i* and *r* denote valid, dead, intermediate nodes and the root node of the tree correspondingly

One of the most important aspects of our data model is the assignment of unique identifiers to valid nodes of the video activity tree. The point is that utilization of these unique values for referring to video clips allows us to develop efficient editing operations with state-independent commutativity relations. This is achieved due to the fact that the considered identifiers do not depend on the actual positions of respective video segments on a timeline, which are also unique, but often change as the users insert new material into the media stream or remove unnecessary parts from it.

Generally, our approach is noticeably different from previous tree-based models, such as those presented in [2] and [12], which are mainly devoted to annotation and content-based querying of video databases rather than to video composition and cooperative work.

Now, let's move on to the formal definition of a video activity tree:

Definition 5 (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 nodes, which are all leaves of T, with and without associated video segments correspondingly.

For any given node $N \in \{Root\} \cup Nodes \cup Valid \cup Dead$ its parent node is denoted by $Parent(N)$ and a set of its child nodes is denoted by $Children(N)$.

Additionally, there exists a total order $<_t$ defined over a set of leaf nodes of T, that corresponds to the order in which associated video segments are located (or were located for dead nodes) on the timeline, i.e.: $\forall\, N, M \in Valid \cup Dead: N <_t M \vee M <_t N$.

For the clarity of further discussion we will not dwell on the implementation of the leaves order and child-parent relationships, which exist in a video activity tree. However, before presenting the internal structure of particular nodes, we will introduce the notion of an operation instance that is crucial for understanding activity histories:

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 and *Name* is the name of the corresponding operation, whose input and output parameters are reflected in *Input* and *Output* sets.

Intuitively, an operation instance can be treated as a record establishing the fact of execution of a certain editing operation by some user. Such records act as elementary entities for modeling the authoring process and building activity histories. Furthermore, they have unique identifiers for tracing identical instances during merging.

In the concurrent video model, operation instances are stored in the nodes of a video activity tree along with other relevant information in the following manner:

Definition 7 (valid node) A valid node V of a video activity tree is a tuple (*NID*, *Segment*, *History*), where *NID* is a unique node identifier of V, *Segment* is a video segment associated with V and *History* is an ordered set of operation instances related to the given node V.

Definition 8 (intermediate node and dead node) An intermediate node as well as a dead node of a video activity tree is merely a tuple (*History*), where *History* is intended for the same purpose as its counterpart in valid nodes.

From the above definitions it should be clear that in the presented model raw video data is referenced only by those video segments which are contained in valid nodes of the video activity tree. One of the key aspects is that these segments can be addressed by means of the dedicated identifiers, which are unique for each valid node. Such values can be generated, for example, by combining in a single entity a unique name of the current user and a trivial counter of identifiers that were assigned to particular video clips during the overall work of that user.

In general, all data structures defined above are managed by editing operations provided by the concurrent video model and partly by cooperation primitives which are responsible for information exchange in the collaborative environment and which are in fact supposed to update status of the operation instances only.

3.4 Editing Operations

Evidently, cooperating users involved in the authoring process are not supposed to know anything about the internals of the concurrent video model. The only related thing that they see is the timeline and the most common thing that they are likely to do is altering and assembling the selected video clips.

To support such kind of work we introduce appropriate editing operations, which can be easily utilized for modeling user's actions within a particular workspace. These operations adequately modify the video activity tree, preserving the correspondence between ordered valid nodes and video clips on a timeline and at the same time efficiently maintaining the activity history.

First of all, we define an initialization routine that should be used to form an initial state of the cooperative activity in the common database.

Definition 9 (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, forms a corresponding video activity tree with an empty history and yields no output:

1. Construct N valid nodes V_i: $\forall\ i,\ 1 \le i \le N$: $V_i := (NewID(), vs_i, \varnothing)$,
   where $NewID()$ denotes a function generating a new unique identifier.
2. Construct a video activity tree T: $T := (Root, \varnothing, \{V_i \mid 1 \le i \le N\}, \varnothing)$, such that:
   $\forall\ V_i$: $Parent(V_i) = Root \wedge \forall\ i, j$: $1 \le i < j \le N \Leftrightarrow V_i <_t V_j$.

