Revisiting the Inter and Intra-media Synchronization Model of the NCL Player Architecture

Soares, L.F.G.; Lima, G.F; Moreno, M.F; Azevedo, R.G; Araújo, E.C; Rios, R.; Batista, C.
PUC-Rio, Rio de Janeiro, Brazil
E-mail: {lfgs, gflima, marcio, robertogerson, edcaraujo, rrios, bidu}@telemidia.puc-rio.br

Abstract: There are many aspects concerned with the management of intra-media and inter-media synchronization of hypermedia application presentations. This paper revisits the integrated inter-media and intra-media synchronization model proposed and implemented by the NCL Player architecture as a specialization of a generic QoS orchestration framework. Some hot-spots of the framework, like the algorithm for building the presentation plan to guide the presentation scheduler, algorithms for pre-fetching, for elastic time computations, and for synchronization tuning are also discussed. In addition, the paper presents a discussion on when the synchronous approach can be used in the presentation engine operation, and when the asynchronous approach is the only possible solution.

Keywords: inter-media synchronization; intra-media synchronization; QoS framework; NCL presentation engine.

1 INTRODUCTION

The control and maintenance of the inter-media and intra-media synchronization in hypermedia document presentations can be stated in terms of QoS orchestration models.

No matter if media objects’ content to be synchronously presented come from different servers (including when end-users act as servers), no matter if a single or multiple communication networks are used, and no matter if the document presentation uses a single or multiple exhibition devices, a general QoS framework for hypermedia presentation engine can be used in many scenarios. This is the choice of the NCL Player engine.


In this paper, we revisit the previous architecture of Ginga reference implementation [6] in view of the latest scenarios involving multiple content sources and multiple exhibition devices, and also based on the experience we have got using Ginga.

The paper is organized as follows. Section 2 gives an overview of generic QoS orchestration models for communication environments, and presents a specialization of one of this models focusing on inter-media and intra-media synchronization. Section 3 discusses the limits of using the synchronous approach in managing the communication with media players embedded in hypermedia presentation engines. Section 4 presents some work related with particular aspects of the framework introduced in Section 2, including the solutions adopted in the current version and in the next version of the Ginga reference implementation. Finally, Section 5 presents our conclusions.

2 THE QoS ORCHESTRATION MODEL

This section gives an overview of generic QoS orchestration models for communication environments, taken SCM as example. In SCM (Service Composition Model) [6, 7], a (communication) service-offering environment (SOE) is delineated by user components and service providers, as illustrated by the cloud with dotted texture in Figure 1 (the other components are explained in Section 2.1).

User components have base-level interaction points and interfaces through which they communicate with each other to define the whole system behavior. Service
providers offer interaction points to which components must bind their interfaces to be able to communicate. An SOE is made up by components, providers, and resources that support the processing of its components and allow them to communicate.

Both user components and service providers can be composed of other components and providers, which can be composed of other components and providers, and so on, until the level of detail the designer is interested in (the primitive level). Figure 1 also reveals a compound user component and a compound provider, both associated to their representatives by dashed lines.

Figure 2 shows a specialization of an SOE for hypermedia systems. In this client-server system we can have several clients, although only one (composite) client component is shown in the figure. Likewise, we can have more than one server feeding the clients. Thus, we can have content coming from different servers via different networks (i.e., possibly composite service providers) that may require inter-media and intra-media synchronization.

In Figure 2, upon a user request, a document presentation begins by starting the Presentation Scheduler component of the presentation engine composite component. Like any resource scheduler, the Presentation Scheduler must be fed with a data structure specifying the media-object (media components of the application) scheduling order. This data structure, here called presentation plan, is built by meta-services, as explained in Section 2.2. Usually this plan is built gradually, during document presentation time. In order to compensate delays and jitters, and thus to avoid inter and intra-media synchronization mismatches, the presentation engine should have a component responsible for controlling the pre-fetching of objects to be presented. This component, named Pre-fetching Scheduler, gets objects from servers (or from object-carousel streams, transmitted as pushed data by servers) based on a data structure that specifies the media object pre-fetching order. This data structure, here called pre-fetching plan, is also built by meta-services, as explained in Section 2.3. In order to get a new object, the pre-fetching scheduler requests the services of the Object Loader component.

