Journal of Universal Computer Science, vol. 17, no. 4 (2011), 605-617
submitted: 30/10/10, accepted: 15/2/11, appeared: 28/2/11 © J.UCS

Markup upon Video – towards Dynamic and Interactive Video Annotations

Peter Schultes
(University of Passau, Germany
schult16@stud.uni-passau.de)

Franz Lehner
(Business Administration II, University of Passau, Germany
franz.lehner@uni-passau.de)

Harald Kosch
(Distributed Information Systems, University of Passau, Germany
harald.kosch@uni-passau.de)

Abstract: Interactive video is increasingly becoming a more and more dominant feature of our media platforms. Especially due to the popular YouTube annotations framework, integrating graphical annotations in a video has become very fashionable these days. However, the current options are limited to a few graphical shapes for which the user can define as good as no dynamic behaviour. Despite the enormous demand for easy-creatable, interactive video there are no such advanced tools available.
In this article we describe an innovative approach, to realize dynamics and interactivity of video annotations. First we explain basic concepts of video-markup like the generic element model and visual descriptors. After that we introduce the event-tree model, which can be used to define event-handling in an interactive video formally as well as visually. By combining these basic concepts, we can give an effective tool to the video community for realizing interactive and dynamic video in a simple, intuitive and focused way.

Keywords: interactive video, event tree, video markup, hypervideo, video 2.0
Categories: H5.1, H.5.2, H5.4, D1.7

1 Motivation

In recent years, several projects came up that center on the topic of interactive video. By the means of Fraunhofer institute's "non-linear-video" project [Fokus, 10] interactive video is moving further into our living rooms. Spectators are able to interact with special objects in video by clicking on them or selecting additional material. Interactions generate new presentation layers filled with additional context information like text annotations, images or even embedded videos. The whole system is packed into a set-top-box and visualization is managed by a common television.

The ADIVI video-annotation software from [InnoT, 09] provides an integrated solution for merging video with additional resources. The main focus of ADIVI lies on documentations (e.g. technical) and enriched video reports. Here spectators are able to configure their own information level by interacting with the given hotspot objects and additional media.

Another project is currently accomplished at the University of Passau called “SIVA Suite” (from [Meixner, 09a] and [Meixner, 09b]). SIVA seizes the idea of YouTube annotations and presents an easy way for adding interactivity to an existing video. The supported features go far beyond the capabilities of the previous mentioned annotation environments: here the user can even branch the storyline or add interactive option dialogs to the underlying video.

Some research work is also currently done on combining interactive video with TV broadcasting: in their article *Watch-and-Comment as a Paradigm toward Ubiquitous Interactive Video Editing* R. Cattelan und C. Teixeira (c.f. [WAC, 08]) present a very interesting way on how interactive TV watching can be accomplished. The authors have already implemented a software tool called “WAC-tool” which runs on a handheld or tablet pc while the user watches a TV program. Spectators can add text or images into the video, skip scenes, chat with other users and share the new content with other members of a peer2peer group. The main focus of this work lies on collective TV watching inside a peer group, so here defining video annotations is already a form of communication.

The pioneer in user generated interactive video is the internet video portal YoutTube with its “YouTube annotations“-framework. In [YouTube, 09] you get a brief introduction on how video users can add graphical shapes to an existing video subsequently without having any further technological know how. These shapes are also applicable as hotspots and can be used for navigating the internal media library.

Especially the great success of YouTube annotations is an indication of the increasing demand for user generated interactive video. As we can see here, even the simplest capabilities – YouTube annotations only support three different, very simple graphical shapes – are used by the video community with growing enthusiasm. When browsing the video library we can find thousands of user annotated videos and, by the way, very artistical and creative ideas: an interactive piano[1], interactive games[2], videos which provide alternating endings[3] and much more.