The initialization algorithm is not considered as an operation that should be reflected in the activity history, it merely constructs an initial version of the video activity tree from the specified ordered set of video clips, which then can be manipulated by means of other operations to form the final production.

Typically, throughout the editing process authors may wish to include some new clips into their project as well as remove existing ones. This can be achieved with the help of insertion and deletion operations, which allow users to insert a particular video fragment into the specified position on a timeline or delete a selected video clip, accordingly shifting the subsequent material in both cases.

Definition 10 (insertion algorithm) The insertion algorithm takes as input a new video fragment, denoted by the segment vs, and its desired location on the timeline, specified by a destination point pos within the frame sequence of a clip with given identifier id. The algorithm splits the affected node of the video activity tree T, updates its history and yields a set of newly created clip identifiers $\{V_i.NID\}$ as output:

1. Find a node $V \in T.Valid$: $V.NID = id$, report failure if such node does not exist.
2. Construct an intermediate node V': $V' := (V.History)$.
3. *If* $pos > 0 \wedge pos < len$, where $len = Length(V.Segment.F)$, construct valid nodes V_i:
   $V_1 = (NewID(), (Extract(V.Segment.F, [1, pos]), V.Segment.A), \varnothing)$,
   $V_2 = (NewID(), vs, \varnothing)$,
   $V_3 = (NewID(), (Extract(V.Segment.F, [pos + 1, len]), V.Segment.A), \varnothing)$,
   Else construct only two of the above nodes having non-empty frame sequences.
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 $(private, NewID(), Insert, \{vs, id, pos\}, \{V_i.NID\})$ to $V'.History$.

Since the insertion algorithm is based on the node splitting technique, the affected valid node is replaced with a subtree whose leaves correspond to the resulting combination of involved video clips, as illustrated in figure 2. Still, the actual behavior of this operation depends on whether the user inserts new material between two adjacent video segments or inside one of them. The former case is accomplished straightforward, while in the latter case the destination clip should be properly divided into two independent fragments which inherit their attributes from the common predecessor.

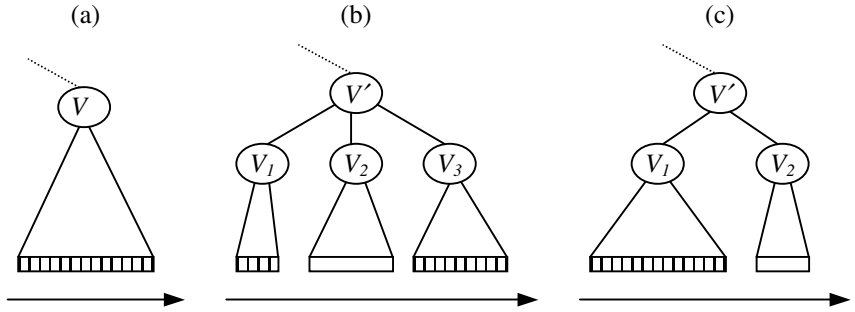

Fig. 2. Node splitting carried out during the insertion: (*a*) – an original valid node with the associated clip, (*b*) – the splitted node after an insertion inside the clip, (*c*) – the splitted node after an insertion to the end of the clip (insertion to the beginning of the clip is analogous)

Apparently, the discussed algorithm affects a bounded part of the video activity tree, operating on a single valid node and enabling commutativity of insertions executed within different video segments. Thus, users are able to add new material to different parts of the media stream concurrently.

At the same time, to delete previously inserted clips from the project authors are supposed to employ the next operation:

Definition 11 (deletion algorithm) The deletion algorithm takes as input an identifier *id* of the clip that has to be removed from the media stream. It correspondingly adjusts the video activity tree *T*, updates the history and yields no output:

