Modelling Interaction with HyTime

Stefan Wirag, Kurt Rothermel, Thomas Wahl

University of Stuttgart
IPVR
Breitwiesenstr. 20 - 22
D-70565 Stuttgart
wirag@informatik.uni-stuttgart.de

Abstract

Interactive multimedia presentations are an essential issue in many advanced multimedia application tools. Before presenting multimedia data, media items, interaction types and synchronization constraints have to be specified in a multimedia document. This paper identifies and classifies the temporal interaction types in multimedia systems, and shows their impact on the specification process and the supporting system. Then, we describe how to specify the interaction types by using the standardized multimedia document language HyTime. The HyTime mechanisms are demonstrated by examples followed by a discussion of the advantages and limits of each technique.

1 Introduction

With the emerging multimedia technologies, more and more application tools are developed to process and present multimedia data. Generally, multimedia data are stored as multimedia or hypermedia documents using a proprietary format [Appl91], [BuZe93], [BHL91], [LiGh90]. The result is that tools of different vendors or developer groups cannot exchange their documents without a format conversion. Converting document formats is expensive if possible at all. A general document standard for multimedia would alleviate this problem.

HyTime [HyTi92] defines a standardized language for specifying the essentials of multimedia documents, such as addressing documents or defining temporal constraints. When developing our multimedia presentation system TIEMPO[1] [WaRo94], we considered HyTime as a document model. The architecture of HyTime is described by [Gold91] and [NKN91]. Further, [Erfl94]

[1]TIEMPO: grant of the Deutsche Forschungsgemeinschaft DFG
Temporal **i**nt**e**grated **m**odel to **p**resent multimedia-**o**bjects

examined how to specify synchronization constraints in HyTime but he excludes the question how to specify interaction. As multimedia application tools increasingly support interaction, this question becomes a crucial issue in multimedia documents. Specifically, those interaction types that affect the synchronization constraints are critical because the predefined presentation schedules specified in HyTime might be modified in case of an interaction.

Although HyTime does not provide any direct mechanisms to express interaction, there are generic mechanisms that can be used for modelling interaction since HyTime is an encompassing standard. Therefore, this paper examines the mechanisms of HyTime, shows how interaction might be expressed and describes specification techniques that exceed the approaches of [KRRK93] and [BRR94].

In section 2, we identify the interaction types affecting the HyTime schedules, and describe the issues for the specification and the system support of the interaction types. Section 3 introduces the essentials of HyTime. HyTime mechanisms to support interaction are presented and discussed in section 4 followed by section 5 describing how to model the interaction types in HyTime. Finally, we summarize the results.

2 Interaction

Various methods exist to interact with a system during a multimedia presentation. A user might resize a presentation window or control the volume of an audio channel. Some of the interaction types affect the temporal layout of a multimedia presentation. E.g. pushing the pause-button delays future events, or with the fast-forward-button future events occur earlier than originally scheduled. Interaction types that affect the temporal layout of a presentation are critical because most multimedia presentations include a schedule of all events within the presentation. Thus, interaction might result in several changes of the presentation schedule because events have to be rescheduled, new events are added or other events are no longer valid and have to be removed from the schedule.

In this section, we identify the interaction types with a temporal impact. Then, the issues of specifying and executing multimedia presentations are discussed in respect to the interaction types.

2.1 Interaction types

Table 1 summarizes the interaction types with a temporal impact on multimedia presentations. The interaction types are described by their name, symbol and their impact on the presentation speed. The symbol represents the default presentation trace by a grey arrow and the modification of the trace by a black arrow. Multimedia applications can be classified according to the interaction types that are offered when presenting documents. The fourth column of table 1 indicates which interaction type is included in which class. Hence, the lower class interaction types might be included in the higher classes.

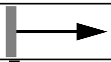
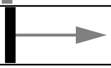
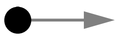
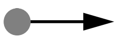
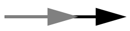
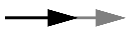
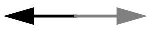
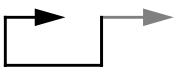
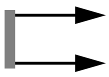
interaction type	symbol	presentation speed	class
start		$v_{new} = v_{default}$	basic
stop		not defined	linear directed
pause		$v_{new} = 0$	linear directed
continue		$v_{new} = v_{before\_pause}$	linear directed
faster		$v_{new} = sign(v_{before})* \|v_{before}\| ++$	linear directed
slower		$v_{new} = sign(v_{before})*\|v_{before}\| --$	linear directed
reverse		$v_{new} = -v_{before}$	linear undirected
jump		$v_{new} = v_{before}$	linear undirected
selection		$v_{new} = v_{default}$	non-linear

Table 1: Classification of interaction types

A first class of multimedia presentation systems does not provide any interaction during the presentation. But still a start-mechanism is needed to produce any perceivable output. Therefore, *start* is an essential interaction type of any presentation system. In a second class, the presentation speed can be varied by interaction types such as *faster*, *slower pause*, *continue* or *stop*. However, the direction of the traversal through the multimedia document remains forward during the entire presentation. For this reason, this class of systems is called *linear directed*. More advanced presentation systems also allow to reverse the presentation direction or to jump to another part within the document. So, the sequence of the presented events is no longer predefined. But still the default presentation of a document is linear, i.e. all events are totally ordered. Therefore, this class is called *linear undirected*. The most comprehensive class of systems additionally provides the selection-interaction, by which the next media item can be chosen from a list of items. The path through a multimedia document with selection-interaction is no longer predefined. A variety of paths are possible. So, the selection is a *non-linear* interaction type.