That raises the question, why a simple method like YouTube annotations is so successful but comparatively powerful technologies like Adobe Flash, Java FX or Microsoft Silverlight are still rarely used for user generated video annotations. There are two reasons for this apparent: first of all these technologies require a relatively high skill level and technical knowledge, which cannot be accomplished by the typical video user. And second – which also constitutes a disadvantage of the tools introduced at the beginning – the user needs special software environments before he/she can start annotating his/her video, which may also lead to deterrent practicing efforts.

As we see from YouTube annotations, tools that do not require any technical skills and are very easy to use can open the way towards user-generated interactive video – or even video 2.0. In near future video will become a more and more dominant way of communication and user generated video annotations will be part of

[1] http://www.youtube.com/watch?v=oD-sSolVDiY

[2] http://www.10000words.net/2009/06/quick-guide-to-interactive-youtube.html

[3] http://www.youtube.com/watch?v=JFVkzYDNJqo

this “social” canal. So maybe video itself becomes a social media that people use to communicate, chat or create content in any way. One consequence of video as a social media would be, that the way people create content will change: text is no longer the only content type but also images, graphical forms or interactive elements. User generated content could more turn towards video games containing hidden information, interactive hotspots or dynamic content depending on the underlying video content.

However, the standardization efforts for user generated video annotations are still in early stages. Lots of problems are not solved until these days and because of the lack of experience with video as a form of communication even not identified. One of these technical issues is dynamic video annotations. In the approaches mentioned above, we simply can show and hide graphical annotations at certain media time points. The scenario “annotation ‘xy’ becomes visible, when the user clicks on hotspot ‘yz’”, for example, is not realizable via current online video annotation environments. Furthermore, interactive elements are acting statically and, for instance, are not able to change their appearance over the entire video time span.

In this article, we try to make complex dynamics and interactivity accessible for user generated video annotations. Here we will evolve standard methods for the authors of interactive video projects. These can be used to define complex interactivity screenplays by solely operating with graphical editors. The screenplay controls the look and behaviour of video annotations and supports standard mechanisms for handling user-, media time and annotation specific events.

2 Introduction into video-markup and prior work

In [Schultes, 10] we introduced the term *“video-markup”*, by which we mean any sort of graphical video annotations that follow a predefined interface. Video-markup does not change the underlying video material physically, but is maintained separately and synchronously interpreted at video runtime. Contrary to common markup languages like html or xml, video-markup is less of defining but more of annotating nature. In [Ram, 98] this type of markup is called a Procedural Markup Language (PML) contrary to the declarative approach (DML) used in HTML. Video-markup adds additional information, graphical shapes, audiovisual media or interactive elements to an existing video. The impact of video-markup for one thing is that the classical consumer-producer relationship gets broken up, because owner of a video can extend it with additional markup informations during a postproduction step (without modifying the underlying video material). Besides, interactive markup dismisses spectators from their passive roles and animates them to act. Viewers are now responsible themselves for behaviour, appearance and content of their extended video presentation.

In this work our aim is to develop a common procedure for defining a markup script without any programming or technological efforts for the author. Therefore we are currently working on a video markup system, which can be used to view and edit videos containing dynamic markup information. The system consists of three basic components: first, there is a media player which plays the actual video and provides default control functions (play, pause, seek…). Furthermore there are the markup elements which are presented above the video in a separate presentation layer. And finally we have an abstract markup screenplay, which holds all dynamic instructions on the markup elements (e.g. what happens in case of a button-selection and so on).

Before we explain, how dynamics and interactivity are handled in our video markup environment, we first take a look at the markup elements itself: In [Schultes, 10] we already worked out, that a generic approach is absolutely necessary for the definition of a semantic video markup model. This is the only way to ensure, that the markup environment can deal with any new element in the future, no matter how this element acts or what it stands for. Consequently the element palette of our markup system is not limited to a set of elements, but provides a mechanism on loading required markup elements on demand (through the use of plug-ins). We think that this function is less a feature but rather a requirement. Especially from the point of view that interactive video platforms are just coming up and we do not have sufficient experiences. We just cannot predict which markup elements a markup environment must have available in the future. So to reach acceptance by the user community an extensible system is absolutely necessary.

