

Change Awareness for Collaborative Video Annotation

Cristian Hofmann, Uwe Boettcher, and Dieter W. Fellner

Abstract Collaborative Video Annotation is a broad field of research and is widely used in productive environments. While it is easy to follow changes in small systems with few users, keeping in touch with all changes in large environments can easily get overwhelming. The easiest way and a first approach to prevent the users from getting lost is to show them all changes in an appropriate way. This list of changes can also become very large when many contributors add new information to shared data resources. To prevent users from getting lost while having a list of changes, this paper introduces a way to subscribe to parts of the system and only to have the relevant changes shown. To achieve this goal, the framework provides an approach to check the relevance of changes, which is not trivial in three dimensional spaces, and to be accumulated for later reference by the subscribing user. The benefit for users is to need fewer time to be up-to-date and to have more time for applying own changes.

Introduction

When working alone on a single document the user always knows what and where changes occur. But as soon as more users collaborate on one or more documents, they need to realize all changes made to the documents that are modified by the community. If the applied collaborative system only provides viewing and modifying

C. Hofmann (✉) and D.W. Fellner

Fraunhofer Institute for Computer Graphics Research Darmstadt, Fraunhoferstr. 5,
64283 Darmstadt, Germany

and

Technische Universität Darmstadt, Interactive Graphics Systems Group, Fraunhoferstr. 5,
64283 Darmstadt, Germany

e-mail: cristian.hofmann@igd.fraunhofer.de; d.fellner@igd.fraunhofer.de

U. Boettcher

Technische Universität Darmstadt, Interactive Graphics Systems Group, Fraunhoferstr. 5,
64283 Darmstadt, Germany

e-mail: boettcher@gris.informatik.tu-darmstadt.de

of documents, any user has to inspect all documents carefully to notice all changes which have been made. As that may be a time consuming and defective activity, users must spend much of their productivity to avoid errors and track the new modifications in an appropriate way. Thus the system does not scale well with increasing number of users. The more users there are, the more changes must be inspected prior to applying own changes. But the main goal of collaborative systems is to scale well with increasing number of users to multiply productivity. Consequently, there must be a way to avoid spending much time on getting up-to-date. The system has to announce changes to the users, so they do not have to search for them. This protects collaborating users from missing any change that has been made. Particularly, that also reduces the time needed to inspect all changes for relevance. As a second step, in order to additionally reduce the time needed to get up-to-date, certain filters can be implemented to further reduce the time needed to get up-to-date. The user selects videos or parts thereof to follow and only changes of these specific elements are shown. In doing so, the user can spend a great share of actual productive time. When trying to apply filters to videos, there are specific constraints to be solved.

The main contribution of this paper is to present an environment for Collaborative Video Annotation that supports tracking of changes by applying suchlike filters to video-based media. In doing so, we expect to alleviate essential activities that users have to perform when annotating videos in collaborative settings. Here, we introduce a way to support filters in Collaborative Video Annotation to assist the users with *Change Awareness*. By allowing the user to select which informations to be filtered and which not (rather than filtering automatically), a maximum control for the user is assured. The presented concepts have been developed in the scope of the THESEUS program. THESEUS is a research program, initiated by the Federal Ministry of Economy and Technology (BMWi), to develop a new internet-based infrastructure in order to better use and utilize the knowledge available on the internet.

Fundamentals

First, we introduce the involved fundamentals and specific characteristics Video Annotation and Collaborative Video Annotation as a sub-category. Finally the field of Change Awareness will be illuminated. There, the main question is: Why do you need Change Awareness?

Video Annotation

In general, Video Annotation means applying additional content to videos. This content can be of different types, for example metadata or additional multimedia content. Metadata can either be descriptive [1, 5] or semantic [3]. Additional content can be of numerous formats: texts (notes, comments, discussions, etc. [5]), multimedia (sounds, videos, etc.) [1] or other formats containing certain information.



Fig. 1 Comparison of annotations in books and in videos [7]

While metadata is usually designed to be read by machines, additional content should be displayed to the users [1].

This concept is closely related to side notes applied to books by readers. These are applied to make it easy to understand difficult parts of the book either for oneself or for fellow readers. Sometimes they are applied to remember additional information found in other books or own thoughts to be followed later on. Annotation to books is obviously of the second type: additional user information, and can only be used for texts or user drawn graphics. In the scope of digital video content, this limitation is obsolete, since you can display a much broader selection of content. Figure 1 shows the concepts of Annotation to books and to videos. You can easily realize the similarities and the differences.

While annotation to books is easy and only a pen is needed, annotation to videos has to be supported by the applied video player [5]. This is an important difference between the two worlds, as book writers can be sure that readers are able to annotate, while video creators do not have that guarantee. This problem has to be solved with a specific player for videos supporting annotations. Also videos have a different structure compared to books. Videos consist of events, objects, and scenes as temporal parts and they show moves as spatial parts [5, 9].

