Models and operators for extension of active multimedia documents via annotations

Diogo Santana Martins

Models and operators for extension of active multimedia documents via annotations

Diogo Santana Martins

Advisor: **Profa. Dra. Maria da Graça Campos Pimentel**

Doctoral dissertation submitted to the *Instituto de Ciências Matemáticas e de Computação* - ICMC-USP, in partial fulfillment of the requirements for the degree of the Doctorate Program in Computer Science and Computational Mathematics. *FINAL VERSION*.

USP – São Carlos
January 2014

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Martins, Diogo
M379m Models and operators for extension of active multimedia documents via annotations / Diogo Martins; orientador Maria Pimentel. -- São Carlos, 2014.
201 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2014.

1. multimedia. 2. document engineering. 3. authoring methods. 4. multimedia documents. 5. temporal layout. I. Pimentel, Maria, orient. II. Título.

Modelos e operadores para extensão de documentos multimídia ativos via anotações

Diogo Santana Martins

Orientadora: **Profa. Dra. Maria da Graça Campos Pimentel**

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação - ICMC-USP, como parte dos requisitos para obtenção do título de Doutor em Ciências - Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

USP – São Carlos
Janeiro de 2014

Acknowledgments

Primarily, I acknowledge my advisor, Prof. Maria da Graça C. Pimentel, for the continuous support and guidance during the development of this dissertation. Thanks for helping me make this dissertation possible. I also thank CNPq for financing my research.

Thanks to the evaluation committee for the important insights that contributed to improve the final version of this text.

Thanks to research colleagues that have collaborated, directly or indirectly, to the discussions and experiments that lead to this research: Didier Vega-Oliveros, Lílian Oliveira e Diogo Pedrosa.

I extend acknowledgements to the researchers from the SEN-5 group at CWI, who, during my internship with them, have contributed in important ways to my research: Pablo Cesar, Rodrigo Guimarães, Jack Jansen e Dick Bulterman.

Finally, I thank my family for the support through this journey. First, my mother, Arlete, for standing up for me along my entire academic career and believing in my choices. My grandmother, Seide, whose early support in my education made it possible that I reached this milestone. In addition, Mario Liziér, for the understanding, support, and encouragement whenever I needed.

Abstract

Multimedia production is an elaborate activity composed of multiple information management and transformation tasks that support an underlying creative goal. Examples of these activities are structuring, organization, modification and versioning of media elements, all of which depend on the maintenance of supporting documentation and metadata. In professional productions, which can count on proper human and material resources, such documentation is maintained by the production crew, being key to secure a high quality in the final content. In less resourceful configurations, such as amateur-oriented productions, at least reasonable quality standards are desirable in most cases, however the perceived difficulty in managing and transforming content can inhibit amateurs on producing content with acceptable quality. This problem has been tackled in many fronts, for instance via annotation methods, smart browsing methods and authoring techniques, just to name a few. In this dissertation, the primary objective is to take advantage of user-created annotations in order to aid amateur-oriented multimedia authoring. In order to support this objective, the contributions are built around an authoring approach based on structured multimedia documents. First, a custom language for Web-based multimedia documents is defined, based on SMIL (Synchronized Multimedia Integration Language). This language brings several contributions, such as the formalization of an extended graph-based temporal layout model, live editing of document elements and extended reuse features. Second, a model for document annotation and an algebra for document transformations are defined, both of which allows composition and extraction of multimedia document fragments based on annotations. Third, the previous contributions are integrated into a Web-based authoring tool, which allows manipulating a document while it is active. Such manipulations encompass several interaction techniques for enriching, editing, publishing and extending multimedia documents. The contributions have been instantiated with multimedia sessions obtained from synchronous collaboration tools, in scenarios of video-based lectures, meetings and video-based qualitative research. Such instantiations demonstrate the applicability and utility of the contributions.

Keywords: multimedia, document engineering, authoring methods, multimedia documents, temporal layout, document transformations, authoring tools

Resumo

Produção multimídia é uma atividade complexa composta por múltiplas atividades de gerência e transformação de informação, as quais suportam um objetivo de criar conteúdo. Exemplos dessas atividades são estruturação, organização, modificação e versionamento de elementos de mídia, os quais dependem da manutenção de documentos auxiliares e metadados. Em produções profissionais, as quais podem contar com recursos humanos e materiais adequados, tal documentação é mantida pel equipe de produção, sendo instrumental para garantir a uma alta qualidade no produto final. Em configurações com menos recursos, como produções amadoras, ao menos padrões razoáveis de qualidade são desejados na maioria dos casos, contudo a dificuldade em gerenciar e transformar conteúdo pode inibir amadores a produzir conteúdo com qualidade aceitável. Esse problema tem sido atacado em várias frentes, por exemplo via métodos de anotação, métodos de navegação e técnicas de autoria, apenas para nomear algumas. Nesta tese, o objetivo principal é tirar proveito de anotações criadas pelo usuário com o intuito de apoiar autoria multimídia por amadores. De modo a subsidiar esse objetivo, as contribuições são construídas em torno uma abordagem de autoria baseada em documentos multimídia estruturados. Primeiramente, uma linguagem customizada para documentos multimídia baseados na Web é definida, baseada na linguagem SMIL (*Synchronized Multimedia Integration Language*). Esta linguagem traz diversas contribuições, como a formalização de um modelo estendido para formatação temporal baseado em grafos, edição ao vivo de elementos de um documento e funcionalidades de reúso. Em segundo, um modelo para anotação de documentos e uma álgebra para transformação de documentos são definidos, ambos permitindo composição e extração de fragmentos de documentos multimídia com base em anotações. Em terceiro, as contribuições anteriores são integradas em uma ferramenta de autoria baseada na Web, a qual permite manipular um documento enquanto o mesmo está ativo. Tais manipulaçoes envolvem diferentes técnicas de interação com o objetivo de enriquecer, editar, publicar e estender documentos multimídia interativos. As contribuições são instanciadas com sessões multimídia obtidas de ferramentas de colaboração síncrona, em cenários de aulas baseadas em vídeos, reuniões e pesquisa qualitativa baseada em vídeos. Tais instanciações demonstram a aplicabilidade e utilidade das contribuições.