The generic approach is based on two similar technical concepts: first, each element fully describes itself, which meats that an element manages its appearance and behaviour itself. So, for example, not the environment paints an element, but the element paints itself on a given graphical context.

The other concept deals with following dilemma: the environment only has basic knowledge about an element and no detail information about its actual interface (so we can reach extensibility). But providing dynamics requires changing the state of an element from the outside and so accessing its interface is yet necessary. Our approach is based on the idea that element specific behaviour can be accessed via generic identifiers represented as string literals. We do so to abstract from a concrete implementation (in an element) towards a generic access method. So every markup element – a simple shape as well as an embedded video – can be seen as a black box which provides standardized methods for querying the graphical appearance as well as dynamically changing its current state. For that reason we identified three basic components that make up a generic makup element and can be used to access individual element state and behaviour:

- **Element-properties**
  Each markup element provides a common way, to obtain the value of a given property and to assign a new value to it.
- **Element-actions**
  Each markup element contains an access routine, which can be used to request the execution of a given action out of its repertoire.
- **Element-events**
  Each markup element must inform the markup environment of element specific changes of its state by sending events (which contain detailed informations about the occurred change). Furthermore each element provides a common way to evaluate user interface events (for interaction).

Our *"generic element model"* is basically an object orientated design pattern, which does not depend on a particular implementation. To illustrate a concrete implementation, figure 1 shows a summary of the generic element interface in the JAVA-programing language.

```
public void paint(GraphicalContext graphics);

public String[] getProperties();
public void setPropertyValue(String property, Object value);
public Object getPropertyValue(String property);

public String[] getActions();
public void executeAction(String action, Object[] params);

public String[] getEvents();
public void addEventListener(String event, IObserver observer);
public void handleUiEvent(Event event);
```

Figure 1: A simplified summary of the generic element interface (in JAVA)

The surrounding markup environment is not aware of the nature and impact of this element components, it only has knowledge on how to access them. As a result we achieve that an existing video markup system can deal with new elements by not knowing any further details of this element. On the other hand we are already able to formally describe a scenario, where elements dynamically change their internal state. We simply have to connect an event with a property-reassign (let’s call it an event-handler). An example scenario could possibly be: “after the spectator clicked on markup element ‘xy’ (element-event), element ‘yz’ changes its background colour into ‘blue’” (element-property). If we also consider typical environment components like media player, control panel or the embedding website to be instances of the generic element interface, we can define scenarios that exceed the actual video content: “at media time point 00:05:15 (element-event) the markup element ‘xy’ should get invisible (element-property) and the webbrowser (meta-element) should open the link ‘http://siva.uni-passau.de’ (element-action)”. Figure 2 shows a formal description of this scenario in xml notation, which can be used as a markup screenplay implementation:

```
<eventHandler eventSource="mediaPlayer" eventType="mediaTime">
    <detail name="timePoint" value="00:05:15" />
    <actionList>
        <property target="xy" property="visibiliy" value="false"/>
        <action target="browser" action="openLink">
            <Detail name="param1" value="http://siva.uni-passau.de"/>
        </action>
    </actionList>
</eventHandler>
```

Figure 2: A possible formal description for the example scenario

3 Element Descriptors

As we can see in figure 2, the presented concepts are already sufficient for formal description of instructions contained in a generic markup screenplay. But this leads us to the question on what is the advantage of an xml-based script instead of a Flash or Silverlight action script? It would certainly make no difference to the video user because both ways do not seem very vivid to him (without any further technological understandings). The main difference consists in the fact that our generic approach is a domain specific semantic model that opens extensibility and especially visual editing capabilities – which is a great advantage when designing an authoring method.

One of the main usability challenges of our authoring tool is to provide an intuitive way on how the required components (properties, actions and events) can be specified by the user. But as we have seen so far event-handlers are merely defined in a generic way. So an element simply provides information on which properties it supports but not about the actual meaning of its (eventual) property ‘background’ or ‘bg’ or ‘backCol’ etc. Authors are not able to obtain detailed information about the effects and impacts of a generic property label. The markup environment does not care of this at all, because the environment only has to ensure that a new value is correctly assigned.