Based on the presentation plan, the Presentation Scheduler instantiates new media players (presentation tools), represented by the Player components in Figure 2, whenever it is necessary to present media objects. The media object content, previously retrieved by the Object Loader component, must then be passed to the media player for content exhibition. There can be as many instantiated media players as different media objects are being simultaneously presented. Moreover, player components can reside in different exhibition devices. This is the case when more than one screen is used to exhibit hypermedia applications.

Commands and data sent by the presentation scheduler to control the presentation are subject to delays and jitters issued from communication providers, in particular in the case of media players running on devices other than the one running the Presentation Scheduler or the Object Loader, as discussed in Section 3. In order to decrease the probability of inter-media and intra-media synchronization mismatches, players should be instantiated before they are needed (based on a pre-instantiation plan, not shown in Figure 2), and media content should be delivered to the player device before its presentation takes place. Especially due to non-deterministic events, like user interactions, delays and jitters are inevitable and should be treated by the QoS meta-services discussed in Sections 2.1 to 2.3.

When synchronization mismatches occur, they should be reported to the Presentation Scheduler, by the inter-media synchronization meta-service, as discussed in Section 2.2. This component can then notify the media players to correct the disparities. For example, temporal mismatches may be corrected by commanding media players to shrink or stretch the duration of its content. The presentation engine is the appropriate place to detect this kind of presentation inconsistency and to fire adaptation mechanisms, since it represents the end point in the path from the server to the beginning of the document presentation.

### 2.1 SCM Meta Services

An SOE, namely, its processing and communication resources, its components, and its providers, can be the target of adaptations. In the Service Composition Model (SCM), the representation of all kinds of adaptations is based on the concept of open implementations [8]. In an open implementation, besides base-level interaction points and interfaces that allow normal bindings and interactions, components and providers also have meta-level (ML) interaction points and interfaces (as shown in Figure 2) that reveal some of their internal aspects, making them susceptible to adaptations.

In SCM, adapter components are called meta-components. As any component, a meta-component may communicate with other meta-components through a provider, thus defining a meta-system. An SOE can be the target of several meta-components of different meta-systems. Inter-media synchronization and intra-media synchronization are examples of meta-systems for QoS (synchronization) provisioning, discussed in Sections 2.2 and 2.3, respectively.

The meta-system provider may be deemed independently of the target system provider, as represented in Figure 1. However, in some cases, it may be also worth
representing direct interactions among users, services, and meta-components through one single provider, as, for example, in reflective systems [9]. This is usually the case of Hypermedia System SOEs.

In order to supply an end-to-end QoS, an SOE must divide the QoS provisioning responsibility among its resources. This process is known as QoS orchestration. Considering Figure 2Figure 1, in order to provide hypermedia presentations with QoS, all QoS requirements must be satisfied by the joint work of processing SOE components (presentation engines and servers) and by the communication between them (communication system provider). Each compound component and each communication provider must, in turn, orchestrate the QoS portion previously allocated to them among their internal components and providers, and so on, recursively. This process continues until a primitive component or provider is reached. Processing and communication resources must then be reserved in order to guarantee the QoS portion attributed to this element.

In general, QoS provisioning can be divided in two main phases: QoS negotiation and QoS maintenance. In Figure 1, upon receiving a new service request, the admission controller component verifies the feasibility of its admittance in an SOE, taking into account the current resource utilization and the proposed load associated with the new request. The admission controller starts the negotiation mechanism – which can be centralized, as in Figure 1, or distributed. The negotiation mechanism must identify all possible SOE components and providers that would be involved in the service provisioning. The negotiation agent then establishes portions of the QoS responsibility to each identified element. Afterwards, the mapping mechanisms are launched to translate the requested service category (and its associated parameters) to service categories (and parameters) directly related to the operational capacity of each assigned resource. Then, the admission control mechanisms, linked to each of these elements, are called, recursively.