Keywords: multimídia, engenharia de documentos, métodos de autoria, documentos multimídia, formatação temporal, transformações em documentos, ferramentas de autoria

Contents

List of Tables

List of Figures

List of Listings

List of Algorithms

Chapter 1

Introduction

1.1 Context and motivation

The prominence of user-generated content[1] on the Web — e.g. in the form of video, audio, presentations, among others — has created strong demands for research on methods to support amateur production of multimedia content. To a lesser extent, such demands have been tackled by advancements in capture, editing and communication technologies, all of them associated to the abundance of digital repositories and the incentives for sharing in social networks. Notwithstanding these improvements, when amateur productions are contrasted with the material and human resources of professional productions, it is observed that the majority of amateur producers lack time and technical abilities to create digital multimedia content of considerable quality [Engström et al., 2010; Kirk et al., 2007]. Such situation, to a greater extent, contributes with several opportunities and challenges on the investigation of computational methods to support amateurs on bridging the quality gap in multimedia production.

Of all tasks comprised in multimedia production processes, the ones related to organization, modification and versioning of media elements are compelling candidates for research with a focus on document engineering. This is because these particular tasks are highly influenced by the availability and maintenance of supporting documents and tools. Taking as an illustration of this influence the process of professional multimedia production, it is observed that various documents that support the process are key to secure the quality of the final product. Examples of these documents are scripts, edit decision lists, logging of shots, all of which, maintained and created by the production crew, document and promote the evolution of the content from conception to distribution [Nack, 2005]. Naturally, the expectations and requirements of amateur productions are quite different from professional counterparts but, on the other hand, the possibility of counting with at least part of these documentation resources has great potential for enhancing selection, revision and reuse of media elements. Even though efficient methods for helping the creation and management of these resources would be beneficial for any production configuration,

[1] User Generated Content, or UGC, is broadly defined as any content which is authored by amateur individuals, differently from professional productions originated from mass media or qualified content producers.

in the case of amateur configurations this requirement is more critical. This is because the cost of crafting them could be prohibitive in the absence of computational methods designed to facilitate their creation [Perry et al., 2009].

The perceived quality of multimedia content is influenced by the resources and techniques employed in the various stages of multimedia production (e.g. pre-production, capture, post-production and so on). In professional video productions, for instance, it is common to employ proper human and material resources to secure the quality of the final product, for instance by employing high-end capture devices, controling environmental variables (e.g. lightning, background, ambient sound, make up, costumes, positioning, and so on), and dedicated post-production equipment and crew. Amateur video production, on the other hand, rarely count on such human and material resources, on many cases being a production conducted by a untrained single individual, which is concentrated on documenting events in uncontroled environments, generally using consumer-level equipment. The differences between amateur and professional content are generally easily detected by humans and, in some cases, via algorithms (e.g. via classification over features such as motion, color, texture and structure) [Guo et al., 2013]. Such discrepancies can also be generalized, for the same reasons, to other modalities, such as image production, audio production, and so on.

High quality content is desirable at any point in the spectrum of amateur production, but in certain contexts quality requirements deserve more attention than in others. At one extreme of the spectrum, which we classify as *amateur-personal*, occurs informal recording of everyday experiences, for instance in the context of families and groups of friends, situations in which people might have little interest, inspiration or incentives to use editing tools [Engström et al., 2012]. On the other extreme of the spectrum, which we classify as *amateur-professional*, the communication conveyed via multimedia content occurs in a more structured manner and generally with a longer duration. Examples of amateur-professional production are publication of lectures [Tung et al., 2011], technical talks [Adcock et al., 2010], independent and collaborative journalism [Lindstedt et al., 2009], multimedia-based qualitative research [LaRosa et al., 2009], among others. In these cases, the difficulty to organize and to modify in fine granularity the captured information might be an impediment to enable proper communication of ideas or to keep the attention of the audience. A solution sought for many amateur-professional productions is to compromise part of the production budget on hiring a multimedia specialist. But for the cases where such ideal conditions are not met, the availability of methods for easier authoring has potential to not only improve the quality of the material currently produced, but also to engage more people willing to produce and publish content.

Recurrent themes in amateur-professional recordings are situations of interaction between small groups, for instance: interviews and debates, in the journalism domain; lectures, webconferences and talks, in the education domain; and observation of human behavior, individually or in groups, in the qualitative research domain. Such recurrent thematic on activity and communication creates several opportunities for taking advantage of the dynamics of these interactions

for helping to document, manage, navigate, reuse and extend the underlying multimedia content. The attainment of these requirements is convergent with research efforts concerned with improvements in the documentation and authoring of multimedia information. From the standpoint of documentation, much research has been concerned with annotation of multimedia content, tackling problems such as interaction techniques for multimodal annotation [Goularte et al., 2004], models of annotation [Haslhofer et al., 2012], and techniques for navigating multimedia collections based on annotations [Tur et al., 2010]. If the problem of multimedia annotation is constrained to the scope of multimedia production, annotations can be defined as informational units that document and enrich media assets generated along a production process. From the standpoint of editing, research on methods for multimedia authoring (either manual [Meixner et al., 2012], semi-automatic [Liang et al., 2013] or automatic [Zsombori et al., 2011]), for instance, investigate the orchestration of spatial, temporal and semantic aspects involved in the composition of multimedia content with variable user intervention. Both themes, annotation and authoring, are closely related, since, to a great extent, improvements in authoring methods take advantage of the availability of annotations obtained manually or automatically.

1.2 Research problem and approach

The perceived difficulty in managing and transforming content can inhibit amateurs on producing content with acceptable quality. For that reason, two challenges are emphasized here to properly support amateur production:

***Challenge i*)** How to assist amateurs to document the production process without incurring in costly interactions;

***Challenge ii*)** How to help amateurs to take advantage of these documentation resources to analyze, modify and version multimedia content.