As mentioned above, the environment actually does not know anything about the particular element. But it would be a great advantage for the author of an interactive video, if the annotation tool could inform him about the features of the current markup element in readable form. For that reason we now introduce another concept which we call *“generic element descriptors”*. A descriptor is a meta object which describes its underlying element component (property, action or event) by specifying:

- Label
- Description
- Icon / preview image
- Component details

Component details differ from the particular component type:

Properties	➡	Value type, value scope, etc.
Actions:	➡	quantity, type and descriptions of potential parameters
Events:	➡	description of the supported event details (e.g. mouse button, mouse pointer location, media time, …)

Descriptors are managed by a central registry and can be obtained from there by an editor specifying the element component. The presence of descriptors makes the editing process of video-markup much easier for the author. Especially the visual editing capabilities of component descriptors lead to an enormous increase of productivity (see [Horton, 94]). Figure 3 compares two potential editing tools: the left box ignores and the right box interprets the given descriptors. Another advantage of component descriptors lies in the fact, that they can be appended to an existing element implementation afterwards, because of their declarative nature.

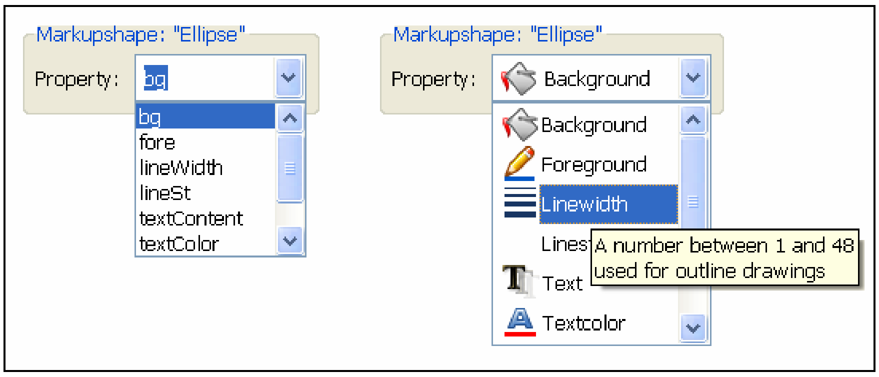


Figure 3: Two possible tools for specifying element properties visually. Left: descriptors are ignored, right: descriptors are interpreted

Component details provided by a descriptor are very useful during editing process: in case of assigning a new value to an element property it would be pretty convenient, if the author had information about possible valid values at editing time. It would not make sense, if the editing tool allowed specifying a string for the background property of a markup shape. But if the tool already knew about the requirement of a RGB-value through a descriptor, it could offer an appropriate editing method. By interpreting the descriptor we can reduce invalid user inputs and support a faster and more intuitive working process. And as before, the usage of component descriptors still supports genericity, what ensures a maximum of flexibility and extensibility to our markup environment.

As a summary of the results made so far, the following list describes the interaction of all introduced concepts by explaining the process of inserting a new video annotation shape in a video and changing its background colour:

1. Editor palette lists up all markup elements, which are currently plugged in the video markup system.
2. User selects a markup shape and inserts it into the video at a specific media-time point.
3. Markup environment instructs shape to paint itself.
4. Editor queries all properties from the new element.
5. Editor queries all related property-descriptors from the registry.
6. Editor shows the descriptor labels as well as the current values in a table. The current values are queried from the element by using the generic element interface.
7. User double-clicks the “background-colour” row; editor shows a colour-choose dialog in response to the property-type defined in the descriptor.
8. User selects a new value.
9. Editor calls setPropertyValue(...) of the generic element interface and updates visually.

4 Event tree model

By now simple instructions and event-handle routines can be modelled formally as well as visually. In the next step we examine how more complex processes in the markup layer can be defined by the author of an interactive video.

