Authoring and Formatting of Documents Based on Event-Driven Hypermedia Models*

Rogério F. Rodrigues[1] rogerio@telemidia.puc-rio.br
Luiz Fernando G. Soares[1] lfgs@inf.puc-rio.br
Guido L. de Souza[2] guido@dimap.ufrn.br

[1]Dept. Informática, PUC-Rio, R. Marquês S. Vicente 225, 22453-900 - Rio de Janeiro, Brasil
[2]DIMAp, UFRN, Campus Universitário, Lagoa Nova, 59072-970 - Natal-RN, Brasil

Abstract

This paper presents a critical analysis of specification and presentation environments for hypermedia documents with time constraints. Throughout of its sections, the solutions implemented in the HyperProp authoring and execution environment are also presented.

1. Introduction

Hypermedia systems must satisfy, at least, three main requirements. First, it must support different types of media segments. Second, it must allow explicit definition of temporal and spatial relationships among several media segments, including those relations triggered by user control. Third, it must implement a versatile temporal formatting algorithm.

Moreover, hypermedia systems may introduce several desirable facilities, such as:

- allowing a structured authoring (hierarchical or not) of the document;
- supporting, for each media segment, a rich set of capabilities, like: presentation duration flexibility, different exhibition alternatives, behavior changes during presentation, etc.;
- supporting, for each media segment, relations anchoring on internal points of the segment (fine granularity), and not only on its start or end points (coarse granularity);
- allowing quality of service (QoS) definition required by media segments (e.g., jitters, bandwidth, etc.);
- allowing the presentation environment description (e.g., network delay, supported devices, etc.);
- allowing the explicit definition of temporal relationships with non deterministic times;
- supporting a temporal formatting algorithm that considers the non determinism and allows correcting the presentation on-the-fly, when unpredictable events occur (e.g., network delay, user interaction, etc.); and
- supporting a temporal formatting algorithm that takes profits of media segments QoS specification and the environment characteristics, for instance, realizing pre-fetch of the objects' content.

Hypermedia systems have been addressed in three different levels in the literature: storage, specification (authoring) and execution (formatting). This paper focus on the latter two. The specification level aims at defining hypermedia applications requirements and associated constraints. The formatting level refers to the development of protocols and schemes for document exhibition, dealing with intra-media temporal synchronization, and inter-media temporal and spatial synchronization. In this paper, we only consider inter-media synchronization.

Several systems discussed in the literature deals with authoring and formatting issues. Examples are Firefly [2], CMIF [15], I-HTSPN [16] and

* In Proceeding of the IEEE Conference on Protocols for Multimedia Systems and Multimedia Networking. Santiago, Chile, November 1997. pp. 74-83.

HyperProp [13]. All of them use constraint-based specification models. This paper discusses one of these models in more detail, the conceptual model using the object oriented paradigm. More precisely, this paper presents solutions given in the HyperProp system, comparing them with solutions proposed in related works.

The paper is organized as follows. In Section 2, issues related to hypermedia document authoring are discussed, pointing out the importance of document logical structuring, and showing how to use compositions to support that structuring. Section 3 describes the temporal and spatial formatter of the HyperProp system. Finally, Section 4 contains the conclusions. Comparisons with related works are presented throughout the text.

2. Structured authoring of hypermedia documents with temporal constraints

An authoring environment should offer good editing and browsing tools for defining the logical structure of a document, its components' content and the content granularity, which specifies the set of information units that can be marked and used in the definition of events. The exact notion of information unit and marked information unit (an anchor) is part of the definition of the document's component, from here on called a node. For example, an information unit of a video node could be a frame, while an information unit of a text node could be a word or a character. Any subset of information units of a node may be marked.

An *event* is defined by the presentation of a marked set of information units of a node (presentation event) or by its selection (selection event) or by the changing of an attribute of a node (attribution event).

An event can be in one of the following states: *sleeping*, *preparing*, *prepared*, *occurring* and *paused*. Moreover, every event has an associated attribute in the node where it is defined, named *occurred*, which counts how many times an event transits from occurring to prepared state during a document presentation.

Intuitively, taking a presentation event as an example (see Figure 1), it starts in the sleeping state. It goes to the preparing state while some prefetch procedure of its information units is being executed. At the end of the procedure, the event goes to the prepared state. At the beginning of the information units exhibition it goes to the occurring state. If the exhibition is temporarily suspended, the event stays in the paused state, while the situation lasts. At the end of the exhibition, the event comes back to the prepared state, when the attribute occurred is incremented. Obviously, instantaneous events, like selection and attribution, stay in the occurring state only during an infinitesimal time. The event state machine is executed under the document formatter responsibility, as will be seen in Section 3.