QoS parameters should be treated from their higher-level specification. As pointed out by [10], technical QoS parameter specification is just half of the problem. QoS parameters need to take into account the media being shared and the social situation, called user-level synchronization requirements in [10]. By relegating synchronization support to the network layer, important semantic information is lost on the nature of the communication taking place. The mapping mechanisms are important in translating these higher-level parameters.

After the negotiation phase, during service operation (maintenance phase), some system adjustments may be necessary in order to honor the QoS specifications of the admitted flows. The QoS Maintenance meta-service monitors the negotiated QoS parameters and issue alerts to the tuning mechanism when disturbs are detected. The tuning actions may vary from small parameter adjustments in some schedulers to the request of a complete QoS renegotiation in a manner similar to the establishment phase.

2.2 Inter-media Synchronization Meta-Service

Following the model discussed in Section 2.1 to provide inter-media synchronization (the current QoS requirement in focus), the negotiation mechanism will divide the responsibility of providing the inter-media synchronization among the Hypermedia System SOE resources (presentation engine, communication provider, and servers). However, communication providers that support inter-media synchronization are rare. Therefore, only presentation engines and servers are supposed to provide this facility. Nevertheless, since media streams may come from different servers, it is difficult to control inter-media synchronization on server side, except for streams coming from the same station. Therefore, our approach just expects from servers a best effort to maintain this type of QoS requirement for their own flows. As an example, for transmitting pushed data content, a good object-carousel management is expected [11].

Taking into account these assumptions, the presentation engine is the sole component responsible for providing inter-media synchronization, which simplifies the implementation of the Inter-media Synchronization Meta-Service, whose functionalities resumes to the Presentation Engine Controller, as illustrated in Figure 3. Note that this meta-service is usually reflexive, that is, all its components communicate by the same communication provider used by the Presentation Engine’s internal components (Figure 2).

Based on the hypermedia application specification, the Presentation Builder component builds the aforementioned presentation plan. Through the intervention of the Proxy for Context Information (PCI1), in Figure 3, the Presentation Builder gets parameters that define the exhibition context. Exhibition context parameters define user preferences and characteristics, and also the hypermedia SOE platform profile: hardware and software resources available in the client and server sides; performance parameters of the client, servers, and communication providers; etc. In getting this information, the Presentation Builder asks the Adapter component to

---

1 We define the context management as another meta-service in the system, whose role is to maintain information about the exhibition context. For this reason it is illustrated as a separated meta-provider in Figure 3, which is not detailed since it is not the focus of this paper. The Proxy for Context Information acts as a client of the context management meta-service and it is responsible for making a bridge between this meta-service and the intra and inter-media synchronization meta-services.
implement the necessary adaptations, like choosing the best alternative for an object presentation, adjusting the object playing duration, etc. In this phase, the temporal and spatial consistency of relationships defined in the hypermedia document should be checked.

After building the presentation plan, the Pre-fetching Builder component is called to build the pre-fetching plan, which defines when each object should be retrieved in order to keep the inter-media synchronization described in the presentation plan. Pre-fetching mechanisms, as already mentioned, are used to minimize the probability of on-the-fly adaptations.

The pre-fetching plan could be based only on information provided by the presentation plan. However, it is important to take also into account the performance parameters of the Hypermedia System SOE, such as delays and jitters imposed by the network, operating systems, media players, etc. This information can be obtained through the intervention of the Proxy for Context Information (PCI) and from in advance QoS negotiation mechanisms. In this last case, the Intra-media QoS User component is called to start the QoS negotiation for each object to be retrieved.

If service providers only offer best effort services, the Intra-media QoS User component does not exist. In this case, heuristics must be employed to build the pre-fetching plan, based only on assumptions about the SOE performance parameters.