The first stated challenge could be partly solved via automatic annotation methods, for instance encompassing annotation of high level semantic events in the multimedia streams. However, despite the limitations and low generalization of these methods, there are also situations that only manual annotations will suffice. This is the case of subjective comments which a user/producer might apply over the content as a way to curate it, for instance expressing opinions and insights. In summary, for the problem of documenting an amateur production, on one hand there is the difficulty of automatically annotating multimedia streams and, on the other hand, the need to have enough flexibility and simplicity to accommodate manual annotations as well.

Accommodating manual annotations can be tackled, for instance, via a flexible annotation model and adequate interaction techniques. Obtaining automatic annotations, on the other hand, constitute a more complex problem. Certain types of productions attempt to manage this complexity by focusing on domain-specific abstractions. This is the case of recordings of

small group activities, in which a recurrent alternative for documenting the captured information is indexing interaction events that are typical in these scenarios. The level of abstraction of the indexed interactions can vary from low-level physical actions (e.g. change of slides, live annotations, speech turn taking, primitive gestures, addressing, nodding, etc.) [Koyama et al., 2010; Terken and Sturm, 2010] to high-level behaviors (such as decisions, disagreement, dominance, extroversion, social roles, competition, personality states, etc.) [Gatica-Perez, 2009; Popescu-Belis et al., 2012]. Naturally, lower level interaction events are easier to detect, whereas higher level events might demand more complex methods. Additionally, in many of these recognition tasks accuracy and generalization are important issues, as far as application to multiple situations is concerned.

Aside these difficulties, the utility of interaction events as cues for representing key moments of a recording has been positively assessed by several user studies in different small group configurations, such as meetings [Nathan et al., 2012; Popescu-Belis et al., 2012] and webcasts [Dufour et al., 2011]. These results demonstrate that focusing on the semantics of the interactions between participants of a captured experience can be a feasible alternative to document the experience itself. Therefore, these approaches provide alternatives for "logging" a multimedia production process with minimal user intervention, i.e., automatic documentation as a natural consequence of the underlying activity.

Once a user/producer counts on useful annotations associated to the multimedia content, what remains is how to take advantage of these annotations in producing content, which configures the second challenge in the aforementioned ones. A direct alternative would be to manually assemble the annotations from various sources, load and align them with multimedia assets into a full-featured authoring tool and, finally, start a long process of selecting, fragmenting and composing the content until a product with acceptable quality is obtained. However, as discussed earlier, most amateurs are unwilling to engage even in simple authoring processes, for a lack of time, ability or interest. Consequently, even the preliminary task of gathering all the necessary information is likely to be avoided. Therefore, the general challenge of authoring via annotations encompass, among others, the following problems:

Problem i) How to assist the use of annotations for authoring purposes in a natural manner;

Problem ii) How to provide means to efficiently analyze the captured information and make editing decisions, using the annotations;

Problem iii) How to support easy enrichment and editing of the captured information, taking annotations as starting point.

In order to tackle these issues, interactive multimedia documents (iMMDs), especially those specified via declarative languages, constitute a compelling foundation. Interactive multimedia documents are used to represent, in a specific authoring language, a complex multimedia scenario. On problem *(i)*, most iMMD languages provide declarative constructs to associate

documentation content to a document in fine granularity. On problem *(ii)*, the interactive nature of multimedia documents support the construction of efficient browsing interfaces that allow non-linear exploration of composite presentations. On problem *(iii)*, declarative multimedia languages, as long as they properly support reuse features, can significantly ease transformation and versioning of documents in a non-destructive, non-duplicating manner. Provided these advantages, in this dissertation iMMDs are used as an general approach to construct a group of solutions, which are detailed as follows:

Solution i) Automatically associate to iMMDs annotations generated along the production process. This solution involve models and strategies to properly encode annotations of interest and associate them to multimedia documents in adequate granularity;

Solution ii) Generate flexible iMMD-based exploratory browsing interfaces for captured content. iMMDs provide synchronization constructs that allow the orchestration of multiple media elements in a single presentation, which is the case of the scenarios tackled in this dissertation. Additionally, the interactivity resources, such as linking and time-oriented access, provides a foundation to automatically generate access interfaces to captured information. These interfaces are used for analyzing captured content alongside annotations, thus subsidizing sensemaking and decision making regarding what to include in a final content piece;

Solution iii) Support transformation of content based on annotations. The formal semantics of multimedia documents is an important enabler to define robust transformation mechanisms over composite presentations. Such transformation mechanisms are used for providing simplified, abstract editing operations over composite presentations.

In order to realize these general solutions, this dissertation brings several contributions to multimedia authoring, in general, and to engineering of multimedia documents, in particular. In the following Section, these contributions are discussed in more detail.

1.3 Contributions

The primary objective of this dissertation is the realization of an annotation-centered approach to enrich and extend multimedia content under a document engineering perspective. Realizing this objective involves the orchestration of several individual contributions which support amateur-oriented multimedia production:

- **A concrete model for editing of active multimedia documents**.

  Modifying a multimedia document while it is active is an key feature to support annotation-centered enrichment and editing tasks by amateurs. By using live editing operations,

users can immediately perceive the effect of the annotations and editing operations on the document, which significantly improve user experience. In this dissertation it is defined the ActiveTimesheets language, based on the SMIL (Synchronized Multimedia Integration Language), which provides a number of extensions to tackle limitations in its base language. Among the most important contributions of this language, and of its implementation as a temporal layout engine as well, is a live editing model, i.e., a method for editing a document while it is active. Incorporating live editing features in the SMIL language, which has an elaborate temporal model, is not a trivial task. To tackle this problem, ActiveTimesheets, first, subsets the SMIL language to encompass a group of language constructs that are critical to dynamic Web-based multimedia applications, in general, and to the problems dealt with in the dissertation, in particular. After subsetting the language, a formalization of a graph-based model for temporal layout is provided. The mentioned layout model, called timegraph, is not a novel contribution, provided that it has been vaguely discussed before in the literature. However, the high degree of formalization of all the underlying processes and algorithms deserve mention as a contribution to support future work on this issue. Based on this temporal layout model, a group of live editing operations is defined, encompassing manipulation of every element and attribute of the ActiveTimesheets language. A group of algorithms is defined to efficiently tackle the effect of every editing operation on the underlying temporal layout of the document.