The definition of a node content and the definition of its granularity (definition of its events) are out of the scope of this paper. From here on, objects that contain these definitions are called *data objects*. Usually, the creation of such objects is done with specific media editors.

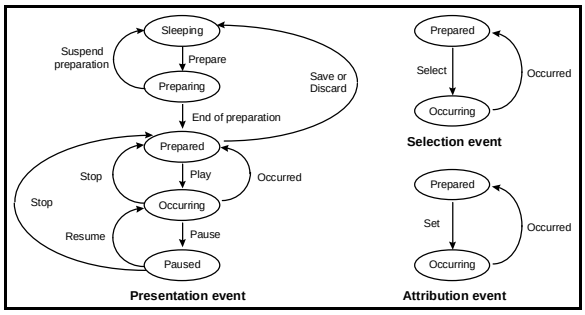

Figure 1 - Events state machine

An authoring environment, however, should also permit the definition of each component expected behavior when presented, and the specification of relationships among components.

The possibility of having relationships among components of a document represents the main characteristic of a hypermedia system. However, it is important to combine this facility with mechanisms to organize the document in order to reduce the so-called "lost in the hyperspace" problem [8]. Thus, we need some scheme to structure the document and to define relationships among its components, preferably independent of the component content, so that we can reuse data objects without inheriting relations defined over them, as happens in several hypermedia systems, exemplified by WWW[1]. Conceptual Models with composite nodes support such schemes, in particular, models that allow nested compositions.

[1]More detailed discussions can be found in [8]. Also [1] discusses and exemplifies the difficulties found in hypermedia systems where links are tightly bound with node contents.

The remain of this section discusses how to specify the expected behavior of each document node when presented and how to specify relationships among them.

2.1 Temporal behavior specification of multimedia objects

For each presentation event, one should be able to specify how and with which tool the associated data object will be presented. All these definitions must be preferably specified independent of the associated data object. In HyperProp, objects that contain this specification are called descriptors. *Descriptor*, thus, defines behavior changes during the presentation of a data object.

The independence between descriptors and data objects will permit better reuse of objects. For example, using distinct descriptors, one can define different presentations for the same data object. A text media segment can be presented as text, using descriptor D_1, or it can be synthesized as audio using descriptor D_2. As different descriptors can lead to different event duration, different quality of presentation and different platform requirements, the definition of all these issues should also be part of the descriptor and not part of the data object. The aggregation of a data object and a descriptor in order to present a component is called *representation object*.

The object oriented model used in the HyperProp (named Nested Context Model - NCM [3]) defines a *descriptor* as an entity that has as attributes a start specification, an end specification and a collection of event descriptions.

A *start specification* contains all information needed to start a node presentation. In particular, it defines methods for node exhibition and editing. It also has an ordered list of operations that must be executed when preparing the node for exhibition. This list defines all parameters needed for creating a representation object from a data object.

Similarly, an *end specification* contains all information needed to finish a node presentation. In particular, it defines methods to be run at the end of a node presentation. It also has an ordered list of operations that must be executed when finishing a node exhibition.

An *operation list* contains an ordered sequence of operations. Each operation has a condition and a set of actions. Conditions must be satisfied in order that the associated actions be triggered. Conditions evaluate logical expressions over event states and attribute values of the representation object created from the descriptor. Conditions can be binary simple or compound. A binary simple condition has one previous condition, that must be satisfied immediately before the time of evaluation, and a current condition, that must be satisfied on the time of evaluation. We say that a binary simple condition is satisfied if both its previous and current conditions are satisfied. A compound condition is a logical expression of binary simple conditions. The actions of an operation list must always correspond to a behavior change in the exhibition of the representation object created from the descriptor. For example, change an audio volume to “X” dB.

An *event description*, in its turn, consists of the identification of an event and its type, the event exhibition duration (in the case of a presentation event), and an operation list, which must be executed if the event occurs. In other words, all behavior changes during the presentation of a data object are specified in the operation list of the descriptor used to create the corresponding representation object.

2.2 Structuring documents with compositions

The structured definition of documents is desirable as it carries built-in concepts of modularity, encapsulation and abstraction. The notion of structured documents arises from the introduction of the composition concept, as a container of documents’ components and their relationships.

The definition of hypermedia documents in NCM is based on two familiar concepts, namely nodes and links. *Nodes* are fragments of information and *links* interconnect nodes into networks of related nodes. The model goes further and distinguishes two basic classes of nodes, called *content* and *composite* nodes, the latter being the central concept of the model.

Intuitively, the *content node* contains data whose internal structure, if any, is application dependent and will not be part of the model (they are the usual hypermedia nodes). The class of content nodes can be specialized into other classes (*text, video, audio, image*, etc.), as required by the applications.