Our markup screenplay consists of all actions, which will be executed at video runtime in response to certain events. In the context of video markup, an event is either a media time event occurring in the abstract media player element, a user event (e.g. through input devices) or an element event created by the markup elements themselves (such as "option selected" in a multiple choice element). A markup screenplay action is simply a container filled with arbitrary instructions on markup elements – for example reassigning element properties or executing element actions.

The problem with simple event-handler routines (which we mentioned in section 2) is that authors cannot specify conditional evaluations. For instance the event "option selected" from a multiple choice element could eventually evoke two different actions: either "signal ok" or "signal incorrect" depending on its current selection. So after receiving the target event, the markup environment should compute the resulting action by evaluating the selection-index (= element property!) of the multiple choice element. That leads us to another concept in our semantic video markup model: "*element-conditions*". A condition therefore is basically a couple of property-value pairs, which have to be verified at runtime. In detail a condition consist of four parts:

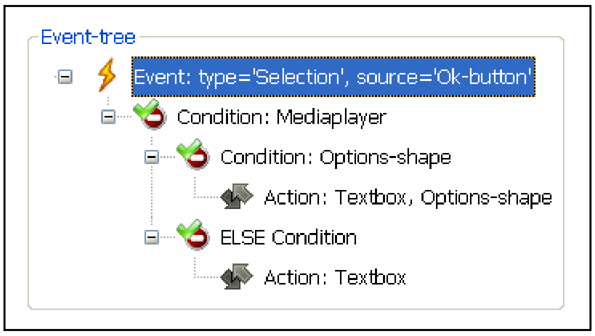
- target element
- target property
- value which must be verified
- operator used for verification (<,=,>, <>..)

As a special case the "ELSE"-condition must also be mentioned, which evaluates to true if all other conditions on the same level are not fulfilled.

Since conditions can be nested hierarchically, a tree structure seems to be suitable as an intuitive visualization: the root elements represent the events containing possible restrictions to event details (e.g. event source, event type, current media time point…). The next levels consist of zero or more nested conditions and finally, at the leaves-level, all actions are stored. However a tree structure also qualifies itself as an appropriate semantic model by making use of nested data items (as we can also see using the example of the Document Object Model). The question here is why shouldn't we use directed graphs instead of trees to avoid the duplication of paths? For example, if two different events evaluated to the same action we could insert a cross-edge in case of a directed graph and reuse the action. A tree structure, in contrast, does not allow us to specify multiple parent-edges, so duplication of the action-node is necessary. But the main benefit of the tree model over a graph lies again in its visible editing capability. Our application can make use of standard tree widgets, which likely everybody has already interacted with. Directed graphs require a more advanced visualization to clarify their edge structure. Since usability is possibly our principle claim, we decided to build our data model and visual design concept on a tree structure.

The following scenario demonstrates the usage of an event tree in combination with the generic video-markup interface. Figure 4 therefore illustrates the appropriate event tree with a standard tree widget and figure 5 shows the related formal description following the xml syntax introduced in figure 2:

The author of an interactive video wants to end his video with a verification question. Therefore he uses a multiple choice shape (containing a question field), some option fields, a (submission) button and a feedback text shape. After the button is selected by the user, the text shape shows feedback information, depending on the selected index in the question shape (let's say index one is the correct answer). As an additional constraint he determines that the evaluation can only be done after media time point 00:05:00. In case of a correct answer the feedback information is "correct!" and the multiple choice element has to dispose. Otherwise "wrong answer" is shown.



```
Event-tree
⊟ Event: type='Selection', source='Ok-button'
    ⊟ Condition: Mediaplayer
        ⊟ Condition: Options-shape
            Action: Textbox, Options-shape
        ⊟ ELSE Condition
            Action: Textbox
```