The Pre-fetching Monitor component is in charge of monitoring the real pre-fetching parameters for each object. If a mismatch between these parameters and those used to estimate the pre-fetch time for that object occurs, the component sends a notification to the Pre-fetching Tuner component that will then try to correct the pre-fetching plan. If this correction is not possible, a notification is sent to the Presentation Monitor component.

Besides receiving notifications coming from the Pre-fetching Tuner component, the Presentation Monitor component receives events reported by (media) Players of the Hypermedia System SOE (see Figure 2) during a document presentation, and checks if they match their predicted occurrence-time, defined in the presentation plan. Players must inform the occurrence of presentation events, for example, the selection of a set of information units of a content, the end or beginning of a content presentation, the moment a content is paused or resumed, etc. Based on these events, the whole document presentation is controlled. The Presentation Monitor also listens for any notification coming from the context management (through the PCI component) and from the Intra-media QoS User component.

If any change or violation that impairs the presentation is informed to or detected by the Presentation Monitor, it requests the services of the Presentation Tuner component in order to adjust the presentation plan. At this time, the Adapter component services may be requested. Note that, in this case, adaptations are made on the fly. Sometimes, the Presentation Scheduler component of the Hypermedia System SOE can also be called to actuate on media players to resolve or minimize any problem (for example, to modify content presentation bit-rate or duration).

At any time and for any reason, if the presentation plan is modified, a notification must be issued to the Pre-fetching Monitor, which will actuate on the Pre-fetching Tuner, starting a procedure for adjusting the pre-fetching plan, accordingly.

### 2.3 Intra-media Synchronization Meta-service

The Intra-media QoS User component of the Inter-media Synchronization meta-service is the client of the Intra-media Synchronization meta-service. This meta-service is usually reflexive, as the Inter-media Synchronization meta-service. Therefore, typically, all components of the presentation engine service and meta-services use the same communication provider.

If in any moment the Intra-media Synchronization meta-service does not guarantee the loading of media objects in restricted accordance to the presentation plan, the Intra-media QoS User component is notified and passes this notification to the Presentation Monitor component that asks the Presentation Tuner to try to adjust the presentation plan, as presented in Section 2.2.

### 3 Communication with Player Components

Hypermedia presentations naturally decompose into communicating concurrent components. The timing constraints imposed demand considering aspects mostly related to the speed of communications and computations. This is particularly true when actions are sent and notifications are received from Player components. These aspects stress, mainly, the inter-media synchronization and impact the corresponding meta-service.

Hypermedia presentations are essentially asynchronous and non-deterministic. The reaction time taken between the start of a communication and its actual achievement can be arbitrary and unpredictable. This is especially true in distributed presentations on multiple exhibition devices connected by asynchronous communication means.

In order to continue our discussion, we shall assume a trivial inter-media synchronization example: “when media A ends, media B₁, B₂ … and Bₙ must be started”.

This can be difficult to implement deterministically. The actions (inputs) must be applied in some order, and depending on the reaction times of the players (even if inter and intra-media synchronization meta-services guarantee that the players have already been instantiated and the content have already been retrieved) the order of the results (output) is non-deterministic. For example, some Bᵢ may start before any other, or eventually they all may start together.

The synchronous hypothesis [24, 25, 26] is an idealization of reactive systems in which internal actions and communications are considered instantaneous. The basic
idea is that outputs are produced synchronously (i.e., simultaneously) with their inputs, their reaction taking no observable time.

This hypothesis, which is well-known to make life simpler, is valid in many practical cases. Moreover, synchronous systems, that is, those that assume the synchronous hypothesis, usually decompose better and are easier to describe and analyze than asynchronous systems.