A *composite node C* is a node whose content is a collection *L* of nodes and links such that every base node of every link occurring in *L* is either *C* itself or a node occurring in *L* (the definition of base node of

a link is given below - the definitions of link and composition node are indeed mutually recursive).

We say that an entity E in L is a *component of C* and that *E is contained in C*. We also say that a node A is *recursively contained in B* iff A is contained in B or A is contained in a node recursively contained in B. Note that a component may be included more than once in L. However, in this paper we will assume the links and nodes collection as a set, without loss of generality for our discussion. An important restriction however must be done: a node cannot be recursively contained in itself.

As the model allows different composite nodes to contain the same node and composite nodes to be nested to any depth, it is necessary to introduce the concept of perspective. Intuitively, the perspective of a node identifies through which sequence of nested composite nodes a given node instance is being observed. Formally, a *perspective* of a node N is a sequence $P=(N_m,...,N_1)$, with $m \geq 1$, such that $N_1=N$, N_{i+1} is a composite node, N_i is contained in N_{i+1}, for $i \in [1,m)$ and N_m is not contained in any node. Note that there can be several different perspectives for the same node N, if this node is contained in more than one composite node.

A *link* has three main attributes, the *source end point set*, the *destination* or *target end point set* and the *meeting point*. The set of source and destination end points will define events. The meeting point will define relationships among events.

The end points are tuples of the form $<(N_k,...,N_1), \alpha,$ type> such that N_1 is a node, N_{i+1} is a composite node and N_i is contained in N_{i+1}, for all $i \in [1,k)$, with $k > 0$, α is a set of information units (an anchor) of N_1 or an attribute identifier (and its value) of N_1, and *type* specifies the event type associated with α: (selection, presentation or attribution). The node N_k is called a *base node* of the link.

The *meeting point* contains one operation, as usual, composed by one condition and one action. Once the condition is satisfied, the associated action is triggered. The conditions evaluate logical expressions over event states and attribute values of representation objects specified by the source end points of the link. Conditions can be binary simple or compound, as already defined for descriptors. Actions in a meeting point are operations that must be executed over the destination end points of the link, or are just to introduce a time delay. They can be simple or compound. Examples of simple actions are: start, pause, stop, prepare a presentation event E; activate, enable, disable behavior change operations associated with an attribution event E; wait a time delay; etc. A compound action is defined by an expression of actions based on the | (parallel) and $\rightarrow$ (sequential) operators, which define the execution order of each element of the expression.

In a future work we intend to have the expressions written in RT-LOTOS [6], what will bring the benefit of having well-known algorithms for consistency validation.

Turning back to the use of composite nodes in the modeling of structured documents, they must bring out some desired properties, such as:

- composition nesting, that is, compositions that contain other compositions;
- grouping of the components of a document and the relationships among them independent of their types (synchronization relationships for presentation, selection relationships for usual hyperlinks navigation, etc.);
- composite nodes use as a new type of node, in all senses, that is:
  - that they can be presented[2] — since in a presentation, it is important to exhibit not only the data content of a document, but also its structure specified in the composite node (for example, when accessing a book chapter modeled as a composite node, besides seeing its content, one may want to visualize its section structuring).
  - that different entry points (anchors) in a composition can be defined, i. e., that in a composition, components may have different presentations, depending on the entry point. For instance, the duration of a composition (duration of its components exhibition) will depend not only on the duration of its components, but also on the associated entry point;
  - that relations among compositions can be defined.
- inheritance in the composition nesting, in the sense that relations can be defined in a composition C, referencing components recursively contained in C. This mechanism is extremely important in object reusing. For example, suppose a composition

[2] Composite node presentation is different from the presentation of its components. Composite node presentation is the exhibition of the structure defined in the composition and not the exhibition of each one of its components.

representing a book chapter. For a book given to a reader (composition B_1), it could be desirable to introduce a relation between two sections to give a hint of related matters; for another advanced reader, the book (composition B_2) should be delivered without that relation. Note that B_1 could be defined as a composition containing B_2 and the introduced relation, reusing all the structure of B_2.

- composite node presentation to help user navigation through a document — what can require the use of some filtering mechanism to present the document structure, as discussed in [11], in order to lessen the user disorientation problem.

The object oriented model NCM used in the HyperProp system defines compositions with all the requirements specified before. Figure 2 shows a simplified version of the NCM class hierarchy. A complete description of the model can be found in [13] and [14].