Within or upon a video, annotations consist of three parts: an anchor in the source video selecting the topic to be annotated, the annotation (of any player supported type which might also include other videos or annotated videos) and the link connecting these two items. The anchor can be any spatiotemporal object contained in the source video, for example a scene or a specific object shown. Figure 2 shows the principal structure of annotated videos. It shows the videos, annotated objects and the links connecting them.

Collaborative Video Annotation

Collaborative Video Annotation means having the principles of Video Annotation in collaborative environments, whereat many users annotate the same video or videos. It can be used in different contexts and use cases such as learning environments

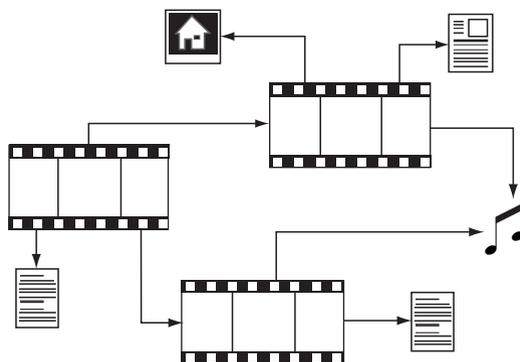


Fig. 2 Structure of annotated videos [7]

[5, 8, 18] or video analysis [6, 13]. A growing number of application scenarios for (collaborative) video analysis in education can be identified. Pea and colleagues report on a university course of a film science department, in which two different movie versions of the play “Henry V” are analysed by a group of students with respect to the text transposition by different actors and directors [13]. Other examples for the application of video analysis in education are motion analyses in sports and physical education, or the acquisition of soft skills such as presentation or argumentation techniques [10, 13, 14]. Bruggmann et al. [4] describe another example where a shared group of scientists work on papers about sign language [4]. As this topic is underrepresented in science, scientists are far away from each other. These ways are commonly overcome by travel which requires much time. As the scientists are often deaf-mute themselves, telephone conferences are no solution. Video conferences could be used, but the sign language demands a great share of the scientists attention and leads to few attention for the topic discussed over. The participants must have two videos in attention, one showing the speech and the other the topic. With video annotation the speech can be textual and the scientists can rewind and see the same part again if that matters [4].

In greater projects many people are working together on one document. The number of modifications can rapidly become overwhelming especially in asynchronous systems, because the numerous modifications and points of modification exceeds the humanly possible [2, 16]. Thus the participants coherent work can no longer be ensured [13] and mistakes will happen.

Change Awareness

Mentioned problems (having overwhelmingly numerous changes and thus losing coherence) can be mitigated by providing Change Awareness. That basically means to track down every change in any document contained by the system and to inform the user about these changes. “Information elements include: knowing who changed

the artifact, what those changes involve, where changes occur, when changes were made, how things have changed and why people made the changes” [17]. Thus the information is supporting the user in his work and may yield to more productivity. Papadopoulou [12] defines Change Awareness as “the ability of a person to track the changes that other collaborators make to a group project at any time while interacting with it” [12].

Change Awareness can have two different underlying principles. The first system is that one user does changes in the system and the other users follow these changes live while having some sort of communication (e.g. telephone conference). This way does automatically fulfill all the aspects called for by Tam and Greenberg (who, what, where, when, how, why) [17]. But this way does only work in synchronous systems as all users have to follow the changes live, while the framework presented in this paper should be indifferent about synchronous and asynchronous. The second way is to accumulate the changes and save them into the database. Then the server informs the user about the changes made (either with login or with some other message the user sends to the server). This results in the user being always well-informed about all changes made, knowing what happens where, and able to make own changes without interfering with others [12].

This basic principle must be further enhanced when larger projects should be handled by the server. If projects grow larger, the number of changes increases proportionally. That results in the changes itself becoming overwhelmingly numerous. So a strategy must be found to show only relevant changes. That means a type of filter must be implemented. The basic variant of filtering in this case is to have the user decide which parts of the project are relevant to him.

This yields to a new problem in regard to Video Annotation: the annotations are connected with anchors, which may be three dimensional spatiotemporal objects contained in a video, as there are two dimensions in space (x-axis and y-axis) and the time as third dimension. These anchors can and will overlap with other anchors and if the user for example is interested in the car shown in a video, he may also want to know about changes concerning the tire. That means the tracking system should include some sort of test for overlapping anchors.

To gain Change Awareness, the system has to track every change the users commit to the documents. The tracking objects are known as Awareness Information: “... the value of information elements that need to be collected and presented to users, to preserve their awareness” [12]. Tracking changes may result in a higher productivity because the users need fewer time to get up-to-date while staying sure not to miss any relevant change [11].