Although a real presentation engine is naturally asynchronous, it can be represented by a synchronous engine with a high frequency tick. Here the notion of “high” depends on the application and on the exhibition platform. For instance, if all actions can be executed during screen (or speakers) refresh time interval, then a synchronous model can be used without impairment. In our experience using the Ginga middleware, almost all applications broadcasted in the Brazilian DTV System allow this approach, since pre-fetching of media content and pre-loading of media players are performed. The exception comes from applications with a large number of actions to be executed in parallel (in our example, when \( n \) is large) or applications targeting multiple exhibition devices. The dilemma is that these applications turn to be more frequent.

In the case we have a large number of actions to be executed, we have two approaches:

1) Apply an action on \( B_{i} \) without waiting for the result of the action applied to \( B_{i-1} \); or
2) Wait for the result of the action applied to \( B_{i-1} \) to apply the action on \( B_{i} \).

The first approach may introduce non-determinism, since a given sequence of inputs can produce different sequences of outputs, depending on the delays of the players.

Since deterministic programs are easier to analyze and debug than non-deterministic ones, hypermedia languages and their engines should be deterministic: a given sequence of inputs, together with its timing, should always produce the same output. Our claim is that one should stay within the ideal synchronous model as much as possible. Therefore, the next version of the Ginga reference implementation will use the second alternative (today, the first alternative is used).

However, the second approach also has its problems. In particular, there are two important cases to analyze:

1) When the delay in communications with media players is irrelevant (zero delay);
2) When this delay is not irrelevant and must be taken into consideration.

In the first case, since players report the result of an action (in our example, when the presentation of each media object really starts), the presentation engine knows exactly the relative time lags between actions that should have occurred at the same time. Therefore, elastic time adjustments can be applied, as discussed in Section 2.2.

In the current Ginga reference implementation we could have used the algorithms proposed in [12] to make the necessary adjustments. However, to decrease the adjustment reaction time it is possible to apply a simpler heuristic:

- If only static media (text, image, etc.) is being presented, then the presentation engine should wait for the notifications of all players before rendering the joint result synchronously. Note that when we have more than one rendering machine (for example, in the case of multiple exhibition device), this is not that easy to implement;
- If continuous media (video, audio, etc.) is being presented, then each media object starts its presentation in its different moment, but adjusted to match the synchronization with the lowest possible cost, as defined by the elastic-time Adapter component of the presentation engine.

The elastic-time Adapter component performs an elastic time computation algorithm. Based on cost functions for stretching or shrinking interval-based events, the algorithm computes the optimal duration that gives the best presentation configuration [12]. In the NCL language, for example, costs are described as two linear functions (one for stretching and other for shrinking) and the optimization is calculated using tension graphs, derived from each time chain [12]. However, the NCL profile for DTV supported by Ginga does not allow for defining interval based durations. Thus, only default assumptions are supported by Ginga.

This first case solution for the second approach satisfies all applications we have tested for a single exhibition device, in which the communication between the presentation engine components can indeed be considered instantaneous. However, this assumption fails when components communicate through an asynchronous network.

In the case of multiple exhibition devices, in spite of all QoS care discussed in Section 2.2 and 2.3, the non-deterministic and large delay imposed by the communication provider prevents the use of the synchronous approach.

When the additional content presented in a secondary device does not have a strict synchronization with the content being presented in other secondary or primary devices, the synchronous model can still be used by each device, individually. Fortunately, this is the case of a great number of applications that only plays additional information on secondary device without any synchronization with media content presented in other devices, unless that the additional content must be started after some moment established by the primary device.

However, if inter-media synchronization is necessary between content running on different devices, some known asynchronous communication mechanism must be used to maintain synchronization.

We have not devised a concrete solution for the Ginga middleware yet, but we believe that the language used to
develop applications (in our case NCL) should distinguish synchronous actions from asynchronous actions.

In a centralized system, the solution is not difficult, since system time may be dictated by the central component, and synchronization can be attached to this time. Cristian’s algorithm [13] and the Berkeley algorithm [14] are some solutions to the clock synchronization problem in a centralized environment, with a good result in low-latency communication provider (like a local network), in which the round-trip time of the request is short compared to required accuracy. In a distributed system the problem is harder because a global time is not easily known.