2.2 Specification of interactive documents and system support

The basic class of presentations without any interaction except the start-command can be specified by using real time synchronization constraints since all events except the start are pre-known and predictable. Also once the presentation is started, the supporting system can meet all synchronization constraints by prefetching all necessary presentation data.

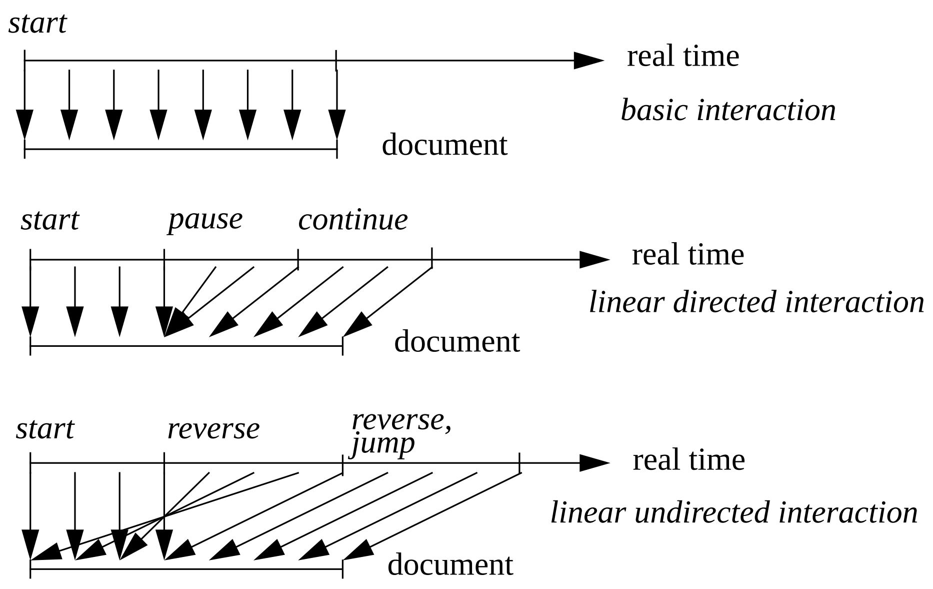


Figure 1: Mapping logical time to real time

The non-basic classes of presentations are specifiable by real time constraints because the duration of a presentation segment might vary due to interaction such as pause slower, faster, etc. Therefore, the concept of virtual [HyTi92] or logical time [Lamp78], [AnHo91], [RoHe94] was introduced. The presentation data is considered as a totally ordered sequence of information units. Then, the logical time is defined by the sequence of information units. A logical time unit can be given in frames, samples, bits, bytes or simply an abstract unit. Now, synchronization constraints are specified in terms of logical time units. However before rendering such a document, its logical time has to be mapped to real time. Figure 1 shows how the mapping is done in the basic, the linear directed and the linear undirected class. Each point in real time is assigned the multimedia data that is rendered at that time.

Executing a presentation of the linear directed class can be implemented fairly easily as the supporting system knows at any time what presentation data might be rendered next. This holds because the rendering direction of a linear directed document is always forward.

Implementing the linear undirected class is more sophisticated because several presentation data units might be rendered next depending on the interaction events that occur. In case of the reverse interaction, the system might present either the last data unit or the next data unit depending on whether the reverse-button was pressed or not. In case of a jump, the number of possible data units to be rendered next is theoretically infinite.

The most complex class, a non-linear presentation, cannot be specified on a single logical time line as it is not known which selection will be chosen by a user. E.g. figure 2 shows a scenario in which a first talk is followed by a selected video and then by a second talk. Depending on the duration of the selected video, the sequence numbers of the logical units of the second talk are different. So, there is not a unique document time. Then, synchronization constraints cannot be aligned on a single logical time axis. Implementing selection, prefetching of presentation data by the supporting system is not trivial because several data units are in question to be presented next depending on the number of options offered by the selection. This can vary from a few to a theoretically infinite number of choices.