- **Reuse features in multimedia documents**.

  Reusing content or fragments of content is at the core of multimedia production. In the general editing case, users may need to include external content in a presentation in order to convey a message. In the case of enrichment operations, applying additional content over an existing document can also be seen as an instance of reuse but, in this context, for purposes of versioning. The lack of reuse features in multimedia documents can make such activities more difficult to realize, for instance by requiring duplication of declarative code that leads to verbosity and, consequently, decreased manageability of a document. Several multimedia languages provide reuse features in fine granularity. In SMIL, such features are present, but only to a limited extent. ActiveTimesheets extends the reuse features in SMIL by including reuse of document elements and element fragments, two important features for supporting fine-grained enrichment of multimedia documents. These novel features are demonstrated from the standpoint of new syntax constructs and their impact to the presentation-related data structures.

- **Infrastructure for synchronized multimedia on the Web**.

  Distribution of multimedia content on the Web has been traditionally associated with self-contained execution environments loosely coupled to user agents as plug-in components, all of which are known for their performance and integration problems. Currently, native multimedia execution is widespread in modern user agents, creating many opportunities

for designing native Web-based multimedia applications. Language constructs for synchronized multimedia presentations, on the other hand, are still a gap in Web-native multimedia technologies, aside from scripting-based execution environments. The ActiveTimesheets engine, an implementation of the ActiveTimesheets language, brings contributions to the technological spectrum of synchronized multimedia on the Web. Besides allowing the execution of distributed multimedia presentations with live editing and reuse features, it provides several important extensions, such as an extended linking model which is closely integrated with media fragments. These extensions are discussed with their associated algorithms.

- **A model for tracking and binding interaction events in the production process**

  An important issue when taking advantage of annotations collected along the multimedia production process is how to represent and manage them. This dissertation contributes with a conceptual model, the I+WaC-IE (Interactor+WaC Interaction Events) model, which abstracts the various collected annotations as interaction events. The abstractions provided by this model allow the representation of the multiple dimensions of an interaction event. Additionally, the concept of an annotation operator is defined, which is used as an abstraction to associate interaction events to content fragments. Via these models and mechanisms, annotations can be managed along the production workflow and used for authoring tasks.

- **Operators for transformation of multimedia documents**.

  One of the challenges that motivate the dissertation is how to support authoring tasks via annotations. An important contribution toward tackling this challenge is the formal definition of a family of operators to transform multimedia documents: the Interactors Algebra for Multimedia Document Transformations (IAMmDocT). Taking as input groups of annotations, the operators manipulate a base multimedia document in order to generate new versions. These operators contribute with editing operations such as joining fragments in the temporal scope, constraining a new version to specific media elements and extracting fragments of interest from the presentation. The most important advantage of these operators is that all operations are based on annotations, i.e., an editing result is a consequence of manipulating annotations. The integration of the operators with the ActiveTimesheets language is demonstrated as an example, which also reinforces the relevance of the extensions provided by this language.

- **Infrastructure and interfaces for annotation-centered browsing of captured information**.

  The concrete integration of ActiveTimesheets and IAMmDocT is realized in an Web-based infrastructure and tool, the I+WaC Editor. This client-server tool allows users to non-linearly access a multimedia recording by visualizing and browsing annotations. Regarding

browsing functionality, the IAMmDocT operators are used to provide a dynamic, query-based visualizations of the annotations, so that users can properly analyze the recording to make decisions. Additionally, the tool includes a number of browsing enhancements, such as adaptive playback based on annotations.

- **Infrastructure and interfaces for amateur-oriented authoring via annotations.**

  Besides taking advantage of existing annotations, the I+WaC Editor also provides functionality for modifying and creating new annotations and interactivity behaviors in the underlying document, all taking advantage of live editing. As a consequence, users can modify an annotated document and perceive the changes immediately. Annotations in the I+WaC Editor can assume interactivity behaviors, which also extend their applicability for authoring purposes. Versioning a session in this tool occur by executing transformations in the document using available annotations. Such transformations are mapped to high-level interaction techniques that, at a lower level, are realized via the IAMmDocT operators.

1.4 Organization of the dissertation

Chapter 2 overviews the theoretical foundation of the dissertation research, which encompass multimedia production processes and a background on previous work upon which the research reported in this dissertation has been built;

Chapter 3 reports the document engineering contributions of the dissertation, comprising the definition of the ActiveTimesheets language, its semantics and processes, methods for editing of active multimedia documents, extensions for reuse and extended linking of media elements;

Chapter 4 covers the multimedia authoring contributions of this dissertation, which includes annotation and authoring operators over multimedia sessions, comprising the I+WaC-IE conceptual model, as well as annotation, selection, fragmentation and composition operators.

Chapter 5 discuss I+WaC Editor as infrastructure and tool support designed to support the contributions reported in the previous chapters. After the infrastructure and tools are reported, an instantiation regarding multimedia support for video-based qualitative experiments is discussed, followed by an instantiation to support extension of multimedia material from web-based lectures.

Chapter 6 discusses the contributions and limitations of the dissertation, pointing out ground for future research.

Chapter 2

Background

This chapter provides an overview of multimedia production (with an emphasis on amateur processes) and delineate the main stages of these processes (Section 2.1). Requirements are identified for bridging the gap between professional and amateur production, by making it easier for amateurs to produce content. Important tasks of multimedia production that are relevant to this dissertation are discussed in more detail, regarding multimedia annotation (Section 2.3), browsing (Section 2.4) and authoring (Section 2.5), this last theme encompassing an extended discussion on interactive multimedia documents (Section 2.6). After, requirements derived from previous research are contextualized, discussed and compared with the contributions of this dissertation (Section 2.7). Finally, the chapter concludes with an analysis of how all background issues are related to the dissertation (Section 2.8).