However it should be remembered that some contributors will want to have all changes in any document displayed. Papadopoulou describes this as role dependent: “The awareness information people require when collaboratively authoring a document is highly related to their roles as well. Some users may need to be informed about every last detail of changes made to a document while others may be satisfied with information showing only major revisions” [12]. She explains further that lawyers for example have to know every change of a contract while

professors only need to have an overall view to the documents and need only be informed about major changes [12].

Related Work

Tam and Greenberg [17] introduce asynchronous Change Awareness to Collaborative Systems. They take a broad view on Collaborative Systems in general and show in detail what is needed to make the user able to be change aware. The theoretical basics for tracking changes are discussed and a framework is introduced. But the main goal for Tam and Greenberg is to provide theoretical background for others to research further on. They neither built a prototype nor applied their findings to any real case. In their conclusion they say that “we have to analyse the need for change awareness in particular application domains. By understanding these differences, custom versions of the framework tailored to that domain may be produced” [17].

Papadopoulou [12] introduces a framework for change awareness in collaborative systems. In her dissertation she describes how this framework can be used to enhance a text editor with contributor’s awareness for changes. She also discusses the problems related with concurrent work of multiple users on the same document, which may lead to users change documents which already changed after they downloaded the contents. Additionally she discusses the privacy issues of collaborative systems in detail and concludes with further uses of her framework for other fields of work. She states that if “more than one user has worked on the same document part, possible conflicts to be resolved may arise. In the worst case, the changes of one user might have to be discarded, which could be disappointing and time-consuming for the users” [12]. While this is very similar to this paper, she focuses on texts and graphics and did neither discuss nor examine the specific hindrances to be solved when implementing Change Awareness into Collaborative Video Annotation Systems.

The Vannotea project [15, 19] implements some rudimentary forms of change tracking. But it reveals that the method collaborators are made aware of changes is not tailored to the special needs of video annotation. The changes are announced via RSS feeds delivered by the annotation servers. These feeds include all changes made. The filtering should be further improved to be configurable to the special needs of any contributor. As Papadopoulou [12] figured out there are different roles which contributors may take on. Depending on these roles, different filtering methods should be applied [12].

Framework Requirements Analysis

Based on the aspects pointed out in the fundamentals section with respect to Change Awareness, we examined existing systems for Video Annotation. We checked if and how these systems provide Change Awareness to their users and if any filtering is provided.

Many state of the art-systems do not provide any Change Awareness explicitly. Those which do, only provide some basic level of Change Awareness. They do not track all changes, or they track all changes but do not provide filtering in a satisfying way. None of the systems which are state-of-the-art does provide filtering of the Awareness Information for specific spatiotemporal parts of the video(s) (which is essential as figured out above).

By means of that state-of-the-art analysis, we were able to model the following requirements for our framework: *Subscriptions*, *Overlap Detection*, *Event Triggering*, *Change Tracking*, *Awareness Information Providing*, *Server Messages Enrichment*, and *Client Enrichment*. In the following, these requirements are described.

Subscriptions

The system has to provide a way to subscribe to certain parts of the database. As we discussed earlier this should not only target videos but also parts of a video. To achieve this goal, the underlying Object Model must be enriched to allow to attach subscriptions to any object representing content. This includes (not intended to be exhaustive): Accumulations of videos, videos itself, scenes or objects contained in videos, or comments attached to videos.

Additionally any change to an object should be detected and provided to the corresponding subscriptions. This should be least intrusive to the Object Model to allow for easy integration into existing systems or reuse.

Overlap Detection

The objects or scenes shown in videos do naturally overlap. Cars shown have tires, a dialog has different speakers involved, and chapters are divided into several locations shown. When a change occurs on any object contained in or overlapped by another one, the Subscriptions to this other object should also be informed.

While this goal seems trivial at first and is easy to implement for video aggregations (e.g. documents) or scenes of videos, there are much more hindrances when it comes to parts of videos (e.g. objects contained in the video). Parts of videos are three dimensional spatiotemporal objects, so detecting if two parts overlap means solving a three dimensional overlapping problem.

To minimize time needed to announce changes, which is crucial to synchronous systems, overlaps should be precalculated. This can easily be implemented using adapters for the objects to which Subscriptions are applied. These adapters track which other objects (adapters) are overlapping to themselves. If a change occurs the object informs the corresponding adapter, which acts like a single subscription to the object. These adapters inform not only their subscribers but also the overlapping object's adapters.

Event Triggering

The server's implementation has to be enhanced with an event triggering system for change events. Any change made should trigger an event which is passed to the subscription system. This system handles the event appropriately. All (straight or oblique) subscribers should be informed about the change event and the triggering action.