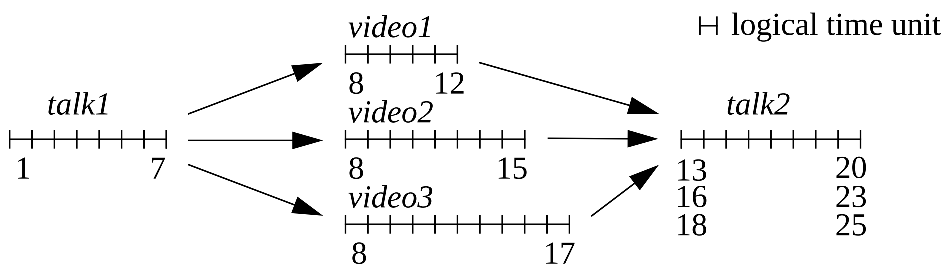


Figure 2: Non-linear interaction

3 HyTime

Any platform for hypermedia applications might have its own proprietary method of representing documents. Thus, it is difficult or even impossible to interchange documents created by different applications. Therefore, HyTime (Hypermedia/Time-based Structuring Language) was developed as an international standard [HyTi92] for structured representation of hypermedia documents for integrated open hypermedia applications. It is an SGML (Standardized Generalized Markup Language) application and is interchanged using ASN.1 for OSI-compatibility.

A hypermedia document is a set of documents and other information objects connected by links. When the definition of a document type is created, content and rendering instructions are distinguished. HyTime standardizes those facilities dealing with the addressing of portions of hypermedia documents and their component multimedia information objects including the linking, alignment and synchronization of document items. HyTime does not standardize the data content notation, the encoding of the information objects or the application processing them. The HyTime standard does not impose any particular implementation architecture, and it is possible to integrate HyTime-processing in application programs if desired. The HyTime architecture is modular and only the required facilities need to be implemented. The HyTime standard consists of the following modules: The *base module* specifies the basic issues. The *location*

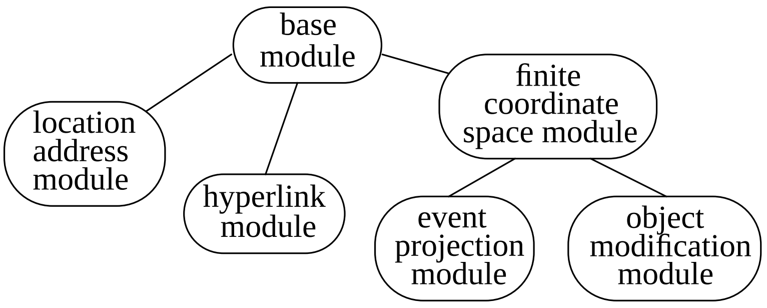


Figure 3: HyTime modules [NKN91]

address module specifies the addressing facilities. The *hyperlink module* specifies the hyperlink facilities and the *finite coordinate space module* deals with the position of objects in space and time and their modification. Figure 3 shows the relations of the modules.

A brief description of the HyTime features that are useful for the specification of interaction is given in the following sections.

3.1 Document structure

The architecture of an SGML document is expressed in its Document Type Definition (DTD). The syntax is expressed as a set of elements, each with its own generic identifier, a set of attributes and the content model, which determines the data types to be used in the element. The HyTime standard defines element types called *architectural forms* (AF) identifiable by the attribute HyTime. By including the attribute HyTime and conforming to the model of a particular HyTime architectural form, document authors can create derived element types with specific semantics. Additionally, attributes can be inserted containing information according to the semantics. Using AF's and derived element types, document authors can create DTD's which incorporate only those semantics of HyTime that are needed.

3.2 Control flow

HyTime documents are interpreted by a HyTime-engine. If a HyTime document is ready to be processed, the application calls the HyTime-engine which in turn calls the SGML-parser. The parser notifies the HyTime-engine about anything important. The HyTime-engine performs address resolution, linking, alignment and synchronization and passes the entire output of the document back to the application controlling the presentation. The flow through a hyperdocument is controlled by an application program and can be modified by scripts that are embedded within a hyperdocument. The application calls programs which interpret the scripts.

3.3 Temporal relations

In HyTime, an object is a piece of information of any type. An object may consist of data such as video, audio, graphical objects or text. To position objects in space and time, HyTime uses a *finite coordinate space* (FCS). A FCS is described by *axes*. Any FCS establishes a specific measurement domain with a reference unit defined for each axis. Objects in a HyTime finite coordinate space occur as the content of *events*. An event is a conceptual frame for an object. Each event has a dimension specification that represents its position and extent on the coordinate axes of the FCS. Elements of the type *dimref* allow to position events dependent on other events. An application might associate synchronization constraints with these relations. If the determination of the dimension specification of events requires complex computations *marker functions* can be applied. Such a function computes the position or extent of an event on one axis. Events are organized in *event schedules*. A FCS may contain any number of event schedules, and each event schedule may contain any number of events.