2.1 Multimedia production processes

Traditional processes of multimedia production, such as those observed in TV and cinema, are generally regarded as professionally-oriented workflows, so as they do not directly encompass the context of amateur producers. Professional media production follows a generally agreed upon process roughly composed of four stages [Musburger and Kindem, 2009; Nack, 2005] (Figure 2.1):

- *pre-production*, in which the logic and main ideas of the production are conceived, generating artifacts such as scripts, treatments, synopses and storyboards;
- *production*, in which the actual shooting takes place and that generates, besides the media assets or essence, also artifacts such as plots of actions and scenes, camera placement annotations, director editing decisions, continuity plans and annotations automatically generated by the capture devices (e.g. low-level media features);
- *post-production*, in which the produced media assets are orchestrated in order to convey a message, generating artifacts such as editor notes, edit decision lists (insertion, deletion,

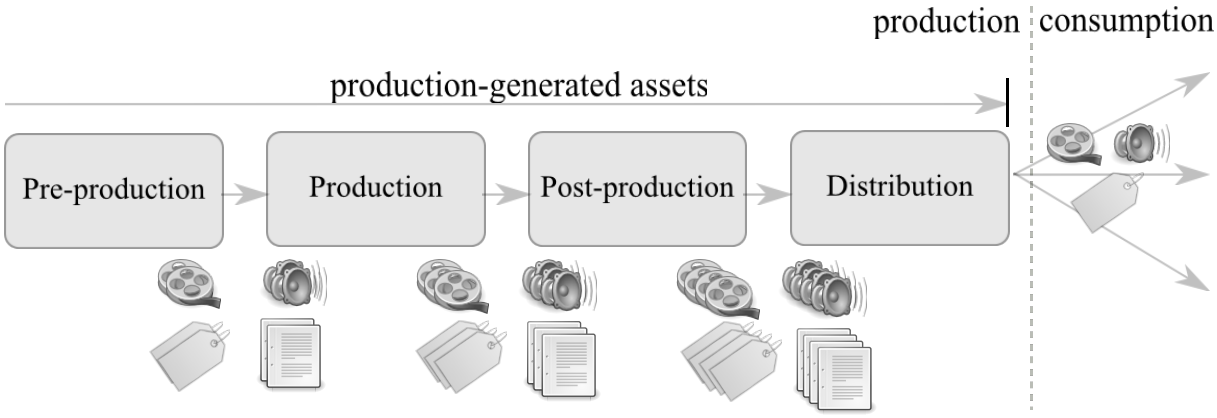


Figure 2.1: Traditional process of multimedia production

substitution and permutation of fragments) and groupings of spatio-temporally related shots; and

- *distribution*, in which the final product is published through the various distribution channels, being the product generally described by global annotations to enable the users to find (e.g. via descriptive metadata) and navigate (e.g. via tables of contents) the media.

This classical process has served well the mass media industry for several years with considerable success. However, when end-users are regarded not only as consumers, but also as producers of multimedia content, the feasibility of directly transferring it to amateur production must be adequately thought about. This is due to the fact that traditional processes entail some implicit assumptions regarding the role of end-users:

i) the user, from the standpoint of production, is regarded only in a latter stage of the process, namely after the distribution. This assumption largely ignores the potential role of consumers on enriching and aggregating added-value content to media products, as long as users have the possibility to interact with multimedia content from several channels (e.g. via interactive TV, Web, mobile applications, etc.);

ii) the process is characterized by the capture and transfer of metadata along the several stages, but the majority of this metadata is retained with the producers and not made available to users after distribution. Naturally, much of such documentation is related to the creative process of the product, thus not being relevant to the majority of users who will consume the content in a lean-back attitude. But, in situations where the user adopts a lean-forward attitude towards the content, the availability of documentation about semantic units of a multimedia asset would be of considerable value for optimizing these interactions.

In face of these issues, understanding end-user multimedia production processes entail not only alternatives for creating multimedia content from scratch, but also for reusing and

customizing existing (either professional-generated or amateur-generated) multimedia content. Several models for amateur production have been proposed to analyze these issues. For instance, Hardman et al. [2008] propose a configurable model of multimedia production composed of nine canonical processes (premeditate, create, annotate, package, query, construct message, organize, publish and distribute) that can be orchestrated in order to describe more complex processes (e.g. both in professional [Rijsselbergen et al., 2008] and amateur [Kopf and Effelsberg, 2008] settings) with anticipated opportunities for capture and transfer of metadata. However, the model inherits some limitations of the classical production model, namely it doesn't anticipate the potential of end-users to enrich media assets after the distribution of the content.

The process proposed by Cesar et al. [2009], on the other hand, is fully oriented towards user enrichment. The authors propose four stages (fragment, tag, enrich and send) that are validated in several case studies regarding enrichment (e.g. text/audio annotations and temporal links) of broadcasted TV content and the automatic authoring of multimedia documents for differentiated viewing. Even though the process tackles user-led enrichment, it focuses mainly on enrichment of professional content and does not anticipate the possibility of iterative enrichment over user-generated content.

Kirk et al. [2007], based on several user studies in home settings, model a process for user-generated (video) content encompassing four stages (pre-capture, at capture, post-capture and end use). Although their process describes user-generated content workflows with fine granularity, it lacks treatments on the enrichment of professionally-generated content and does not tackle issues of capture and transfer of metadata.

A direct comparison between these processes suggests two fundamental approaches end-users employ to extend content: *enrichment*, or enhancement, and *editing*, or customization, as briefly compared and exemplified in Table 2.1. An immediate consequence of enriching a media asset is that the original narrative keeps largely unaltered, being the enrichment just a "layer" over the original content. On the other hand, as a consequence of editing, the narrative or message originally conveyed by the content can be significantly altered as a result of operations.

These models demonstrate several important aspects of amateur multimedia production which, associated to user-friendly methods to assist authoring and customization of multimedia content, allows to advocate a production process that fulfills the following requirements:

1. promotion of end-users as first-class nodes in the multimedia production process. This requirement entails two facets:

   (a) assistance to end-users on the task of enriching professionally generated content. Mass media content is very unlikely to disappear, provided its high quality and societal importance [Cesar and Chorianopoulos, 2009]. Therefore, end-user engagement to annotate and to re-purpose professional content, for instance for sharing purposes, is a considerable aspect of user-oriented editing workflows;

Table 2.1: Versioning and transforming multimedia: enrichment and editing operations.