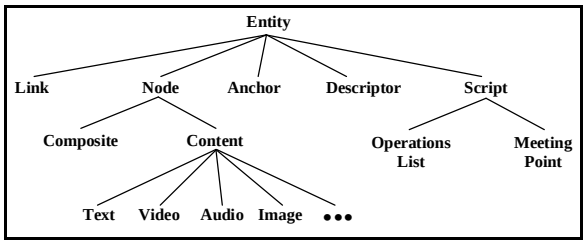

Figure 2 - Simplified illustration of the Nested Context Model class hierarchy

2.3 Related works

Spatial synchronization can be defined, optionally, in the same relationships responsible for defining temporal synchronization. A link relation can not only specify the moment that a given event will happen but also how must be its presentation inside the current presentation context. The most common case, however, is to have only temporal constraints specified by relations and have spatial constraints defined in other object. This is the case of Firefly and HyperProp, where behavior changes and spatial synchronization are handled by operation lists associated with data objects (in HyperProp these lists are contained in the corresponding descriptor).

Some systems allow a join definition of the spatial synchronization. They specify an object that can be shared by several representation objects. In this case the descriptor does not realize the spatial synchronization. This is the case of CMIF, where an event descriptor is assigned to a *channel*, which is an abstraction of a set of properties shared by other objects of the same type.

All the authoring systems discussed in this paper, Firefly, CMIF, I-HTSPN and HyperProp have an object descriptor separated from the associated data object. Also, in all of the systems the data objects only refer to their contents that are stored in another objects.

Commercial systems addressing synchronization are generally timeline (MAEstro, MacroMind Director, etc.) or scripting based (ToolBook, etc.). Conceptual models using timeline and scripting paradigm have several problems. In both is very difficult, if not impossible, to use concepts of modularity in a document presentation. When one needs to reuse part of a presentation, it is not clear where a copy must begin or end. Another issue is the difficulty to specify the temporal requirements between events whose precise duration is variable or unknown until runtime. Like programming, scripting is very useful for small-scale presentations, but editing of large documents can be cumbersome without a facility for structuring the presentation. In addition, the detailed specification of parallel activities shows the same problems found in most programming languages. Maintaining a presentation in a timeline based authoring environment can be very hard. A simple change in an object position on time can represent a new evaluation of all timeline.

Constraint-based (event-driven) synchronization does not have the above limitations. All the mentioned related works discussed in this paper use the event-driven approach as HyperProp.

In [7], a Time Petri Net based model was defined (TSPN - Time Stream Petri Nets). An event is represented in a TSPN by a place and by associating a tuple $[t_{min}, t_{opt}, t_{max}]$ to an outgoing arc from the place. The tuple $[t_{min}, t_{max}]$ specifies the possible duration interval of the event, and t_{opt} specifies the time duration with best QoS presentation. Delays can be introduced in TSPN through adding places with outgoing arcs associated with the possible interval of delay occurrence. A hyperlink (usual hypermedia link) is represented by a new place type having a usual tuple $[t_{min}, t_{opt}, t_{max}]$ associated to an outgoing arc, representing the possible selection interval of the link (note that t_{max} can be ∞ and t_{opt} can be indeterminate).

TSPN allows n:m relationships like HyperProp. Willrich [16] extends this model in order to address document structuring, to allow specifying information access parameters and to allow specifying spatial and audible presentation

characteristics of an object. They named this model I-HTSPN.

CMIF uses synchronization arcs to specify its presentation. A synchronization arc is a 1:1 temporal relation between events. It has an associated delay and an allowed deviation from this delay. CMIF events have always a predictable duration (specified by a unique value), and represent the presentation of the whole component of the document (coarse granularity). A special type of arc between events, called continuous sync arc defines that the delay specified must be maintained during all the concurrent presentation of the related events.

Firefly does not have compositions, which are addressed as future work. Both CMIF and I-HTSPN compositions do not allow relationship inheritance, neither the exhibition of the structure defined by compositions.

The CMIF composition always defines a hierarchical structure and only allows relationships among presentation events. The relationships are given implicitly by the composition type, that can be parallel or sequential, where all components must be presented in parallel or in sequence, respectively. No other relationship is allowed in CMIF compositions. As opposed to HyperProp, CMIF allows other types of relationships in another model entity called synchronization arcs. When defined in these arcs, we cannot have a structured reuse of the document. As future work, CMIF plans to specify a maximum and minimum time in the constraints defined in its relations. This can have no problem in relationships defined in arcs, but can be difficult for relations defined in its compositions.

It is worth to note that the composition class of the HyperProp object oriented model can always be specialized in parallel composition and sequential composition subclasses, with the same functionality defined in CMIF.