Change Tracking

Any change made to an object should be announced to all of the object's subscribers. The awareness information should include [17]:

- Who changed the artifact?
- What are those changes involving?
- Where occurred these changes?
- When were the changes made?
- How have things changed?
- Why made people these changes?

As this should also work asynchronously, the tracked changes have to be persisted. The awareness information is saved for future use by the subscriber, and should be persisted and later presented to the user.

While the first five items are automatically known when a change occurs, the last item is different and needs specific handling by the client. As it is not clear how the user reacts on having to provide a reason for every changes he does, this part of the Awareness Information is optional on the server side of our implementation and not supported by the client.

Awareness Information Providing

Finally the server should be enhanced to be able to provide the awareness information to clients. That means a message should be implemented which retrieves the awareness information. This message should be parameterized:

- Show changes since last message by client
- Show messages marked to be read later
- Show all changes saved in database for the user

The default setting should be to provide all changes since last message exchange between server and client, and the messages marked for later reading. With parameters the client can additionally choose any of the three items above.

Also the client should be able to delete any awareness information in the database. To provide this option, the server should remember the awareness information individually for each user.

Server Messages Enrichment

While most changes to the system will occur while the user is not logged in, some will happen while the user is present. So the user should not only be informed about changes with login but instead be informed with any message submitted.

To slim the message exchange, only key information should be added to messages. Each message should be enriched with a short note providing information if any new change happened. Additionally the client should receive a note about messages marked to be read later when the user logs in. It is the clients responsibility to act properly on the information provided (e.g. to inform the user and to allow the user to retrieve the awareness information).

Client Enrichment

The only step taken regarding the client is to implement a notice to the user, which is shown whenever the server notifies the client about any change made. The user then decides how to react on the changes. He may disregard the notice, take a short look and continue his work, or interrupt to respond on the changes.

This message the server shows to the user should be non-disturbing with regard to the work flow of the user, but prominent enough to be recognized in any circumstances.

Results of the Requirements Analysis

As a result to this analysis, the server should be divided into five parts: the Message Interface, the Command Execution Logic, the Object Model, the Subscription Management, and the Persistence Interface. These are shown in Fig. 3 and be described as follows:

- *Message Interface*. This part of the server receives the messages of the client, parses the relevant data and hands it to the appropriate part of the Command Execution.
- *Command Execution Logic*. In this part the program logic resides. The relevant objects are retrieved from the Persistence Interface, the modifications are applied, and the objects are persisted again.

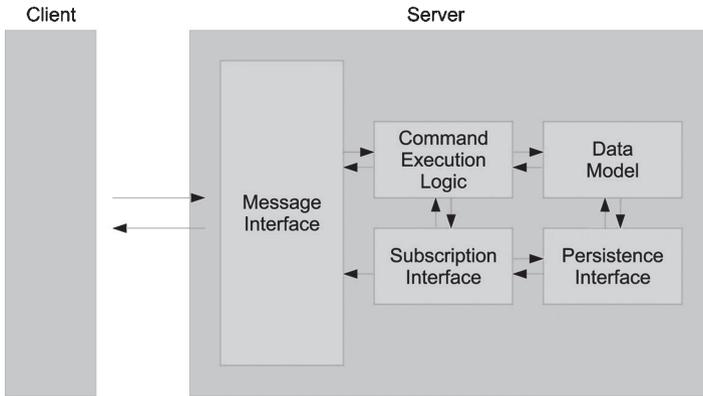


Fig. 3 Global model of the server

- *Object Model*. The Object Model will be described in detail in Chapter 5.
- *Subscription Management*. All changes in any object are handed over to the Subscription Management to be persisted for later use by any user of the system.
- *Persistence Interface*. The Persistence Interface provides simple methods to store and retrieve objects. All objects stored are persisted.

A Framework for Change Awareness in Collaborative Video Annotation Systems

We implemented a video annotation system providing Change Awareness as described in this paper. The application consists of a server and a client application. Our implementation can be used as a framework to enrich other servers with Change Awareness or to implement own servers for other topics. Following, we will describe what we did in general.