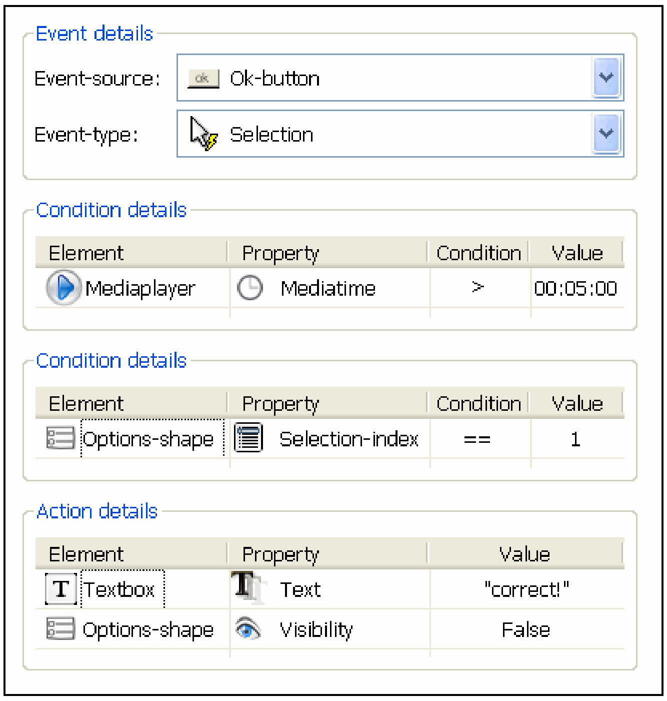
Figure 4: Visual representation of the event-tree taken from the example-scenario

```
<event source="Ok-button" eventType="selection">
  <condition target="mediaplayer" property="mediatime"
             operator=">" value="00:05:00">
    <condition target="Options-shape" property="selectionIndex"
               operator="=" value="1">
      <actionList>
        <property target="Textbox" property="text" value="correct!"/>
        <property target="Textbox" property="visibility" value="false"/>
      </actionList>
    </condition>
    <elseCondition>
      <actionList>
        <property target="Textbox" property="text" value="wrong answer"/>
      </actionList>
    </elseCondition>
  </condition>
</event>
```

Figure 5: Formal description of the example-scenario in xml notation

Furthermore, our editing tool – which is currently in realization stage – supports detail editors for specifying events, conditions and actions contained in the event tree.

These editors are implemented by the use of tables and arranged around the event tree pursuant to *Master-Detail*-screen pattern explained in [Scott, 09]. First experiments showed us, that tables are a very convenient and easy to use way for specifying conditions and actions based on element properties. Descriptors (see section 3) can help to equip the tables with labels, icons and descriptions to ensure a user orientated editing process. A reliable way to select reference elements, properties or compare operators is the usage of combo-boxes (as shown before in figure 2). Specifying explicit values (in case of property assigning/constraining) must be done by suitable “in-place”-editors, which depend on the value type of the selected property: for instance a text-editor should come up in case of string-values and the operating system’s colour-choose-dialog in case of RGB-values and so on. Example-editors for event-, condition- and action-details related to the example scenario are shown in figure 6.



Event details

Event-source:	Ok-button
Event-type:	Selection

Condition details

Element	Property	Condition	Value
Mediaplayer	Mediatime	>	00:05:00

Condition details

Element	Property	Condition	Value
Options-shape	Selection-index	==	1

Action details

Element	Property	Value
Textbox	Text	"correct!"
Options-shape	Visibility	False

Figure 6: Detail editors for event, action and conditions related to the given example. The displayed action relates to the condition-path in figure 4 (not the “ELSE”-path)