1. *If* $\exists\ V \in T.Valid$: $V.NID = id$, *Then* find P: $P = Parent(V)$ and go to step 3.
2. *If* $\exists\ D \in T.Dead, \exists\ O \in D.History$: $O = (*, *, Delete, \{id\}, *)$, * denotes any value, *Then* report success, *Else* report failure.
3. *If* $\exists\ O \in P.History$: $O = (private, *, Insert, \{V.Segment, *, *\}, *) \wedge$ $\forall\ N \in Children(P) \setminus \{V\}$: $N.History = \varnothing$,
   Then undo the effect of the insertion *O* and report success.
4. Construct a dead node V': $V' := (\{O \in V.History \mid O.Status = Shared\})$.
5. Adjust T: $T := (Root, T.Nodes, T.Valid \setminus \{V\}, T.Dead \cup \{V'\})$, such that: $Parent(V') = Parent(V) \wedge \forall\ N \in T.Valid \cup T.Dead$: $V' <_t N \Leftrightarrow V <_t N$.
6. Append an operation instance $(private, NewID(), Delete, \{id\}, \varnothing)$ to $V'.History$.

The presented deletion algorithm has several important features. First, it is designed in a way to support commutativity between any two removals, even those of the same clips, as specified in step 2. Second, it acts as an inverse for some insertions,

that are not shared by other users and are not affected by subsequent operations, as checked in step 3. Evidently, such compensation can be performed by undoing the effect of node splitting, which may involve concatenation of the previously separated frame sequences, and by removing the corresponding insertion record from the history. Lastly, the discussed algorithm eliminates from the dead node those operation instances which are not shared and hence become irrelevant after the clip removal.

In general, the proposed operations provide efficient means of concurrent temporal composition of video segments within a single media stream. However, they do not provide adequate support for moving video clips along the timeline, since a pair of appropriate deletion and insertion methods will conflict with parallel editing of the moved clip. That's why we additionally develop the following algorithm:

Definition 12 (moving algorithm) The moving algorithm takes as input an identifier *mid* of the clip that has to be moved and its new location specified by *pos* and *id* parameters, which are similar to their counterparts in the insertion algorithm. The identifier of the moved clip is kept intact, the activity history is updated accordingly and a set of newly created node identifiers $\{V_i.NID \mid V_i.NID \neq mid\}$ is returned as output:

1. Find a node $M \in T.Valid$: $M.NID = mid$, fail if such node does not exist.
2. Find a node $S \in T.Valid$: $S.NID = id$, report failure if such node does not exist.
3. Apply steps 4 and 5 of the deletion algorithm to the node M to form a dead node D.
4. Apply steps 2 - 4 of the insertion algorithm to the nodes S (splitted) and M (moved) in such a way that: $\exists\, i$: $V_i \in Children(S) \wedge V_i = (mid, M.Segment, \varnothing)$.
5. Append an instance (*private*, *NewID*(), *Move*, {*mid*, *id*, *pos*}, $\{V_i.NID\}$) to the histories *D.History* and *S.History*.

In contrast with previous algorithms the one presented above alters two valid nodes of the video activity tree simultaneously and adds the same operation instance to the histories in both of them. But essentially, it is just a slightly modified sequence of deletion and insertion, which turns out to be commutative with the next operation:

Definition 13 (editing algorithm) The editing algorithm takes as input an identifier *id* of the clip that has to be modified and a new video segment *vs* which represents the results of the modification. The algorithm simply replaces an old video segment with the new one, updating the history and yielding no output:

1. Find a node $V \in T.Valid$: $V.NID = id$, report failure if such node does not exist.
2. Modify V: $V := (V.NID, vs, \{O \in V.History \mid O.Status = Shared\})$.
3. Append an operation instance (*private*, *NewID*(), *Edit*, {*id*, *vs*}, $\varnothing$) to *V.History*.

Obviously, this algorithm is a rough approximation to the actual modifications, such as image-level manipulations, which can occur within a video clip. Nevertheless, we leave the development of intra-segment editing operations to further research, emphasizing that even the presented methods ensure a high level of concurrency.

Moreover, due to existence of the last algorithm we can model transitions and special effects by means of certain attribute-value pairs, which may, for example, describe a sequence of filters applied to the corresponding video clip or indicate a type of transition to the next clip. The values of these attributes may be changed at any time without modifying the source material and after all they can be exploited during the rendering stage for constructing the desired video production.

4 Collaborative Authoring

In this section we present a general description of how cooperation can be achieved in a multi-user video authoring environment by means of the concurrent video data model. This part of our investigations is entirely based on the CoAct transactional concepts, which can be found in [11, 20] and for brevity are not stated here.