Object Model

The object model of the server is based on Finke’s model for hyperlinked video [5]. In order to support Change awareness, we enlarged that model with specific classes which are going to be explained below. In Finke’s hypervideo model, the base classes are modeled to represent the general objects needed to provide different projects containing annotated videos or other information to the users. These classes shown in Fig. 4 are:

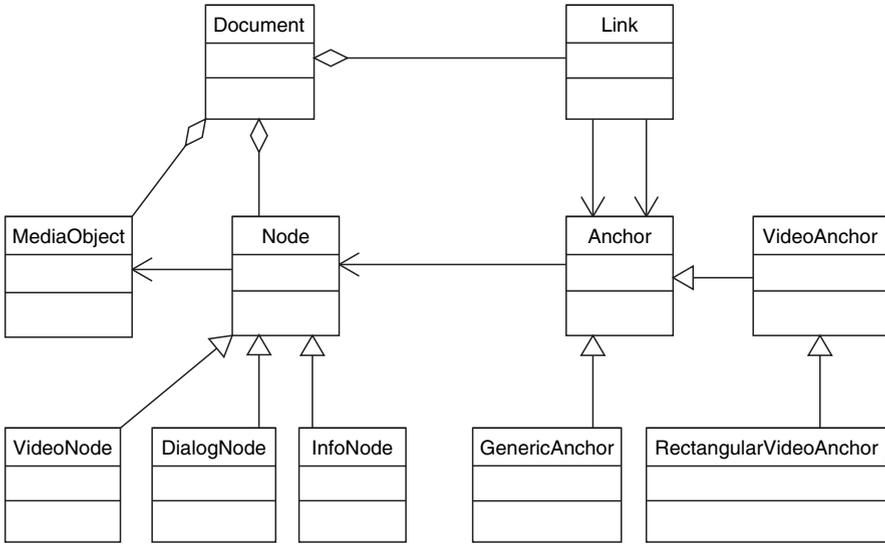


Fig. 4 The object model, which this framework is based on [5]

- *Document*. This class is the container class representing a project. All informations contained in a Document belong to a single context. This context usually is a base video to be discussed by the community. To this base video all comments (which may be of any media type as already figured out) are added either directly or indirectly.
- *Node*. Any information fragment of a specific type is contained in a Node object. A Document may and will contain different Nodes referencing to each other as annotations. In general the server is able to annotate an annotated video to another video, thus chaining nodes together.
- *VideoNode*. The base Node of a Document in our field of work will be a VideoNode, which contains a video able to be annotated. In the Object Model, any media will be able to be annotated.
- *DialogNode*. Not all annotations to the base video will be other videos. In fact, most will be texts, which are stored in DialogNodes. These are also able to be annotated, but in our implementation only the whole DialogNode may be annotated. This could be different for other fields of work. DialogNodes may additionally have sub nodes, which have to be DialogNodes, too. This enables a forum-like discussion contained in a tree of DialogNodes.
- *InfoNode*. All other informations are stored in InfoNodes. These can form a tree, too.
- *MediaObject*. Any Node contains a MediaObject, in which the data is encapsulated. A video for example could have different formats like MPEG or Flash. This information is encapsulated in the MediaObject class to free the VideoNode of this differentiation. Thus the VideoNode can reference to any video with the same methods, regardless of format.

- *Anchor*. This class implements a way to select parts of any media contained in a Node. These are used to select the topic of annotations. Anchors may have different types, depending on the data.
- *VideoAnchor*. These select any spatiotemporal part of a video to make this part able to be referenced and annotated. VideoAnchors may have different shape depending on the object to be annotated. These shapes have to be implemented. In our implementation only a rectangular shape is available, but this can easily be enhanced with additional types of shape.
- *GenericAnchor*. When no other type of Anchor fits, this type can be utilized to refer to the whole Node.
- *Link*. Links connect anchors to enable annotations as described in Fundamentals. Links connect a source Anchor representing the object which should get an annotation with a destination Anchor representing the annotated object. In general, this could also be only part of a video, but in our implementation only GenericAnchors are used for destinations.

To provide Change Awareness, there has to be a way for users to select which parts of a project should be tracked. Thus, some objects must be added to this Object Model to support Change Awareness. These additions are shown in Fig. 5 in gray and described here:

- *Subscriber*. This interface represents any Subscriber. This interface just provides a way to announce a change. All objects containing data that may be changed