I-HTSPN compositions are a little bit more general than CMIF's, but not so general as HyperProp compositions. In I-HTSPN, we can have a composition as an end point of a link, but we cannot have a component inside a composition as an end point. Compositions are represented by a new place type in the Petri network. A composite place and its related subnet must not only be structurally equivalent (i.e., the subnet must have an input place and an output place), but also temporally equivalent. Thus, both CMIF and I-HTSPN compositions have only one entrance (anchor) point.

The way compositions are defined is one of the main differences between HyperProp and the other systems. CMIF and I-HTSPN compositions are used to structure the document presentation; the logical structure of the document and the presentation structure are one. For example, if a component of a desired composition modeling Chapter *A* of a book is the navigation source to another component of another desired composition modeling Chapter *B*, both components must be defined in one composition, loosing the desired logical structure. In HyperProp, compositions indeed model the logic structure of a document.

2.4 Document design environments: authoring in HyperProp system

Almost all systems mentioned in this paper offer authoring tools in a graphical interface, making easier the task of presentation specification. This section briefly presents the HyperProp authoring environment, which is based on three different views, as shown in Figure 3.

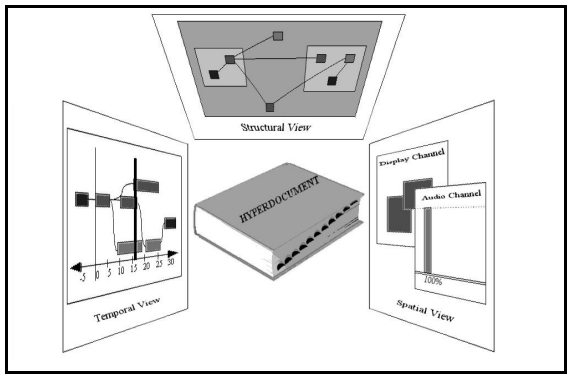

Figure 3 - Different views of a hypermedia document

The first view, called *Structural View*, supports browsing and editing the logical structure of hyperdocuments, providing features for editing nodes and links and grouping them into compositions, as illustrated in the upper plane of Figure 3. In this view, nodes are represented by rectangles, links are represented by lines and the containment relationship of composite nodes is represented by the inclusion of rectangles (nodes) and lines (links), in another rectangle (composite node). Sophisticated filtering algorithms were developed to avoid the author disorientation, mainly in complex documents with lots of nodes and links [11]. None of the already mentioned systems present this facility.

The second view, called *Temporal View*, is responsible for supporting the specification of temporal relationships among components of a hyperdocument, defining their relative position in time, as presented in the left plane of Figure 3. In this plane, nodes are represented by rectangles, whose lengths indicate the mean optimum duration of their presentation in a time axis and whose relative positions establish their temporal synchronization. A remarkable difference between the Temporal View and a timeline is that the time shown in the Temporal View is not explicitly specified by the author. It is derived from temporal constraints, just as an approximation of the real-time presentation [4].

Finally, the third view, called *Spatial View*, supports the definition of spatial relationships among components of a document, establishing its presentation characteristics in a given device, in a specific instant of time, as shown in the right plane of Figure 3. In this plane, rectangles represent spatial characteristics of objects, specifying, for example, their position in a video monitor or their volume level in an audio device, in a given specific time instant.

The three views are related with each other. The focused object in the Structural View is the basis for the time chain shown in the Temporal View. In the latter view, for each point in time, the Spatial View shows how components will be presented in the space defined by the output devices. Links and nodes defined in the Temporal View are immediately updated in the Structural View, and vice-versa.

Since object oriented conceptual models are higher level abstractions than temporal dependent graphs and Petri nets, they are closer to the user. The great expressiveness of these models can be offered to user in an easy and comprehensible way. Furthermore, other facilities can be easily incorporated, such as version control and cooperative work [13], what would be hard to introduce in other models. Object oriented models such as the one used in HyperProp can also be more easily converted to the interchangeable multimedia/hypermedia objects standard defined in MHEG. These are some reasons to adopt an event-driven object oriented model to specify the logical structure and presentation characteristics of a document. The input structure for the temporal and spatial formatter will be extracted from this model. As the formatter is closer to the operational machine, a lower level model for parallel state machines may then be the most adequate for tasks like scheduling, as will be discussed in the next section.

3. Temporal and spatial formatting in HyperProp system

The temporal and spatial formatter is responsible for controlling the document exhibition based on its presentation specification and the platform (or environment) description. The main idea is to build an execution plan (a schedule) to guide the formatter in its task. The execution plan should contain information about the actions that must be fired when an event occurrence is signalized.