At first, we provide an efficient method of extracting correct subhistories from a set of operation instances scattered over the video activity tree, thus enabling selection of consistent units of work that can be subject of information exchange. After that, we look in more detail at the history merging mechanism, pointing out some related problems and examining how they are solved in the concurrent video model.

4.1 Consistent Units of Work

In the CoAct framework two distinct types of commutativity, forward and backward, are exploited for manipulating activity histories [11, 21].

Forward commutativity was already mentioned in section 3.4 and will be thoroughly discussed a bit later, as exactly this kind of relation is used for detecting conflicts between operations coming from different histories during merging.

Backward commutativity serves another purpose and is discussed below, as it is intended for determining dependencies between user's actions within a single history. The point is that the behavior of editing actions is typically influenced by some of the previously executed operations, for example, any modifications of a particular video clip may depend on the preceding insertion of this clip into the media stream. Hence, no operation instance can be exchanged between workspaces without its relevant predecessors. And it is backward commutativity that allows us to identify so-called closed subhistories [11] which have no external dependencies and thus represent consistent units of work. Evidently, such subhistories can be exchanged between cooperating users separately from each other.

Turning back to the concurrent video data model, we aim at the development of a technique for constructing closed subhistories out of a given video activity tree. This implies selecting required operation instances from the nodes of the tree and defining a proper total execution order over these instances. The latter brings an additional complication to our discussion since in contrast to the CoAct framework we do not maintain activity histories as totally ordered sets of constituting elements and thus, the sequential nature of the histories should be somehow recovered at the stage of information exchange to enable cooperation.

We start with the definition of the following partial order, which naturally extends nodes child-parent relationships present in a video activity tree to operation instances:

Definition 14 (hierarchical order) For any two operation instances A and B, contained in a video activity tree T, we consider that $A <_H B$ iff:

$\exists\, N \in T.Nodes \cup T.Valid \cup T.Dead$: A precedes B in the ordered set $N.History \vee$

$\exists\, P$: $P = Parent(N)$, A is the last element in $P.History$ and $B \in N.History$.

Actually, this partial order reflects all relevant dependencies which exist between non-commuting operations in a particular activity tree. In fact, it corresponds to the

restricted (comparing to [11]) backward commutativity relation, which ensures that the execution order of any two successive commuting operations from the given history can be exchanged without affecting the state of a video tree. This means that the last definition specifies a basis for extracting those operation instances from an activity tree which belong to the same closed subhistory.

Moreover, determination of an appropriate total execution order within the selected subhistory becomes straightforward according to the following lemma, which provides a way of constructing equivalent legal histories [11] out of a given video tree:

Lemma 1 Any two total orders $<_U$ and $<_V$ that are defined over a set of operation instances H which are contained in the nodes of a given video activity tree T, such that $<_H \subset <_U \wedge <_H \subset <_V$, form two legal histories $(H, <_U)$ and $(H, <_V)$. The video activity trees resulting from applying these histories to the respective initial workspace state are identical.

Owing to the lack of space, we will omit the proof of the above statements as well as the proofs of some subsequent facts, paying more attention to the final results.

Briefly summarizing the previous discussion we want to emphasize that the selection of a closed subhistory from a video activity tree and its total ordering can be successfully implemented by means of the hierarchical order alone. And since this partial order is naturally reflected by our tree-based data model, we can easily provide an elegant and efficient algorithm that can be used for extracting consistent units of work from a given workspace:

Definition 15 (subhistory extraction algorithm) The subhistory extraction algorithm denoted by *Subhistory* takes an operation instance O contained in a video activity tree T as input and yields as output an ordered list of operation instances from T representing a minimal closed subhistory [11] under $\{O\}$:

1. *If* $O = null \vee O$ is marked as already included in the subhistory *Then* return $\varnothing$.
2. Find a node $N \in T.Nodes \cup T.Dead \cup T.Valid$: $O \in N.History$.
3. *If* $O.Name = Move$ *Then* find a node $M \neq N$: $O \in M.History$ and go to step 6.
4. *If* $\exists A \in N.History \cup Parent(N).History$: $A <_H O \wedge \forall B <_H O$: $B <_H A$ *Then* $N_{prev} := A$ *Else* $N_{prev} := null$.
5. Mark O as already included in the subhistory and return $Subhistory(N_{prev}) \bullet O$.
6. Find N_{prev} and M_{prev} for nodes N and M like it is done in step 4.
7. Mark O as already included and return $Subhistory(N_{prev}) \bullet Subhistory(M_{prev}) \bullet O$.