The following example contains the temporal specification of a scenario where an event (event 2) starts 10 seconds after another event (event 1) has started. Further, event 2 has the same length as event 1. We use a *HyFunk*-element to specify the relative positioning of event 2. *HyFunk* is a HyTime-defined marker function type that can be used to express simple relations between event extents. The first three lines of the example define this function. The elements %1, %2 in the definition represent parameter that are passed on to the function when it is called. Then, the extent lists which define the position and extent of the events in the FCS are specified. The position and extent of event 1 are directly specified. In the specification of the extent list of event 2 the defined marker function is applied to position the event relative to event 1. The reference to the extent list of event 1 and the delay are the parameter of the function call. The duration of event 2 is specified by a *dimref*-element which extracts the duration of event 1. Finally, the finite coordinate space with the event schedule (evsched) containing the two events is specified. (Further examples are found in [Erfl94].)

```
<!-- excerpt from a document instance -->

<HyFunk fn=”cal_start”>   <!-- function definition -->
   @sum(@first(%1) %2)
<\HyFunk>

<extlist id=”ext_ev1”>    <!-- extent list of event 1 -->
   <dimspec id=”dim_ev1”>
      30 210


<extlist id=”ext_ev2”>    <!-- extent list of event 2 -->
   <dimspec id=”dim_ev2”>
      <HyFunk usefn=”cal_start” args=”dim_ev1 10”
      <dimref elemref=”dim_ev1” selcomp=”qcnt”>

<fcs>         <!-- finite coordinate space -->
   <evsched>
      <event exspec=”ext_ev1”> <!-- event1 -->
      <event exspec=”ext_ev2”> <!-- event2 -->
```

The event projection module of HyTime provides the facility to project events from one FCS to another FCS. Thus, event projection might be used to extract a specific part of an object or to modify the presentation speed of an object. The projection is performed by a projector which can be defined in a notation unknown to the HyTime-engine. In this case, the HyTime-engine asks the application to determine the location and extent of the projected events. Simple projection types can be expressed applying marker functions, such as the projection by a constant ratio. Projectors are organized in schedules called *baton.* A *batrul*e-element must be used to express the relation of a baton, unprojected event schedules, and projected event schedules. All event parts in the related unprojected schedules that are within the specified projector-scope of the projector are published to the projected event schedules. Additionally, projected event schedules can contain events which are not derived from projections.

3.4 Links

The hyperlink module of HyTime provides various link types. Links can be used to describe relations between any kind of objects. HyTime knows two major link types: *Contextual links* (clinks) describe a relation between two objects. One link-end is the content of the link element and the other link-end is an arbitrary object. *Independent links* (ilinks) represent a general form of a link. It can have any number of link-ends. With *independent links*, roles can be defined assigning semantics to anchors.

4 HyTime techniques to specify interaction

Documents with interaction abilities contain temporal relations which have to be resolved during rendition. HyTime gives little support to specify relations which cannot be bound to well-known time points. Thus if interaction should be integrated, HyTime extensions are needed. In this section, we describe some approaches to integrate interaction in HyTime.

4.1 Schedule-link approach

In HyTime, events which describe the occurrence of objects in an abstract manner are organized in schedules. Such a schedule determines the presentation for a temporal interval. This interval is normally determined by the start instant of the first event and the end instant of the last event in the schedule. In documents with interaction facilities, multiple alternative renditions are possible. In HyTime, such alternatives can be specified applying a schedule for each particular rendition. Thus each time an interaction occurs, the rendition of the current schedule is aborted, and the rendition continues in the schedule that represents the interaction effect. Enabling interaction in HyTime and a mechanism to switch schedules on interaction are prerequisites of this approach.

HyTime does not deal with user input like mouse clicks or key pressings. Nevertheless, possibilities to interact must be offered to the user, such as buttons, keys or slide-bars. Input facilities which have to be displayed on the screen might be specified as HyTime events. We call such events interactive events. Element types for interactive events with a defined semantics can be derived from the AF *event*. An element type of a simple label-button event might be:

```
<!element button - 0 empty>
<!attlist button
      label       CDATA       #REQUIRED
      linkends    IDREFS      #REQUIRED
      HyTime      NAME        #FIXED event
      id          ID          #IMPLIED
      -- exspec attributes --
>
```

The *label*-attribute determines the text that appears within the button. The *linkends*-attribute is needed to express relations to following event-schedules. The *exspec*-attributes describe the position and extent of the button event. The *HyTime*-attribute specifies that this element type is derived from the AF event. The *id*-attribute identifies an element of this type. The element type has no content.

Hyperlinks are used to define an action which has to be performed if the user applies an interactive event. These hyperlinks relate an interactive element with a schedule that contains the interaction affect. For this purpose, a link element type with special semantics might be derived from the HyTime *clink*-AF:

```
<!element linkbut - 0 empty >
<!attlist linkbut
    trigger         CDATA           #REQUIRED
    HyTime          NAME            #FIXED clink
    id              ID              #IMPLIED
    linkend         IDREF           #REQUIRED
>
```

In the example, the attribute *trigger* defines the condition which must become true on the interactive event so that the application traverses the link and continues processing in the referenced event-schedule. Defining links with different trigger conditions relating different schedules, multiple user interaction can be defined with the same interactive event.