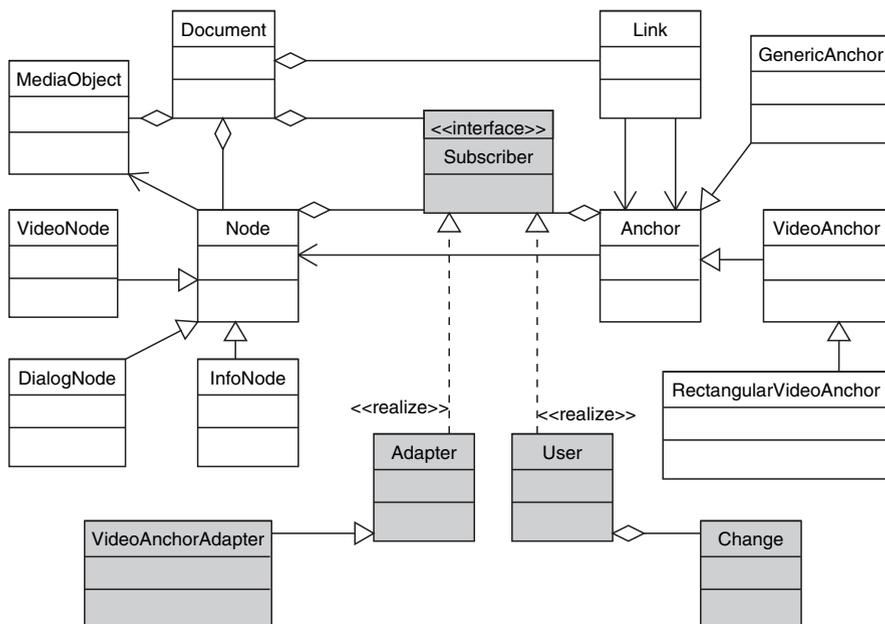


Fig. 5 Final object model of the framework

and tracked provide a way to register Subscribers with them. These Subscribers can be of different type to allow for Overlap Detection.

- *User*. User are the standard Subscriber. User represents any user registered with the system. All User objects keep track of Changes to be prepared to present them to the corresponding human. User may additionally hold any login credentials or user defined settings (e.g. default project to open).
- *Adapter*. The other type of Subscriber is an Adapter to other Subscribers. Whenever an object could overlap with other objects of the same type, an Adapter is created to keep track of all overlaps. This means, whenever an object is created for which an Adapter exists, a corresponding adapter is also created. This adapter is attached to the object and functions as a hook for other subscriptions to that object as an Adapter can also be subscribed to. Secondly any created adapter checks, if any already existing adapter represents an object which overlaps with its own object. If there is any, it saves this overlap for future use and announces that overlap to the other anchor as well.
- *VideoAnchorAdapter*. In our field of work, only VideoAnchors can have relevant overlaps. Thus we only implemented this specific Adapter. Other adapters could be implemented as well taking this one as an example.
- *Change*. This object represents any change made to any object. It stores the Awareness Information that should later be presented to the user.

All objects supporting subscriptions announce their changes to all subscribers, either an adapter or a Subscription object.

Overlap Detection and Change Tracking

The adapters provide methods for detecting overlaps. As there are anchors tailored to special needs, the adapters must also be tailored to a specific Anchor class. Each tailored anchor may or may not have an adapter class, as there could be anchors which can never overlap. With regard to video objects the adapter class has to solve a three dimensional overlapping problem. The adapter first determines if one of the two Anchors is wholly included in the other. Thereafter it checks if any edge of one anchor penetrates the surface of the other anchor. If either of these conditions applies, an overlap is found. Each anchor represents a three dimensional object. Mathematically these objects have 12 edges and six surfaces. These edges can be defined as follows (with g and h being points on the edge):

$$e = \{x \in R \mid x = g + k \cdot (h - g)\} \quad (1)$$

The surfaces are inside planes. These planes are in vectorial form and in Hesse-Normalform (with p , q and r being corners of the surfaces, n the normal vector of the plane, and a the distance between the plane and the origin):

$$P = \{x \in R \mid x = p + m \cdot (q - p) + n \cdot (r - p)\} = \{x \in R \mid x \cdot n = a\} \quad (2)$$

The Anchor can mathematically be defined as intersection of six half-planes defined by the six surfaces. These half-planes are:

$$H_k = \{x \in R \mid x \cdot n_k \geq a_k\} \quad (3)$$

The anchor can thus be written as:

$$A = \bigcap_{k=1}^6 \{x \in R \mid x \cdot n_k \geq a_k\} \quad (4)$$

To check for an overlap, you simply calculate the intersections of the 12 edges of one Anchor with the six surfaces of the other Anchor. This results in 72 points. These 72 points are then checked, if one of them is inside both Anchors. (There fortunately must only be checked once, as you used the edges of one Anchor, thus the points are by design inside this Anchor.) If one of the points is inside both anchors, an Overlap is found.

If an overlap is found, the overlapping anchor's adapters save this information for later use. Whenever any change occurs in one of these two connected anchors, the subscribers of both anchors receive the corresponding Change object.

Visualization

This section describes how users are informed of changes occurred on runtime or during absence. As already mentioned, the user notice should be displayed in a non-disturbing way. However, notices should be distinctive enough, so that the user is able to recognize them whatever he is doing in that moment. Figure 6 shows a screenshot of the client noticing the user that parts of the annotated video have been modified. The client uses no popup window to notify the user, since this would disturb users in the context of their actual activities. Instead, the client provides a panel in the upper right corner of the graphical user interface. In that panel, a red dot indicates new changes drawing only necessary attention to be noticed in any circumstance. The user realizes that something requires his attention, but can work on until a state is reached in which the current work can be interrupted in order to focus on the advised changes. Additionally, the user is instantly informed if a change is connected to a part of his actual work. To accomplish this, the client shows a red sign on the upper corner of each anchor which is connected to a change event (see Fig. 6).