4 RELATED WORK

Several strategies for building the presentation plan were proposed in the literature [15, 16, 17, 18, 19]. The Firefly system [15] is one of the pioneers. Firefly introduced the idea of temporal chains (one main schedule and zero or more auxiliary schedules). The Firefly compiler builds temporal chains and computes the expected time for each event occurrence based on author temporal constraints and on each linear cost function, established for each media object, describing the cost for stretching or shrinking the content duration. Once initiated the presentation, the execution module starts a scheduler and an event handler. These two components play roles similar to the presentation scheduler and presentation monitor components presented in Section 2, respectively.

Madeus [16] offers temporal flexibility through media object duration specifications, defined as an interval, establishing a lower and an upper limit value. Madeus builds a temporal graph, named HSTP (Hypergraph of Simple Temporal Problems) before starting the presentation. This graph can be considered as the Madeus presentation plan. During the document presentation, the formatter monitors the effective duration of each media object and compares them with the expected values. If any deviation that can cause a future synchronization lost is perceived, the Madeus applies an adjustment algorithm, but without considering optimization metrics. The main goal is to quickly find a new presentation arrangement that satisfies the constraints on duration intervals.

The Ambulant SMIL player [17] begins its process parsing the document specification and building its DOM tree. Each node in the tree, which represents a media object (image, text, audio, etc.) or a SMIL composite element (par, seq, or excl), is controlled by a data structure called time node. Edges in the tree preserve temporal relationships among composites and their children, as defined by the composite’s semantics. Other relationships among time nodes are represented by semantic links, in addition to the tree edges. The combination of both relationships forms the complete time graph.

Ginga-NCL player’s procedure is similar to the technique used by Ambulant. NCL context (composition) nodes group media objects and other context nodes, recursively. However, NCL contexts have no predefined semantics. Instead, they include <port> and <link> elements defining spatiotemporal relationships among context’s children. Similar to SMIL time nodes, each NCL node has an associated state machine, which stays in a particular presentation state depending on the document presentation flow.

In both NCL and SMIL, the presentation flow starts from the root node. From this moment on, relationships defined among composition’s children are triggered in the specified relative moments, changing presentation states of related nodes.

The distributed data structure of Ginga-NCL for the presentation plan is not efficient to derive the moment a pre-fetching should be done (or when QoS negotiation for data transfer must be started), usually requiring the simulation of the document presentation flow, which is almost always inefficient. Aiming to bypass this problem, the next version of Ginga bases its presentation plan on a temporal graph data structure called Hypermedia Temporal Graph (HTG) [19]. HTG represents, in a unique digraph structure, all relationships among presentation states of all media objects that compose a document, instead of having this relationships distributed and embedded in composition elements. HTG represents all predictable and unpredictable events that can cause changes in the presentation state.

HTG concept is similar to Firefly’s temporal chains [15]. However, unlike Firefly’s temporal chains, HTG is not a timeline of presentation actions and indeed represents all presentation possibilities for a document. In addition, instead of adding branches (auxiliary chain) to the graph (the main chain), according to the document presentation flow, branches are pruned from the HTG graph, when alternatives or unpredictable events cannot occur anymore. From HTG the presentation plan can be derived [15].

Some strategies for building the pre-fetching plan were proposed in the literature [20, 21, 22, 23].

In Jeong et al. [20], the pre-fetcher component estimates the maximum time for recovering the entire content of each document object and builds a pre-fetching plan in order to have the content of all objects before their playing time. The algorithm postpones the start of the document presentation as a whole, in order to avoid gaps and loss of synchronization during the presentation. The presentation plan, used as input for the pre-fetching plan calculation, is based on the partition of media objects in minimum contiguous segments. In doing that, the main contribution of that work is the implementation of a strategy for building a plan to pre-fetch each partition. However, the proposal only considers documents exclusively based on predictable relationships and predictable media object durations.