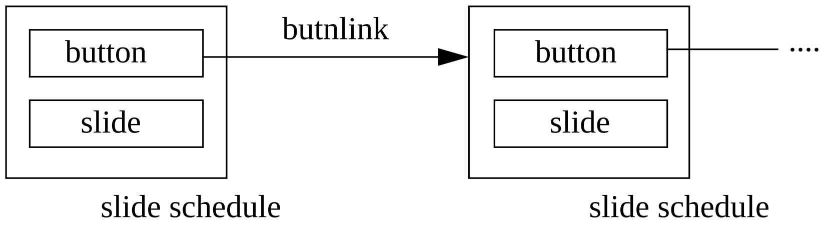


Figure 4: Generic interaction example

In [KRRK93], a DTD for a slide show is described allowing to move to the next slide interactively. In this DTD, links have a similar semantics as the link defined above. Figure 4 shows the link connections of the example. Each slide schedule contains a button event and a slide event. The button event is linked to the subsequent schedule. The link is traversed to find the following slide schedule if the button is pressed.

Generally, all interaction types introduced in section 2 can be specified by links. For complex interaction forms such as faster and reverse, additional information is needed to position the events. This knowledge must be present in the application if the rendition module is not used. Further, the definition of alternative renditions by different schedules is not applicable with infinite interaction effects. For example, if the presentation speed of a media item can be manipulated by a slider, any speed within a certain range is acceptable. The specification of such a behavior requires additional mechanisms.

4.2 Integration of scripting languages

Interaction requires additional processing descriptions within HyTime. Therefore, scripting languages such as HyperTalk might be integrated to describe actions to be executed on interaction. It is possible to define element types for scripts which can be added to any DTD by creating new document elements in the appropriate places [BRR94]. Such script elements are treated as media objects which cannot be interpreted by the HyTime-engine, and therefore would be passed on to the processing application for interpretation. The following example [BRR94] shows an element type *page* which might contain script objects:

```
<!element page - 0 ( graphics*, buttons*, script* ) >
<!element script - 0 CDATA >
<!attlist script
    HyScript        NAME     #FIXED script
    script_type     CDATA    #REQUIRED “HyperTalk”
>
```

The attribute *HyScript* expresses that the AF is not a HyTime AF. The attribute *script_type* identifies the scripting language. Therefore, multiple scripting languages can be integrated. The content type of the AF is the script and is not parsed by the HyTime-engine.

Generally, all interaction affects can be defined using scripts. For example, the application can maintain and control temporal relations apart from the HyTime-engine by including the temporal information within scripts. However, this may lead to consistency problems because HyTime also provides a mechanism to specify synchronization constraints.

4.3 Projection approach

In the existing examples, only simple interaction types are considered, e.g. start and stop. Our goal is to extent the existing approaches to be able to specify the interaction forms introduced in section 2. We developed a method to specify interaction using as many facilities of HyTime as possible.

Analyzing the effect of the interaction types introduced in section 2 in respect to their specification, the following is observed: The basic interaction *start*, the linear directed interaction *stop* and the non-linear interaction *selection* change the object set currently presented. For new objects appearing as the result of an interaction or the remainder of objects which is rendered different as the result of the interaction events have to be specified that are positioned at the current rendition instant of the time axis. With interaction affecting events representing continuous media items the context of the media item has to be preserved.

These effects require that interactive documents contain temporal relations which are resolved during rendition. Therefore, context information is necessary to relate the events to the remaining or new objects. Context information is the current rendition point on the time-axis of a FCS. A derived element type of the AF *evsched* can be created that causes the application to collect the context information. The application stores the information internally.

The projection facility of HyTime is used to specify the necessary mapping of logical time to real time when positioning events after interactions. The effect of projectors can be described using marker functions. Therefore, a derived marker function type has to be defined which contains scripts that define how to apply the collected context information. To compute the value of a marker represented by such a marker function, the HyTime-engine calls the application. Thus, it is possible to use the context information during the execution of the marker function. References identifying context information are passed on as input arguments to marker functions. To apply this method, late computation of event extents and schedules must be given because the needed context information is only available during the rendition.