Evidently, the extraction algorithm can be successfully used for constructing a minimal closed subhistory under a set of operation instances $\{O_1, \ldots, O_N\}$, with the help of mere lists concatenation (and under the assumption that inclusion markers are not reset between the function calls):

$List := Subhistory(O_1) \bullet \ldots \bullet Subhistory(O_N)$

Additionally, since any operation instance depends, possibly transitively, on all instances from the parents of its own node, we can supply a set of leaf nodes to the slightly modified version of the extraction algorithm for constructing the complete activity history of a given workspace. With such histories further exploitation of various aspects of the CoAct framework becomes possible.

4.2 Merging Activity Histories

In general, merging of activity histories in CoAct [20] is a semi-automatic process, which requires external control only in case of a conflict between users' actions. In what follows, we concentrate on utilization of the CoAct conflict detection technique, which does not suppose user interaction, within the concurrent video data model.

For this purpose we first present forward commutativity predicates of the concurrent video editing operations:

Table 1. Forward commutativity relation (symmetric)

Operations	*Edit*(*ID*, *VS*)	*Insert*(*VS*, *ID*, *POS*):*OUT*	*Delete*(*ID*)	*Move*(*MID*, *ID*, *POS*):*OUT*
Edit(*id*, *vs*)	$id \neq ID$			
Insert(*vs*, *id*, *pos*):*out*	$id \neq ID$	$id \neq ID$		
Delete(*id*)	$id \neq ID$	$id \neq ID$	*true*	
Move(*mid*, *id*, *pos*):*out*	$id \neq ID$	$mid \neq ID \wedge id \neq ID$	$mid \neq ID \wedge id \neq ID$	$\{mid, id\} \cap \{MID, ID\} = \varnothing$

Apparently, the predicates specified in the above table demonstrate that operations manipulating different video segments always commute. Moreover, the same clips can be deleted concurrently by several users, and video extracts moved along the timeline can be simultaneously modified by other participants. The latter is especially important since it allows cooperating authors to distinguish editing and ordering of particular clips as independent tasks. Thereby, the overall approach really offers a high level of concurrency to the co-workers, even though they are not encouraged to operate simultaneously on the same video clips.

Now, when proper commutativity relations are defined, they should be exploited to determine conflicts between users' activities. According to the history merging algorithm [20], two activity histories are basically considered to be compatible if their constituting operations pairwise commute. But generally, there may be certain situations in which this algorithm will lead to fictitious conflicts. For example, have a look at the following actions of Alice and Bob on some initial video activity tree:

Alice: [*Edit*(*ID*, *AliceVS*)]
Bob: [*Insert*(*NewVS, ID*, *0*):{*NewID*,..}] [*Edit*(*SomeID*, *BobVS*)] [*Delete*(*NewID*)]

Evidently, in his work Bob inserts a new video extract into the same clip which Alice edits and thus, a conflict is detected between these operations. However, later Bob removes the new segment from the media stream. Therefore, the mentioned conflict in fact stays only in the activity histories, but on the data level the results of such individual work are certainly compatible.

Fictitious conflicts, like one just described, take place since the deletion operation acts as an inverse for the insertion. This is the reason why we have designed the deletion algorithm of concurrent video in a way to rollback when possible the effect of the respective preceding insertion, as stated in definition 11.

Another noticeable aspect of history maintenance that we have introduced in our model is utilization of the masking concept [4]. Intuitively, a subsequent method masks a preceding one if the effects of the earlier operation turn out to be completely overwritten by the effects of the later one, thus, the former instance can be removed from the history without affecting the workspace state. In particular, the behavior of the deletion and editing algorithms, which remove preceding private instances from the processed node are based on the described concept.