Category	Definition	Example operations
Enrichment	Layering additional content or interactive resources over source content.	• overlaid annotations in the spatial scope (e.g. hotspots and digital ink) [Cabral et al., 2011]. • bookmarking/discrimination of points and fragments in temporal scope (e.g. table of contents, annotated timelines) [Vliegendhart et al., 2012]. • linking (to internal or external resources) [Hildebrand and van Ossenbruggen, 2012]. • additional (toggleable) tracks (e.g. subtitles and text comments) [Laiola Guimarães et al., 2012]. • interactivity enhancements (e.g. pan, zoom, etc.) [Carlier et al., 2011].
Editing	Modification of spatial and temporal scopes of source content.	• manipulations such as compositing, cropping and fragmentation [Berthouzoz et al., 2012]. • reordering, removing, adding and joining fragments and tracks [Shrestha et al., 2010].

(b) recognition of end-users as active elements in all stages of production workflows that are totally oriented towards user-generated content. Here users are able to collectively and iteratively evolve user-generated content in non-destructive manner, increasingly aggregating value to media assets.

2. development of easy mechanisms for capture and transfer of metadata along user-oriented production processes. This requirement has the following implications:

   (a) generation of metadata as easily as possible, preferably as a natural consequence of user interaction with the production environment. This implies that metadata should be obtained with low user effort, preferably in automatic ways, through smart environments and devices that sense and record relevant information about the event reported by the media;

   (b) application of easily generated metadata to assist users on re-purposing user-generated content. Especially that metadata obtained by several iterations of enrichment be accumulated and be made available to assist users to consume contents and to perform their editing decisions.

In order to attain these requirements a lot of issues must be sorted out, from smarter capture infrastructures to better media interaction alternatives, just to name a few. Toward such challenges, ubiquitous computing [Schmidt, 2010; Weiser, 1999], which has developed into an umbrella of applied computing themes [Abowd, 2012], has a prominent role in tackling these issues. A lot of research in ubiquitous computing is centered around three themes [Abowd et al., 2002]: capture and access, natural interfaces and context-aware computing, all of them with great potential to contribute to more transparent multimedia production. As evidences to this potential, portable capture and access technologies, such as cameras and smartphones with always-available network connectivity, associated with popular web-based multimedia content repositories, have contributed to a proliferation of media assets; more natural interfaces (e.g. touch, gestures, and voice interfaces) have promoted easier means for recording and sharing multimedia contents; and a sensor-rich environment where mobile devices are equipped with several sensors (e.g. GPS, accelerometer, etc.) have eased the generation of valuable contextual metadata for describing media and assisting authoring tools. The next section presents more details on the theme of capture and access, which is adopted as multimedia production perspective in this dissertation.

2.2 Capture and access

Capture and Access (C&A) has been defined as "the task of preserving a record from some live experience that is then reviewed at some point in the future" [Abowd et al., 2002]. The preservation of experiences in multimedia form is a highly relevant problem due to the potential human difficulty on recording and retrieving, with sufficient details and precision, information of interest in everyday activities. C&A research has a focus towards automated applications whose kernel revolves around the two important phases that comprise its name. In the *capture phase*, occurs the automatic recording of heterogeneous streams of information that document a live experience. Later, in the *access phase*, the application provides interfaces that aggregate the various streams of information in a coherent manner so that the experience can be efficiently recalled.

The preservation and review of multimedia information obtained from work-related and educational human activities are recurrent domains for capture and access applications. From the standpoint of capture, the experiences are recorded in collocated settings (e.g. in instrumented rooms[1] dedicated to lectures [Dickson et al., 2012] or meetings [Ehlen et al., 2008]), in remote settings (e.g. via a videoconferencing infrastructure), or in mixed settings (e.g. via the combination of instrumented rooms and videoconferencing software, for instance). Another recurrent domain is the capture of personal experiences (e.g. personal lifelogging) [Kalnikaite et al., 2010], in which a C&A system seamlessly record relevant events from a person's life during a variable amount of time and provides proper interfaces for the experiences to be revisited in an integrated manner.

[1] An instrumented room is a physical environment with facilities designed to provide synchronized capture

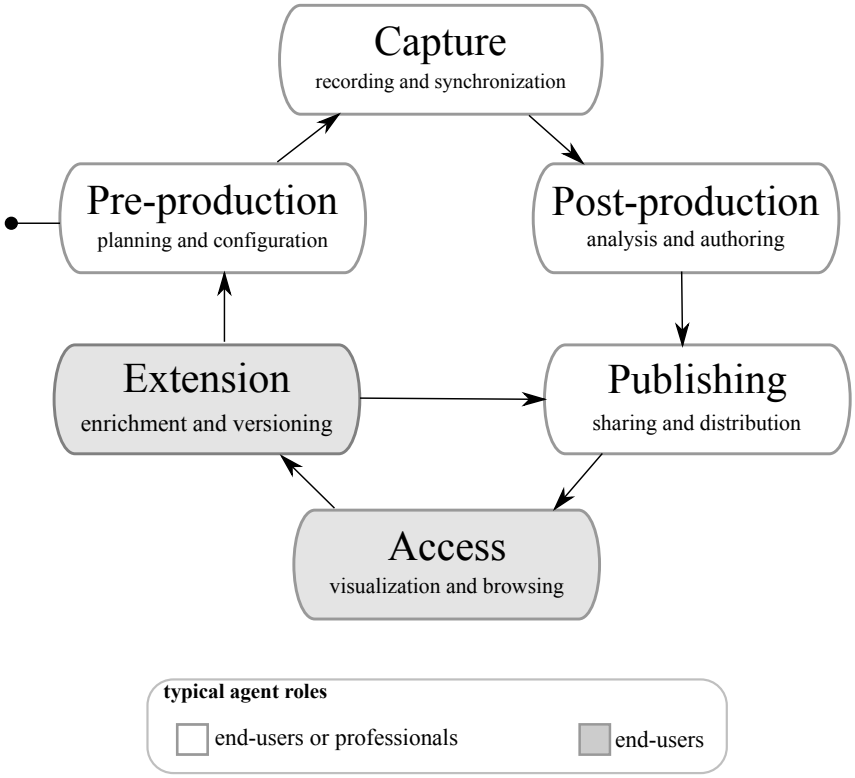


Figure 2.2: *Capture and access* multimedia production process

Capture and access applications are essentially multimedia production applications, and thus can be analyzed under models for multimedia production processes. Pimentel et al. [2001], based on Abowd et al. [1998], proposes a 5-stage process (pre-production, recording, post-production, access and extension), in which each stage provided opportunities for capture and transfer of user-led enrichment with low effort [Pimentel et al., 2000]. Subsequent research on enrichment, lightweight authoring and access/review of captured media created insights that led to an enhanced process [Martins and Pimentel, 2011], which encompasses six stages (Figure 2.2).