5 Known problems

When users define content upon a video, this content strongly relies on the video content. Therefore, in most cases the presentation behaviour of video annotations is controlled by the media time. Let’s consider a video contains some regions of interest, which are tracked and marked as hotspots. So these hotspots must act synchronous with the video (-objects) and update their locations when media time changes – for one thing if the video is playing and for another thing if the spectator jumps back and forth in media time. We can basically say that there is one kind of actions group which is evaluated in media time scope and correlates with the video content. But in the case of event triggered actions - like user interactivity – the time scope is not clear at all. Let's look at following example (c.f. figure 7): we have an annotated soccer game where the players are marked as hotspots and dis-/appear at certain media time points. Furthermore, the video contains a permanent textbox that shows details about a player after the spectator has clicked the particular hotspot. So the hotspots move and dis-/appear in media time scope, while updates of the textbox take place at presentation time. Let's say at media time point 00:00:10 the spectator clicks on a marked player; thus the textbox shows some information text like *"21: Phillip Lahm"*. At media time point 00:00:20 the spectator clicks on the referee and textbox shows *"--: Referee"*. After this action, the spectator decides to jump back at media time point 00:00:12 to watch the last scene again.

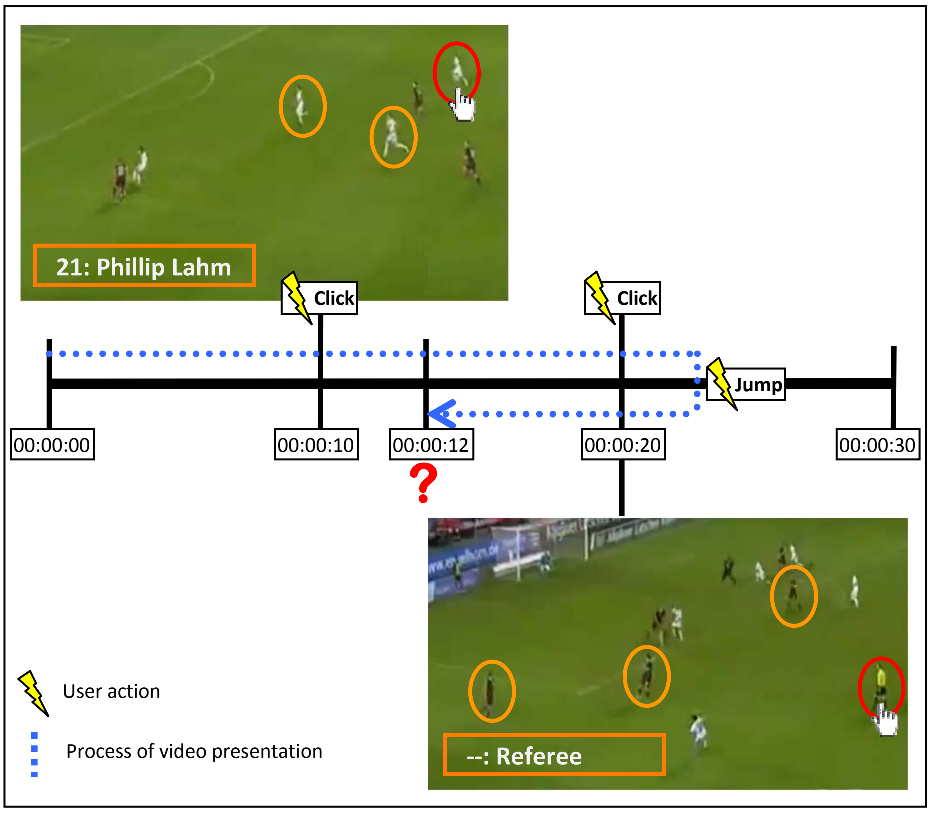


Figure 7: Illustration of the example scenario: media time vs. presentation time

Now how should the system handle this request? First of all, the hotspots must update their location and visibility state appropriate to the target media time point.

But which information must the textbox display? Should the system roll back the last action and display “Phillip Lahm” again because the new media time point lies before the execution media time point of the “referee”-action? Or should the system keep the current information in the textbox whether media time changes or not because the “last” clicked hotspot is still the referee?

In general, the system environment cannot answer this question without having any further knowledge. Our prototype implementation currently does not roll back actions that were triggered by user events (after a media time jump back occurred). So we strictly separate media and presentation time context here. But we get into troubles when presentation time actions strongly rely on the video content - like in the example scenario (e.g. actions triggered by a click on an object in the video). Because the impact of keeping the textbox information is that the textbox shows information about a hotspot (or video object respectively) which was not yet visible – regarding the current media time point. So from a semantic viewpoint, this circumstance leads to an invalid state of markup-media-interaction.