Still in Jeong et al., during document runtime phase, there is a monitor that compares the object actual pre-fetching duration with the expected duration predicted in the plan. If the actual duration exceeds the predicted one, the system runs an instant scheduling algorithm for recalculating the object presentation duration, in order to maintain the temporal segment synchronization. When necessary, the algorithm sacrifices static media objects,
shrinking or stretching their durations. However, only the presentation plan is adjusted at runtime; no adaptation is done in the previously built pre-fetching plan.

In [21], Feng-Cheng Lin et al. explore techniques for optimizing the object sequence for an auto-assembled multimedia presentation to minimize the overall presentation lag. They adapt techniques developed in conventional two-machine flowshop research for computing or approximating the optimal sequences. A variety of settings for commonly found applications with auto-assembled multimedia presentations are discussed. In [22] the work continues, focusing on the scheduling of media objects in a delay-prone network environment such that the overall presentation-lag and the due date penalties can be minimized. The sequencing problem is treated as a flowshop scheduling problem to which a reduction strategy with a branch and bound algorithm to derive optimal sequences is presented.

The current Ginga reference implementation provides a very simple strategy for pre-fetching, since, initially, the implementation only targeted broadcasted applications in which all media objects are previously transmitted by object carousels and are stored before they are needed. The responsibility of managing object carousels [11] is delegated to the broadcasters. However, the HyperProp system, predecessor of Ginga, had some simple pre-fetching polices implemented.

In HyperProp, the Pre-fetching Builder component first estimates, for each media object (actually each interval-based event corresponding to a media object presentation), the content preparation time. This is done with simple heuristics using information coming from the media object (content size that needs to be buffered and maximum presentation lag accepted for this object) and from the exhibition context manager (platform mean transfer rate, platform delay and, for unpredictable events that begins a new auxiliary time chain, the occurrence probability). Afterwards, the Pre-fetching Builder joins all this information and establishes the media object pre-fetching sequence, assigning the expected beginning time and duration for each content loading.

HyperProp pre-fetching implementation offers two compiling strategies. One strategy serializes the content preparations so that each object is programmed to be pre-fetched at the same time, one of them is pre-fetched first, since there is no pre-fetching superposition. The pre-fetching plan is built from the end to the beginning of the presentation plan, having, at the end of the process, an estimative of the minimum presentation initial delay to avoid synchronization losses. The other pre-fetching strategy uses a greedy approach. It simply subtracts from each object starting time the corresponding estimated pre-fetching duration and calculates the initial delay from the difference between the first document object to be played and the first document object to be prepared.

5 CONCLUSIONS

The main contribution of this paper is the discussion raised from the framework for the integrated handling of the inter-media and intra-media synchronization as a QoS orchestration problem. The structure suggested uses a recurrent approach, bringing to a higher abstraction level the same concepts found in traditional infrastructures (network and operating systems).

We intend to explore our framework in the next version of the Ginga reference implementation to test the several algorithms involved: elastic-time adaptation, context-based adaptation, pre-fetching plan compilation, pre-fetching plan scheduling (to best effort platforms and to providers supporting QoS reservation and in advance reservation), pre-fetching plan adjustments, and presentation plan tuning.

In the paper, we also introduce the synchronous model as an idealization of hypermedia systems where internal actions and communications are instantaneous. In Section 3, we discuss when such assumption is reasonable for some real scenarios. Synchronous programs decompose well and turn out to be easier to describe and analyze than asynchronous ones. We are already working on trying to better refine where the synchronous model can be used, and also looking for algorithms to analyze hypermedia specifications and to detect causes of non-determinism, causal paradoxes, and inconsistencies. On the other hand, we have already started an implementation supporting multiple exhibition devices that takes into account the use of the asynchronous communication model, at first using the simple centralized algorithms mentioned in Section 3.

Acknowledgements

This work was partially supported by the Brazilian CNPq, CAPES and FAPERJ funding agencies, and by the Brazilian Ministry of Science, Research and Innovation.

References