- **Pre-production**: typical activities on this stage are decision of what to capture (in reference to the event that the media will document), selection and preparation of the capture equipment and recording environment and, in the case of collaborative activities, negotiation of attendance of the participants;

- **Capture**: this stage involves recording and temporal synchronization of all media streams involved in the captured event, using devices and environments set up in the pre-production stage. Multimedia information is abstracted as sessions which synchronize the several

of heterogeneous media streams. Such environments are commonly employed in the recording of collaborative activities, for instance meetings and lectures. Technologies available in such environments include audiovisual recording equipment (e.g. video cameras e microphone arrays) and interactive surfaces (e.g. interactive whiteboards and tablets, instrumented to record synchronization events) and, in many cases, sensors (e.g. basic biometrics-based sensors and RFID as well as more advanced sensors — especially in the research contexts — such as eye tracking devices and galvanic skin response).

streams of timed information captured during the experience. In the case of groupware activities that employ synchronous collaborative tools, for instance, a session will aggregate streams of continuous media (e.g. audio, video, etc.) and discrete media (e.g. slides, instant messaging, shared documents, etc.), all timed to the temporal scope of the session. Concerning metadata, the capture stage poses several opportunities for gathering useful information via sensors and complementary capture devices: user-media interactions (e.g. slide changes and online multimodal annotations), user-user interactions (e.g. speaker turns, gestures and social behaviors) and user-equipment interactions (e.g. activation of microphones and identification via RFID) can be exploited as valuable metadata to inform activities in latter stages;

- **Post-production**: in the post-production stage captured media is prepared to be consumed by users. In order to achieve this objective, general activities that occur are derivation of metadata from the media assets (for instance via content-based methods), manual or automatic authoring of multimedia presentations and packaging of the presentations in formats that are suitable for easy consumption;

- **Publishing**: after the media assets have been prepared in the post-production stage or enriched by users during the extension stage, the resulting media assets need to be distributed to end-users through the various channels (e.g. web repositories or broadcast medium). This process entails the preparation of the media to be presented in several alternative access devices and aggregation of descriptive metadata or end-user enrichment;

- **Access**: from the user standpoint, access is historically the most familiar stage in the process. It is in the access stage that most users have traditionally experienced passive consumption of broadcast content. However, this phase also opens up several possibilities for users to actively engage on interaction with the media. For instance, on watching or reviewing some content, user navigational actions, curation commands and attention metadata can be selectively captured respecting privacy-preserving enforcements, so as they can be used for audience analysis and derivation of implicit or explicit ratings. Additionally, lightweight editing mechanisms can empower users to actively enrich the content while consuming them;

- **Extension**: it is in the extension stage that all metadata captured along the process can be used to assist the user in authoring decisions to customize professionally-generated or user-generated content. Important requirements in this stage are the presence of intuitive authoring tools that take advantage of contextualized metadata and simple means to generate new versions of the original content with aggregated enrichments.

Whereas this process shares numerous characteristics with previous ones, it presents advantages that make it more suitable for attaining the requirements listed in Section 2, especially

because of its iterative nature. Requirements (1.a) and (1.b) are anticipated by the "loop" comprised between the access, extension and publishing/distribution stages, which enable iterative versioning and transformation of content, regardless if it originates from professional or amateur settings. This same characteristic also promotes requirements (2.a) and (2.b): as long as metadata is properly aggregated in media elements, it can accumulated in several iterations of amateur enrichment and transformation or be reused to subsidize more elaborate productions in the future by feeding the pre-production stage.

In summary, capture and access, in particular, and multimedia production, in general, are complex activities composed of multiple information management and transformation tasks that support an underlying creative goal. Examples of these tasks are: browsing and visualization, review, annotation, cataloging, editing, and so on. From this extensive group of issues, three of them are regarded as of particular interest to this dissertation: annotation, browsing and authoring. These three tasks are closely related to each other, provided that annotations, whether manual or automatic, have potential to aid either browsing tasks (e.g. via visualization and navigation of points of interest) or authoring tasks (e.g. via combination of browsing and underlying content of annotations as an aid for editing decisions). These issues are discussed in the next sections.

2.3 Multimedia annotation

Research on content annotation, either digital or physical, is a far reaching theme with numerous theories and developments in several disciplines, such as library and information science, databases and information retrieval, just to a name few areas. Constraining the problem of annotation to the scope of multimedia production, annotations can be defined as informational units that document media assets generated along a production process. The ways in which such a documentation relationship occurs can be characterized via several dimensions, as illustrated in Figure 2.3. The following sections discuss each of these dimensions.