We say that an event is predictable when it is possible to know, a priori, its start and end relative time to another event. Otherwise, an event is called unpredictable. The sequence of event occurrences in time is called *time chain*. When the first event of a time chain is unpredictable, the time chain is called *partial time chain* [2]. The execution plan is the set of all partial time chains of a document. It will guide the formatter in adjusting object duration on-the-fly, as well as in pre-fetching components' content in order to improve the presentation quality and to reduce the probability of temporal and spatial inconsistencies.

This section briefly describes the HyperProp formatter and compares it with solutions given in the already mentioned related works.

3.1 HyperProp formatter

The HyperProp formatter architecture is composed by four elements: the pre-compiler, the compiler, the executor and the viewer controllers (or simply controllers), as shown in Figure 4.

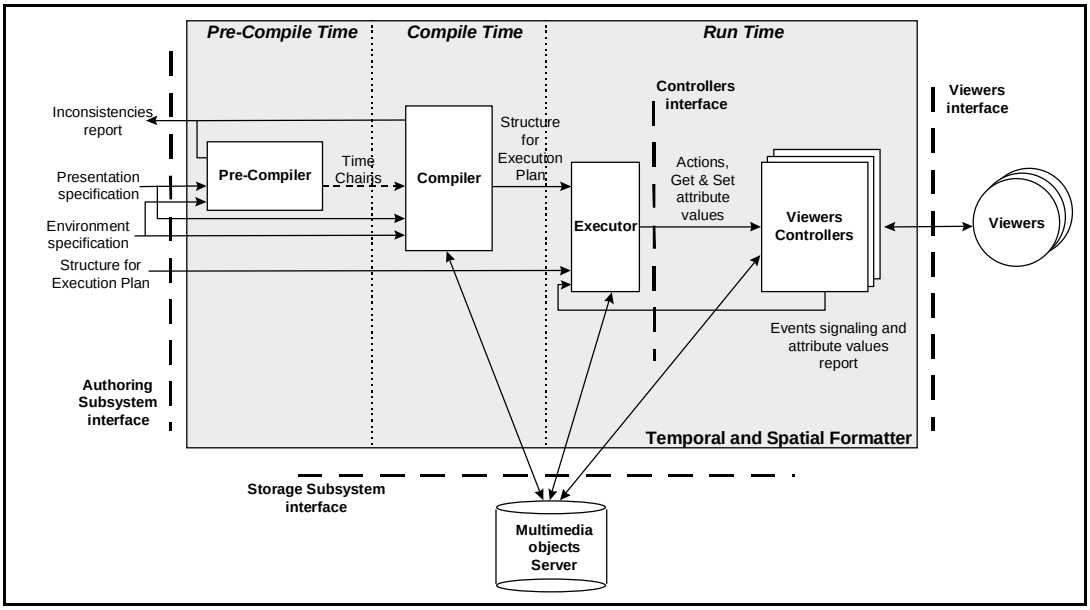

Figure 4 - HyperProp formatter architecture

Although the pre-compiler realizes formatting tasks, it is indeed a support to the authoring environment. Its main function is to check the temporal and spatial consistency of each document partial time chain. The pre-compilation is an incremental compilation done during authoring time. It can help the author to find errors, giving an instantaneous feedback whenever any inconsistency is detected, like mismatched time relationships (temporal inconsistency), or conflicts in a device use (spatial inconsistency), or even an absence of a certain device needed in the presentation (environment inconsistency).

Considering the complete document presentation specification and the exhibition platform description, the compiler is responsible for generating the data structure used by the formatter to build the execution plan. As stated in Section 2, the event presentation duration in NCM is specified in the descriptor. One must specifies a tuple < t_{min}, t_{opt}, t_{exp}, t_{max}, f_{cost}>, where the minimum allowed duration (t_{min}), the maximum allowed duration (t_{max}), the duration that would give the best quality of presentation (t_{opt}) and the cost of shrinking or stretching the event duration (f_{cost}) are defined. The expected value (t_{exp}) is set initially to be equal to the optimum value. At compiler time, the expected duration should be computed based on the cost minimization in order to warrant the spatial and temporal consistency of the document. These adjustments are not yet implemented in the current version of HyperProp and are addressed to future works.

In the current HyperProp implementation, when an user asks to present a document, the compiler gets, from the Multimedia Objects Server (NCM server), all the composite nodes recursively contained in the composition that models the whole document. From these composite nodes, the compiler builds a list containing all links that can be traversed during a presentation. From the links, expressions that are contained in their meeting points are compiled in order to obtain the concerning events and, with them, create the partial time chains. For each event in the time chain, the compiler creates a reference to all meeting point where it is used. Thereby, when an event occurs the executor knows all links that must be evaluated. All partial time chains together make a presentation state machine that feeds the executor.