We think that the only solution for this problem is to give authors the ability to specify whether an action is media time related or not. It should easily be possible to integrate a checkbox into the current workflow that specifies the type of action. But probably users who are not quite familiar with the annotation software won’t use this functionality because they do not know the background and impacts of this setting.

6 Conclusions and future work

Our studies on increasing interactivity and dynamics in user generated non linear video showed us two main aspects: extensibility and a user friendly control concept are of inherent importance to reach acceptance by a potential user community. To achieve this we are currently working on a video annotation system that supports pluggable video markup elements and a graphical authoring environment seizing the idea of YouTube annotations. The aim of this work was to present our main concepts on how we meet the given requirements in our annotation system. The main realization key points therefore are the generic element model, element descriptors and the event tree model, which we introduced in this article. By combining these techniques we are able to implement a video annotation system that fulfils our aims on extensibility, usability and technical features. Currently, our system supports a general media player and several video markup elements. A well defined interactivity screenplay model is also already developed and gets interpreted by the annotation system at start up.

Our next steps include the completion of the authoring tool design, which makes extensive use of the concepts presented in this work. At this point, we also have to consider how moving regions in a video can be modelled and visually prepared for user editing. Since hotspots are a major feature of interactive video, moving elements will be a central aspect in further considerations. Technical questions on how we can make our system available to as many users will also be part of future work.

References

[Fokus, 10] Fraunhofer Fokus: Non Linear Video technology which enables interactive content on TV, PC and Mobile Phones, 2010, http://www.fokus.fraunhofer.de/en/fame/_pdf/FOKUS_non-linearvideo_en.pdf

[Horton, 94] Horton, W.: The Icon book: Visual symbols for computer systems and documentation. Toronto: John Wiley & Sons, 1994

[InnoT, 09] InnoTeamS Global: ADIVI Instructional Guide, Version 1.0, 2009, http://www.adivi.net/Materialien/ADIVI_manual.pdf

[Meixner, 09a] Meixner, B., Siegel, B., Hölbling, G., Kosch, H., Lehner, F.: SIVA Producer - A Modular Authoring System for Interactive Videos. In Proceedings of I-KNOW, 9th International Conference on Knowledge Management and Knowledge Technologies, p. 215-225, Graz, 2009

[Meixner, 09b] Meixner, B., Siegel, B., Hölbling, G., Kosch, H., Lehner, F.: SIVA Suite - Konzeption eines Frameworks zur Erstellung von interaktiven Videos. In Eibl,M. et al (Hrsg.): Workshop Audiovisuelle Medien WAM 2009, p. 13-20. Chemnitz, 2009

[Ram, 98] Ram, A., Catrambone, R., Guzdial, M.J., Kehoe, C.M., McCrickard, D.S., Stasko, J.T. - PML: Representing Procedural Domains for Multimedia Presentations. In College of Computing, Georgia Institute of Technology, Technical Reports 1998

[Schultes, 10] Schultes, P., Lehner, F., Kosch, H.: Videomarkup - Vom Videonutzer zum Produzent. In Eibl,M. et al (Hrsg.): Workshop Audiovisuelle Medien WAM 2010. Chemnitz, 2010

[Scott, 09] Scott B., Neil T.: Designinig Web Interfaces: Principles and Patterns for Rich Interactions. Sebastopol: O'Reilly Media, 2009

[WAC, 08] Catteln R.G., Teixeria C., Goularte R., Da Graca M., Pimentel C.: Watch-and-comment as a paradigm toward ubiquitous interactive video editing. In ACM Transactions on Multimedia Computing, Communications, and Applications, 2008

[YouTube, 09] YouTube: Erste Schritte: Erstellen oder Bearbeiten von Anmerkungen, 2009, http://www.google.com/support/youtube/bin/answer.py?answer=92710