After selecting the "Changes occurred" button, a list of actual modifications are displayed on the changes panel. Figure 7 gives an example, how the changes list is presented to the user. The changes list is linked to the related object. A click on any element opens the related object in the browser window. Thus, users are enabled to follow respective changes in a direct way (see red line in Fig. 7). Furthermore, three

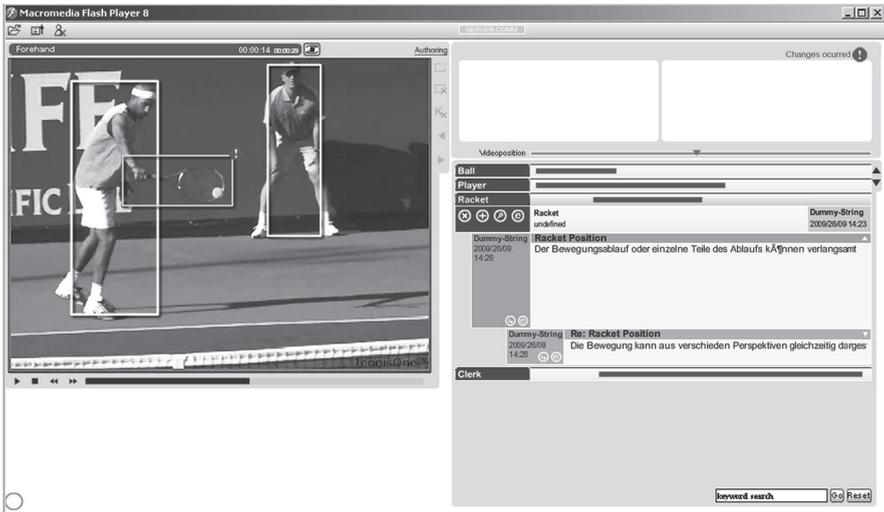


Fig. 6 Client notifying new changes to the user

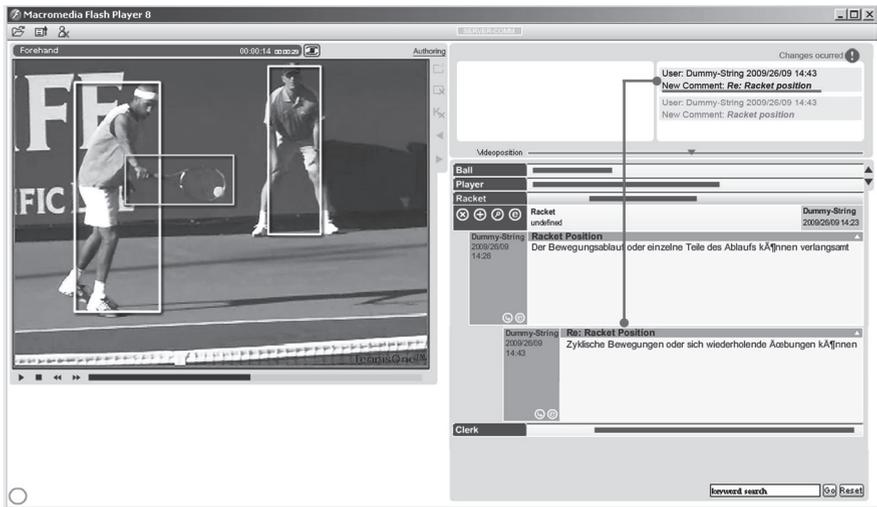


Fig. 7 Client presenting the changes list to the user

of the relevant questions are answered on the spot: You see who made the change, when the change was made, and how things changed. With a simple click the user can open this related object and inspect the changes (which are still marked as new). This link answers the other two questions (what and where), which cannot be answered in textual form in the changes list.

To sum up, users are notified about changes made in the current video project by means of a specific visualization. The integrated changes list provides information about who, when, and how changes were made. Furthermore, elements of the change list are connected to the respective area on the video. Thus, users can follow the changes by viewing the respective contents on the one hand, and information is provided about where changes occurred and what has been modified.

Conclusion

Keeping the user informed about all that is happening is an important field of work when working collaboratively. Users not knowing what others do cannot contribute to the collective work properly, because they might miss others' work and can thus leave no comment, or they might do work which is already done.

While some Collaborative Video Annotation systems already support keeping track of changes, they still do not allow users to select specific parts of interest instead of a whole entity. But, as we have seen, it is necessary to provide that information properly.