To demonstrate the method we present an example. Figure 5 shows a scenario with a continuous object where it is possible to skip a part of the object by pressing a button. Because the interaction time point is not known before run-time, the position of the event presenting the remainder of the object is context-sensitive. A document instance may contain the following lines to describe the situation:

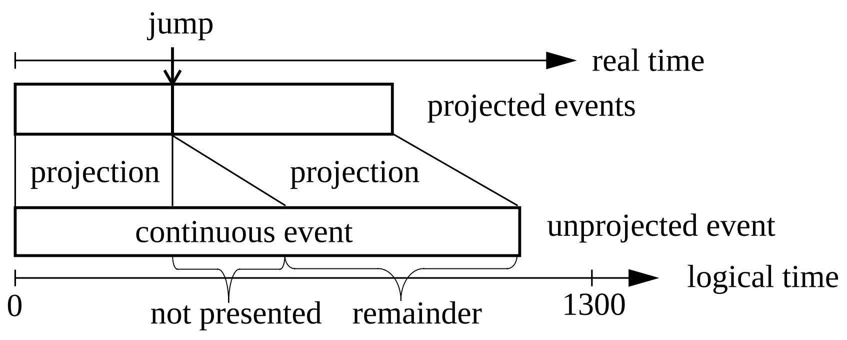


Figure 5: Projection with jump interaction

```
<!-- derived marker function returning the current position -->
<ConFunk fn=”con_last”>
   -- return (current position on the axis %1) + %2 --
<\ConFunk>

<!-- extent lists of the unprojected event -->
<extlist id=”event_1”>
   <dimspec id=”devent_1”>
      0 -100
<!-- extent list of the button event-->
<extlist id=”butt_ev”>
   <dimspec id=”dbut_ev”>
      <dimref elemref=”video” projectr=”prj_11” selcomp=”first”>
      <HyOp name=”subt”>
```

```
         <dimref elemref=”video” projectr=”prj_11” selcomp=”last”>
         300
      <\HyOp>

<!-- projector scope extent before interaction -->
< extlist id=”upexall”>
   <dimspec id=”dupexall”>
      <dimref elemref=”devent_1” selcomp=”first”>
      <dimref elemref=”devent_1” selcomp=”last”>

<!-- projector scope extent after interaction -->
<extlist id=”upsec”>
   <dimspec id=”dupsec”>
      <ConFunk usefun=”con_last” args=”ufcs 300”>
      <dimref elemref=”devent_1” selcomp=”last”>

<fcs id=”ufcs”> <!-- unprojected coordinate space -->

   <evsched id=”upevched”> <!-- unprojected event schedule -->
      <event id=”video” extlist=”event_1”>

   <baton id=”baton1”> <!-- baton used before interaction -->
      <proscope exspec=”upexall”>
         <projector id=”prj_11”>
            <extlist id=”exprj_11”>
               <dimspec id=”dprj_11”>
                  <dimref elemref=”SCOPE” selcomp=”first”>
                  <dimref elemref=”SCOPE” selcomp=”last”>

   <baton id=”baton2”> <!-- Baton used after the interaction -->
      <proscope exspec=”upexsec”>
         <projector id=”prj_21”>
            <extlist id=”exprj_21”>
               <dimspec id=”dprj_21”>
                  <ConFunk usefn=”con_last” args=”pfcs 0”>
                  <dimref elemref=”SCOPE” selcomp=”qcnt”>

<fcs id=”pfcs”> <!-- projected coordinate space -->

   <!-- schedule used before the interaction -->
   <evsched id=”psched_1”>
      <button label=”go_on” linkends=”slink” extlist=”butt_ev”>

   <!-- schedule used after the interaction -->
   <evsched id=”psched_2”>

<!-- link relating the button and the following schedule -->
<linkbut id=”slink” trigger=”but1_press” linkend=”psched_2”>

<!-- batrule filling psched_1 used before the interaction -->
<batrule evsched=”upsched” baton=”baton1” pevsch=”psched_1”>
```

```
<!-- batrule filling psched_2 used after the interaction -->
<batrule evsched=”upsched” baton=”baton2” pevsch=”psched_2”>
```

First, a marker function of a derived marker function type *ConFunk* is defined. Assume this marker function type is able to use context information collected by the application and a particular element (*con_last*) returns the current instant in a FCS added by a constant value. For demonstration purposes, we do not specify a concrete script that describes the processing of context information in the marker functions. When calling the defined function, the identifier of a particular FCS is passed on as a parameter to the function. Then, the extent-list of the unprojected event representing the continuous media object is defined. The endpoint of the event is specified as negative value and therefore is counted from the end of the FCS. The position and extent of the interactive button-event is specified dependent on the projected extent of the media-object. The presentation of the button ends 300 time-units before the projected event because the jump offset is 300 time-units and a jump makes only sense before that instant is reached. The projector-scope position and extent of the projection before the interaction is related to the position and extent of the unprojected event. The position of the projector-scope that is applied after the interaction is specified by the marker function *con_last* because it has to be positioned on the current instant of the unprojected FCS added by the jump-offset 300. Then, the unprojected FCS *ufcs* which represents the logical time is specified. This FCS contains two batons: *baton_1* contains the projector to position the projected events if no interaction occurs. *baton_2* contains a projector that positions the remainder of *event_1*. The projector which positions the remainder of *event_1* uses *con_last* to position the projection. The projected FCS *pfcs* which represents the real time contains two schedules: *psched_1* is presented before the interaction occurs and *psched_2* is presented after the interaction. *psched_1* additionally contains the button-event. Then, the *butlink* that relates the button-event with *psched_2* is specified. Finally, the baton-rules are specified that relate the unprojected schedule with projected schedules and projections.