As a consequence of these improvements in history maintenance, we become able to avoid certain fictitious conflicts and manage to reduce the amount of data kept within the nodes by removing masked operation instances. At the same time efficient implementation of the above features becomes possible due to storing of the activity history within a video tree.

5 Conclusions and Future Work

In this paper we have presented the concurrent video data model as a formal basis for collaborative video authoring environments.

The principal advantage of our approach lies in the fact that we have defined a single tree-based data structure for modeling both the underlying media stream and cooperative transactions. In spite of such duality, concurrent video undoubtedly satisfies basic requirements of authoring applications as well as the needs of multi-user environments in an elegant and efficient manner.

Moreover, the results of our work are general enough to be applicable to other types of stream data. For instance, by simple renaming of frame sequences and video segments into sample sequences and audio segments, the concurrent video model can be successfully transformed into concurrent audio that may be found suitable for cooperative editing of audio streams.

Finally, we consider incorporation of versioning support and development of intra-segment editing operations as probable directions of further research.

References

[1]. Aberer, K., Klingemann, J., Tesch, T., Wasch, J., Neuhold, E.J.: Transaction models supporting cooperative work – the TransCoop experiences. In Proc. of the Int. Symposium on Cooperative Database Systems for Advanced Applications, Kyoto, Japan, December 1996

[2]. 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

[3]. Adobe, Adobe Premiere 6.5: Available at http://www.adobe.com/products/premiere, 2003

[4]. Beaudouin-Lafon, M., Karsenty, A.: Transparency and awareness in a real-time groupware system. In Proc. of the 5th Annual ACM Symposium on User Interface Software and Technology, pages 171-180, Monteray, California, United States, 1992

[5]. Borghoff, U.M., Teege, G.: Structure management in the collaborative multimedia editing system IRIS. In Proc. of the International Conference on Multi-Media Modeling (MMM'93), pages 159-173, Singapore, 1993

[6]. 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
[7]. Duda, A., Keramane, C.: Structured temporal composition of multimedia data. In Proc. International Workshop on Multi-media Database Management Systems, Blue Mountain Lake, NY, August 1995
[8]. 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
[9]. Ghandeharizadeh, S., Kim, S.H.: Design of multi-user editing servers for continuous media. Multimedia Tools and Applications, Vol.11, Issue 1, pages 101-127, 2000
[10]. Gibbs, S., Breiteneder, C., Tsichritzis, D.: Data modeling of time-based media. In Proc. of ACM SIGMOD International Conference on Management of Data, pages 91-102, Minneapolis, Minnesota, May 1994
[11]. 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 (ATMA), pages 234-252, Goa, India, August - September 1996
[12]. Li, J., Goralwalla, I., Ozsu, M., Szafron, D.: Modeling video temporal relationships in an object database management system. In Proc. of IS&T/SPIE Int. Symposium on Electronic Imaging: Multimedia Computing and Networking, pages 80-91, San Jose, USA, 1997
[13]. Mackay, W.E., Beaudouin-Lafon, M.: DIVA: exploratory data analysis with multimedia streams. Conference proceedings on Human factors in computing systems, pages 416-423, Los Angeles, California, United States, April 1998
[14]. 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
[15]. 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
[16]. 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
[17]. Ulead, MediaStudio Pro 6.5: Available at http://www.ulead.com/msp/features.htm, 2003
[18]. Wahl, T., Rothermel, K.: Representing time in multimedia systems. In Proc. of IEEE International Conference on Multimedia Computing and Systems, Boston, USA, May 1994
[19]. 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
[20]. 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
[21]. Weihl, W.E.: Commutativity-based concurrency control for abstract data types. IEEE Transactions on Computers, Vol.37, No.12, pages 1488-1505, December 1988
[22]. Weikum, G.: Principles and realization strategies of multilevel transaction management. ACM Transactions on Database Systems, Vol.16, No.1, pages 132-180, March 1991
[23]. Weiss, R., Duda, A., Gifford, D.: Composition and search with a video algebra. IEEE Multimedia, Vol.2, No.1, pages 12-25, 1995
[24]. Xiao, B.: Collaborative multimedia authoring: scenarios and consistency maintenance. In Proc. of the Fourth International Workshop on Collaborative Editing Systems, New Orleans, Louisiana, United States, November 2002