We pointed out how the system should be designed, and what circumstances must be fulfilled to help the user solve this problem. With the presented framework, any collaborative video annotation systems can be enhanced by providing proper Awareness Information to keep the user up-to-date.

As part of future research activities, we did not implement or suggest any kind of recommender system. These could be used to assist the user in finding more filters which could be relevant to the annotators' interests. We did not bring any illumination to this field of work, which could enhance the productivity further.

References

1. Agosti, M., Ferro, N.: A Formal Model of Annotations of Digital Content. In: *ACM Transactions on Information Systems (TOIS)*, 26,1(3). 2007
2. Bergmann, R., Niederholtmeyer, C.: *Arbeiten im Internet. Virtuelle Arbeitsgruppen in Non-Profit-Unternehmen und Bildungseinrichtungen.* hiba Forum Band 24. hiba Verlag. 2003
3. Berners-Lee, T., Hendler, J., Lassila, O.: The Semantic Web. In: *Scientific American*, 284(5): 34–43. 2001
4. Brugman, H., Crasborn, O., Russel, A.: Collaborative annotation of sign language data with peer-to-peer technology. In: *Proceedings of the 4th International Conference on Language Resources and Evaluation*, pp 213–216. European Language Resources Association, Paris. 2004
5. Finke, M.: *Unterstützung des kooperativen Wissenserwerbs durch Hypervideo-Inhalte.* Universitäts- und Landesbibliothek Darmstadt (2005)
6. Hagedorn, J., Hailpern, J., Karahalios, K.G.: VCode and VData: illustrating a new framework for supporting the video annotation workflow. In: *Proceedings of the working conference on Advanced visual interfaces*, Napoli, Italy. pp 317–321. 2008

7. Hofmann, C.: Entwurf und Neuimplementierung einer Hypervideoanwendung für kooperativen Wissenserwerb. Diploma Thesis, Universität Siegen. 2006
8. Hofmann, C., Hollender, N.: Kooperativer Informationsaustausch mit Hypervideo: Potentiale für das Web 2.0. In: Proceedings of the Pre-Conference Workshops of the DeLFI 2007, Logos Verlag, Berlin. 2007
9. Hofmann, C., Hollender, N.: Kooperativer Wissenserwerb und -Austausch durch Hypervideo. In: M & C '07: Proceedings of the Mensch & Computer 2007: Konferenz für interaktive und kooperative Medien, Oldenbourg, pp 269–272. 2007
10. Hollender, N., Hofmann, C., Deneke, M.: Principles to reduce extraneous load in web-based generative learning settings. In: Workshop on Cognition and the Web 2008, Granada, Spain, pp 7–14. 2008
11. Kim, H., Erklundh, K.S.: Collaboration between Writer and Reviewer through Change Representation Tools. In: Proceedings of the 35th Annual Hawaii International Conference on System Sciences, IEEE Computer Society, Washington, DC, USA. 2002
12. Papadopoulou, S.: Multi-Level Change Awareness for Collaborative Authoring Applications. Dissertation at the ETH Zürich, Zürich, Switzerland. 2009
13. Pea, R., Lindgren, R., Rosen, J.: Computer-supported collaborative video analysis. In: 7th International Conference on Learning Sciences, pp 516–521. International Society of the Learning Sciences, Bloomington, IN. 2006
14. Richter, K., Finke, M., Hofmann, C., Balfanz, D.: Hypervideo. In: Pagani, M. (ed.) Encyclopedia of Multimedia Technology and Networking, 2nd edition, pp 641–647. Idea Group Pub, Hershey, PA. 2007
15. Schroeter, R., Hunter, J., Guerin, J., Khan, I., Henderson, M.: A Synchronous Multimedia Annotation System for Secure Collaboratories. In: Proceedings of the 2nd IEEE International Conference on E-Science and Grid Computing, Amsterdam, The Netherlands (eScience 2006), p 41. 2006
16. Schulmeister, R.: Grundlagen hypermedialer Lernsysteme: Theorie – Didaktik – Design. Addison-Wesley, Bonn. 2002
17. Tam, J., Greenberg, S.: A framework for asynchronous change awareness in collaborative documents and workspaces. In: Int. J. Hum.-Comput. Stud. 64, 7 (Jul. 2006), 583–598. 2006
18. Zahn, C., Finke, M.: Collaborative knowledge building based on hyperlinked video. In: Wasson, B., Baggetun, R., Hoppe, U., Ludvigsen, S. (ed.): Proceedings of the International Conference on Computer Support for Collaborative Learning, Dordrecht, The Netherlands, pp 173–175. 2003
19. <http://www.itee.uq.edu.au/eresearch/projects/vannotea/>