For complex projections, more context information may be needed. Then, a derived projection function containing a script might be used to determine the extent of the projector. Parameters that are references to context information are defined as attributes that are passed on to the projection function.

5 Modelling the interaction types

In this section, we describe how the interaction types introduced in section 2 are represented by the projection approach. Generally, alternative schedules connected by links are used to specify different renditions. Projections are used to specify the effect of interaction with continuous media items. To position events and projections in a context sensitive way, we use derived marker function elements containing scripts.

5.1 Start, selection and stop

The interaction types *start*, *selection* and *stop* modify the set of presented media items. For any possible combination of events that might occur according to interaction, one schedule has to be used. *link*-elements connect interactive events and schedules. To position discrete events which are continued in the subsequently presented schedule, derived marker function are used which apply context information. The projection is applied with events representing continuous media items. The projector-scope and the projection are positioned with derived marker functions. Synchronization constraints are specified by *dimref*-elements.

5.2 Jump

A *jump*-interaction interrupts the regular flow through a hypermedia document. Several types of jump-interaction exist, e.g. to jump to a particular place in a video. The characteristics of this type is the well-known target of the jump. Another type of jump is relative in time, e.g. the user is not interested in the actual sequence of the video and wants to jump forward by 10 minutes. For both types, projections have to be applied. In the first case, a projection has to be defined for each particular instant by defining appropriate projector-scopes. In the second case, the position of the projector-scope is determined by a derived marker function taking the jump-offset into account. The essential part of a possible document instance describing the mapping of the events is specified in the example on page 12-14.

5.3 Pause and continue

The *pause*-interaction causes an interruption in the presentation of media items. A subsequent *continue*-interaction continues the presentation of the media items. During a pause, a schedule describing the pause effect is presented. To present the last displayed information during a pause, a projector with a projector-scope extracting this information is applied for the pause schedule. If other media items should be presented during a pause, they have also to be specified in the pause schedule. Derived marker functions are used to position the pause events. On a continue interaction, the old schedule presenting the remainder of the unprojected events is applied.

5.4 Faster and Slower

Faster- and *slower*-interaction cause the speed-up or slow-down of the continuing rendition. Such a behavior is specified applying projectors that scale the projection according to the different speed factors (Figure 6). In its simplest form, this interaction type causes the speed up or slow down of presentation by a well-known value. In this case, we can define a projector for each speed factor. If not all speed factors are preknown the derived marker function elements have to request the information from the application.

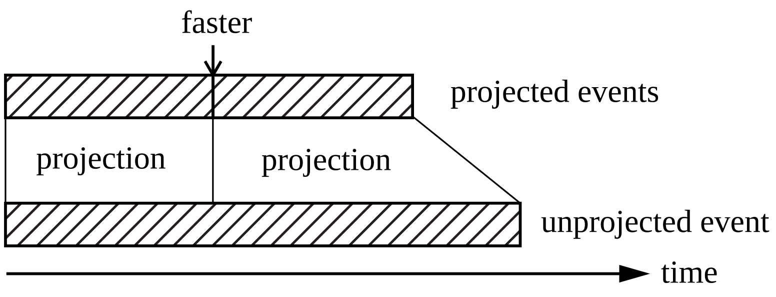


Figure 6: Projection with faster interactions

5.5 Reverse

Reverse-interaction causes a reversal of the presentation direction. Figure 7 shows the mapping of an unprojected event to projected events if the direction of the presentation is reversed twice. The mapping can be specified by a projector. The position of the projector-scope is the position of the first event in the unprojected FCS. The extent of the projector-scope is the current instant in the unprojected FCS which is computed as described above. The projector cannot be specified applying marker functions because they cannot specify the reversion of the projection direction. Thus, we have to use a projector function which contains a script that describes the reverse mapping of events. However, the mapping has to be done by the application because a reverse mapping is not a standard HyTime projection.

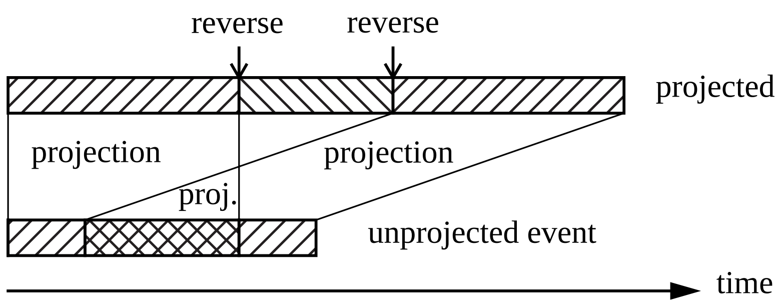


Figure 7: Projection with reverse interactions

6 Conclusion

In conclusion, modelling interaction with HyTime might be done by embedding scripting languages, applying the link module or the rendition module. Embedding scripting languages provides a high flexibility but results in the rendering machines being less compatible as the non-standardized scripting languages have to be interpreted by the application. Also, conflicts may arise because several issues, e.g. temporal constraints, can be specified in HyTime or in scripting languages. With applying the HyTime link module for interaction, more rendering information can be represented within HyTime. But still, the semantics of the links are not defined by HyTime. Thus, the link module introduces some incompatibility due to the necessary interpretation of the link semantics. The third mechanism applying the rendition module conforms to a large set of HyTime mechanisms. However since HyTime does not provide variables, context information during run time has to be passed from the application to the HyTime engine. The context information is needed to map the virtual document time to real time by the rendition module.