HyperProp uses an extension of TSPN to model the execution plan. The extension consists simply of adding to outgoing arcs not only the minimum, optimum and maximum event duration, but also the expected duration calculated by the compiler and maintained by the executor. Note that it was made a conversion from a higher level abstraction model, NCM, to a TSPN. Detailed discussion about the mapping can be found in [12].

As a future work, we are planning to use RT-LOTOS in meeting point expressions. One possibility to be investigated is to map RT-LOTOS expressions directly in a LOTOS DTA (Dynamic Timed Automata) [5] and use it not only for inconsistencies checking, but also as the execution plan data structure, instead of the time Petri net.

Representation objects are created by the executor from the corresponding descriptors and data objects. Given a data object, an associated descriptor can be specified on-the-fly by the end user, or in a data object attribute, or in a link that has the data object as its anchor. Composite nodes also have, for each node that they contain, an attribute that can be used to store a descriptor identifier.

When presenting a node, the descriptor explicitly defined on-the-fly by the end user bypasses the descriptors defined during the authoring phase. These in turn have the following precedence order: first, that defined in the link used to reach the node; second, that defined in the composite node that contains the node, if it is the case; third, that defined within the node; and finally, the default descriptor defined in the node class.

The executor does not create the representation object directly. Instead, it starts a controller and passes all information needed to create the representation object and control its entire presentation. This information includes the descriptor object and a list of all events that should be reported, obtained from the data structure received from the compiler. The executor knows the type of the controller to be started, based on information contained in the descriptor.

The controller creates the representation object obtaining its corresponding data object from the Multimedia Object Server. Through the associated descriptor, the controller obtains the operation list that determines the behavior changes that must happen during the object presentation. The concept of a controller for each representation object, with a well-defined interface to the executor, will allow incorporating commercial exhibitors to the system, such as Word, Netscape, etc. It will be only necessary to implement controllers for these viewers being able to transmit messages according to the established interface.

From the viewers, controllers receive event signaling to be reported to the executor. The executor then updates the appropriate event state machine and evaluates all links associated with this event. If there is any condition in the link meeting point that is satisfied, the corresponding actions are fired. At the moment the controller reports an event to the executor, it can check the expected time in the execution plan to see if some adjustments are needed. The adjustments may be done through messages sent to controllers, telling them, for example, to accelerate the presentation rate, etc. These adjustments are not yet implemented in the current version of HyperProp and are addressed to future works.

In the current implementation, the NCM server is a centralized process that communicates with multiple clients (user sessions) in different machines. As a future work, we intend to implement a distributed NCM server. The authoring environment, the formatter and the viewers are processes, running in a single machine, that compose the user session. All communication (between processes of the user session and between the user session and the NCM server) is implemented over the TCP/IP protocol (sockets library). This permits, with little effort, to implement a distributed user session that has their controllers/viewers running in machines different from that of the formatter executor.

3.2 Related works

The architecture of the Firefly formatter is very similar to the one shown in Figure 3. Indeed, that formatter was the starting point of the HyperProp formatter design. The main differences between them are that in Firefly there is not a pre-compiler module and there is only one controller for all representation objects. In Firefly, the compiler (called scheduler) gives to the executor (called runtime formatter) partial time chains structured as timelines. Firefly tries to avoid timeline problems, mentioned in Section 2, by concatenating partial time chains to support unpredictable events. The compiler builds a main time chain (called main temporal layout) and other partial time chains (called auxiliary temporal layout - time chains), when an user asks for a document presentation. The executor builds the execution plan from these structures. Because each time chain is a timeline, execution adjustments may eventually induce a great change in the remaining timeline. This problem becomes worse in large partial time chains. In the current version, Firefly implements adjustments only at compiler time.

The CMIF formatter (called Player) has a compiler that generates, from the document specification, a directed graph of timing dependencies that feeds the executor. The graphs used by CMIF are similar to Petri nets.

The I-HTSPN toolkit has a pre-compiler (called Analyzer) that checks time and spatial inconsistencies of multimedia modeled scenarios.

The compiler (called MHEG translator) converts the I-HTSPN specification into an MHEG representation. An MHEG machine should control the document presentation.

4. Conclusions

We can already get some conclusions from the HyperProp authoring and formatting environment implementation. First, it is extremely easy and flexible to work with higher-level models at the authoring level. Second, it is easier for the author to deal with a document logical structuring than with only a document presentation structuring. Hence, models based on compositions allowing every type of relationships between their components become very important. Third, it is possible to build graphical editing environment, even in models with great expressiveness, allowing authors to specify complex documents without feeling lost. In order to accomplish that, filtering and animating algorithms were developed in the structural browser project. Fourth, the pre-compiler time is very useful in detecting inconsistencies during authoring phase. Finally, lower-level models for formatting internal data structures make easy the scheduling and temporal adjustments on-the-fly.

The implementation of the tools presented was done over the UNIX platform using C++ language. To implement the structural view we have used some routines of the compound graph editor D-Abductor [10], of Fujitsu Laboratories Ltd., that provide automatic graph layout and animation features. The temporal and spatial views were implemented using a portable tool set for building graphical user interfaces called IUP/LED and CD [9]. Initially, the use of different tools to build the user interface was considered irrelevant to our work, however, the next step is to rebuild the graphical editor user interface using a unique portable tool set. The formatter algorithms for link evaluation are implemented with threads. The communication between the executor and the controller uses TCP/IP sockets. We plan in the next version of the formatter to implement adjustment algorithms both at pre-compiler/compiler time and at run time. We also plan to implement algorithms to check inconsistencies on-the-fly and to realize pre-fetch of object contents in order to improve the presentation quality. In order to do that we will have to integrate the controller implementation to the storage environment using tools to allow negotiating QoS parameters.

References

[1] Batista, T.V.; Rodriguez,. N.R.; Soares, L.F.G.; Resende, M.C.; MMM: Hypermedia Mail over WWW; *Proceedings of the Third International Workshop on Community Networking.* Antuerpia, May 1996; pp. 83-89.

[2] Buchanan, M.C.; Zellweger, P.T.; Automatic Temporal Layout Mechanisms. *Proceedings of ACM Multimedia'93*, California, 1993. pp. 341-350.

[3] Casanova, M.A.; Tucherman, L.; Lima, M.J.; Rangel, J.L.; Rodriguez, N.L.R.; Soares, L.F.G.; The Nested Context Model for Hyperdocuments; *Proceedings of The Third ACM Conference on Hypertext*, San Antonio, December, 1991.

[4] Costa, F.R.; Um Ambiente para Definição da Sincronização Temporal e Espacial de Objetos Multimídia/Hipermídia, *Master Thesis*, PUC-Rio, Rio de Janeiro, August 1996.

[5] Courtiat, J.P.; Oliveira, R.C.; A translation from RT-LOTOS into dynamic timed automata, Technical Report, Laas, Toulouse, 1995.

[6] Courtiat, J.P.; RT-LOTOS and its application to Multimedia protocol specification and validation, *Multimedia Networking*, Aizu, 1995.

[7] Diaz, M.; Sénac, P.; Time Stream Petri Nets, a Model for Timed Multimedia Information. *Proc. Of the 15th Int. Conf. On Application and Theory of Petri Nets*, Zaragoza, 1994. pp. 219-238.

[8] Halasz, F.G.; Reflections on Notecards: Seven Issues for the Next Generation of Hypermedia Systems, *Communications of the ACM*, Vol. 31, n. 7, 1988.

[9] Levy, C. H.; IUP/LED: Uma Ferramenta Portátil de Interface com o Usuário, *Master Thesis*, PUC-Rio, Rio de Janeiro, 1993.

[10] Misue, K., D-ABDUCTOR 2.30 User Manual, Institute for Social Information Science, FUJITSU LABORATORIES LTD., Japan, 1994.

[11] Muchaluat, D.C.; Browsers e Trilhas para Documentos Hipermídia Baseados em Modelos com Composições Aninhadas, *Master Thesis*, PUC-Rio, Rio de Janeiro, 1996.

[12] Rodrigues, R.F.; Formatação Temporal e Espacial no Sistema HyperProp; *Master Thesis*, PUC - Rio, Rio de Janeiro, May 1997.

[13] Soares, L.F.G.; Casanova, M.A.; Rodriguez, N.L.R.; Nested Composite Nodes and Version Control in an Open Hypermedia System; *International Journal on Information Systems; Special Issue on Multimedia Information Systems*, September 1995. pp. 501-519.

[14] Souza, G.L.; Soares, L.F.G.; Modelo de Contextos Aninhados - Segunda Versão; Technical Report, TeleMidia Lab, PUC-Rio, Rio de Janeiro, 1997.

[15] van Rossum, G.; Jansen, J.; Mullender, K.S.; Bulterman, D.; CMIFed: A Presentation Environment for Portable Hypermedia Documents; *Proc. of ACM Multimedia'93*, California, 1993. pp. 183-188.

[16] Willrich, R.; Sénac, P.; Saqui-Sannes, P.; Diaz, M.; Hypermedia Documents Design Using the HTSPN Model. *Third International Conference on MultiMedia Modeling — MMM'96*, Tolouse, November 1996. pp. 151-166.