As HyTime does not supply any predefined architectural forms for expressing interaction, HyTime has to be extended, e.g. by one of the techniques proposed in this paper. But the non-standardized extensions generally result in incompatible implementations of HyTime-engines although compatibility was one of the intentions of the standardization process. Therefore, new interactive multimedia applications require standardized element types with a predefined semantics. Further, it would be advantageous to have HyTime variables to express indeterministic temporal relations [WaRo94], which exist in interactive systems because many temporal relations are not resolved before rendition time due to interactive events.

A Standard Multimedia Scripting Language (SMSL) is currently developed by the International Standards Organization [BRR94]. By SMSL, embedding scripting languages in HyTime might become more attractive. Further, HyTime is not the only approach of standardizing multimedia document data. MHEG (Multimedia Hypermedia Expert Group) [MHEG92] defines an object model for multimedia control data or the scripting language ScriptX is developed by the Kaleida consortium [Kale93]. In contrast to HyTime, these document models explicitly provide constructs to specify interaction.

7 References

[AnHo91] D. P. Anderson and G. Homsy. A Continuous Media I/O Server and Its Synchronisation Mechanism. *IEEE Computer*, pages 51–57, 10 1991.

[Appl91] Computer Inc. Apple. *QuickTime Developer's Guide*. Developer technical Publications, 1991.

[BHL91] G. Blakowski, J. Huebel, and U. Langrehr. Tools for Specifying and Executing Synchronised Multimedia Presentations. *2nd. Intl. Workshop on Network and Operating System Support for Digital Audio and Video, Heidelberg*, 11 1991.

[BRR94] J. F. Buford, L. Rutledge, and J.L. Rutledge. Integrating Object-Oriented Scripting Languages with HyTime. In *IEEE 1st Intl. Conference on Multimedia Computing and Systems, Boston*, pages 456–462. p, 5 1994.

[BuZe93] M. Cecilia Buchanan and Polle T. Zellweger. Automatic Temporal Layout Mechanisms. In *ACM 1st Intl. Conference on Multimedia, Anaheim*, pages 341 – 350, 8 1993.

[Erfl94] Robert Erfle. HyTime as the Multimedia Document Model of Choice. In *IEEE 1st Intl. Conference on Multimedia Computing and Systems, Boston*, pages 445–454, 5 1994.

[Gold91] Charles F. Goldfarb. HyTime: A standard for structured hypermedia interchange. *IEEE Computer*, pages 81–84, 8 1991.

[HyTi92] HyTime. Information technology - Hypermedia/Time-based Structuring Language (HyTime). *ISO/IEC DIS 10744*, 8 1992.

[Kale93] Kaleida. Kaleida, ScriptX, and the Multimedia Market. Kaleida Labs Inc., 1945 Charstom Road, Mountain View, USA CA94043, 1993.

[KRRK93] John F. Koegel, Lloyed W. Rutledge, John L. Rutledge, and Can Keshin. HyOctane: A HyTime Engine for an MMIS. In *ACM 1st Intl. Conference on Multimedia, Anaheim*, pages 129–136, 6 1993.

[Lamp78] Leslie Lamport. Time, Clocks and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 7 1978.

[LiGh90] T. D. C. Little and A. Ghafoor. Synchronization and Storage Models for Multimedia Objects. *IEEE Journal on Selected Areas in Communications*, 8(3):413–427, 3 1990.

[MHEG92] ISO/IEC/WD MHEG. Information Technology - Coded Representation of Multimedia and Hypermedia Information Objects. Working Draft 5, ISO/IEC, 3 1992.

[NKN91] Steven R. Newcomb, Neill A. Kipp, and Victoria T. Newcomb. "Hytime", The Hypermedia/Time-based Document Structuring Language. *Communications of the ACM*, pages 67–83, 11 1991.

[RoHe94] Kurt Rothermel and Tobias Helbig. Clock Hierarchies: An Abstraction for Grouping and Controlling Media Streams. Technical Report 2, Universitaet Stuttgart, 4 1994.

[WaRo94] Thomas Wahl and Kurt Rothermel. Representing Time in Multimedia Systems. In *IEEE 1st Intl. Conference on Multimedia Computing and Systems, Boston*, pages 538–543, 5 1994.