Function

According to Agosti and Ferro [2007], annotations can function either as metadata or as content. When an annotation functions as *metadata*, it has a descriptive role towards the associated content, and it is created for purposes of information organization and retrieval. Such documentation task encompasses many concerns of media assets and their creation, for instance issues of identification, production tracking, rights management, publishing, process documentation, content description and enrichment relations [Bailer and Schallauer, 2008]. As a consequence, it is recurrent that these types of annotations follow structured schemas and controlled vocabularies, such as MPEG-7 [ISO, 2002a], TV-Anytime [ETSI, 2007], Dublin Core [ISO, 2009b] or W3C Ontology for Media Resources [W3C, 2012c]. When an annotation functions as *content*, it has an augmentation role towards the information it is assigned to, in

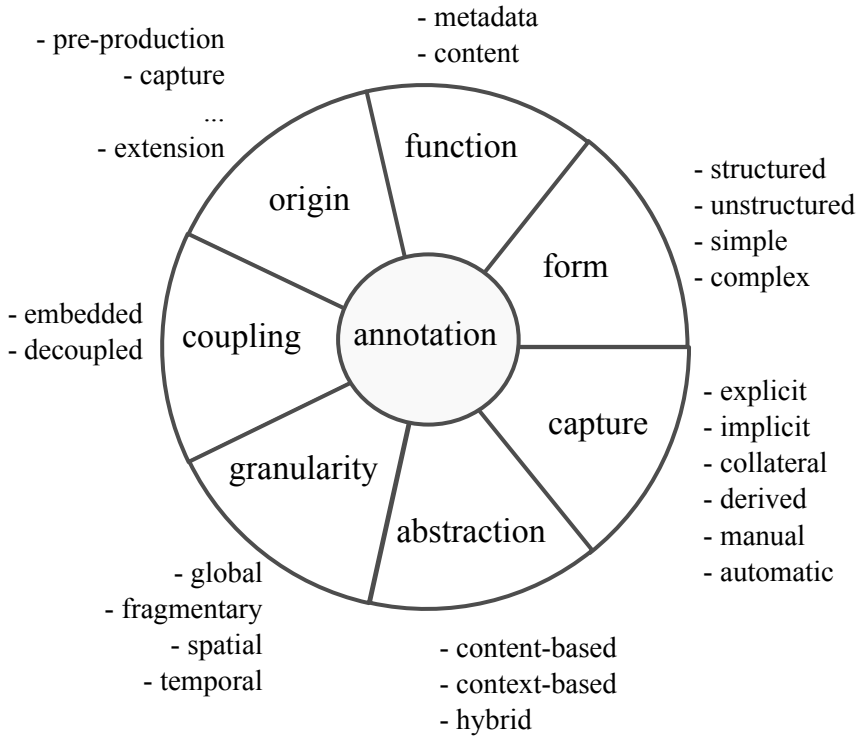


Figure 2.3: Framework for analyzing multimedia annotations

which case the purposes are varied, such as: sensemaking [Kalnikaitė et al., 2012] (e.g. stress or comment fragments as an attention, comprehension or memory cue), extending the conveyed message [Winget, 2008] (e.g. assigning notes with reviews, interpretations, clarifications), associating resources (e.g. by assigning related content via links or other methods) [Friedland et al., 2013], and so on. Naturally, depending on the purpose of a content-based annotation, privacy issues might arise: for instance, users might use annotations to attach personal views as a way to curate content [Whittaker, 2011]. In these cases, it is possible that users may not be willing to share some of these annotations.

Form

Depending on the adherence to predefined schemas, annotations can be *structured* (as commonly observed in metadata models), when a predefined schema or vocabulary is taken as reference, or *unstructured*, when no predefined structure is taken as reference. From the content standpoint, annotations can be *simple objects*, when they are composed of a single element, or *complex objects*, when they are composed of several (potentially multimodal) elements combined. Structured annotations are commonly represented as simple objects composed of a text element (e.g. an element from metadata vocabulary) or as a simple temporal discrimination/bookmarking of actions (e.g. logging playback interactions, implicit feedback, etc.) [Teevan et al., 2012; Vliegendhart et al., 2012]. Unstructured annotations, when represented as single media objects, have a more varied spectrum, encompassing not only text (e.g. unstructured text, for instance

unconstrained tags, text comments, etc.) [Ames and Naaman, 2007], but also other media modalities, such as audio (e.g. audio narrations and commentaries) [Encelle et al., 2011], digital ink (e.g. overlaid markings and highlights) [Cabral and Correia, 2012] and video (e.g. video narration and commentaries) [Kopf et al., 2012] . A single annotation can also be represented as a complex multimodal object, for instance via the association of a whole multimedia presentation as annotation to a media element, commonly practiced in the production of hypervideos [Tiellet et al., 2010].

Capture

Annotations can be captured in many ways by several actors involved in the multimedia production process. With a focus on capture and access applications, Minneman et al. [1995] and Geyer et al. [2005] categorize annotations according to ways they are obtained: *intentional (explicit) annotations* are created explicitly by the content producer or the content consumer; *collateral (implicit) annotations* are produced automatically by the capture environment (e.g. time annotations, geolocation, usage logs, linking associated resources, context information in general, etc.); *derived annotations* are automatically obtained based on content analysis, for instance for the identification of individual discourses or moments of user-user interaction; and *post-hoc annotations* consist on annotations generated during the consumption of the content (e.g. either as explicit user commentaries or as implicit usage logs).

A recurrent facet on this classification is that annotations can be created either *manually* (in the case of intentional and *post-hoc* annotations) or *automatically* (which include derived and collateral annotations). Manual annotations have been extensively studied, mostly regarding better interaction techniques, more efficient annotation tools or user studies to elucidate requirements for such tools [Cabral et al., 2011; Weibel et al., 2011]. There is also considerable volume of research on semi-automatic annotation, which frequently concerns optimization of the user experience in specific situations, such as suggestion of relevant annotations given a target media element [Wang et al., 2012]. Automatic annotations involves the extraction of semantics either from the media elements themselves, from additional information external to the media elements, or from a combination of both sources. Such semantics extraction process, or abstraction process, is discussed in the next section.

Abstraction

In the case of automatic annotations, abstraction encompasses recognition of meaningful information from a set of features internal or external to the annotated media asset. Abstraction of annotations can be achieved via *content-based* analysis of the various signals aggregated in the session, or via *context-based* sources by enriching the session with additional metadata collected with low-effort via instrumented software (virtual sensors which log events) or instrumented environments (physical sensors) [Yu and Nakamura, 2010]. As examples of content-based anno-

tations, the processing of audio utterances have been used to perform speech segmentation, word spotting, skimming and topic segmentation [Bouamrane and Luz, 2006] whereas speech recognition has been applied to index discourse using text-based techniques [Kaptein and Marx, 2010]. Video-based sources have been used to index non-verbal interaction, for instance via computer vision to detect behaviors from patterns of eye gaze, gestures and focus of attention [Korchagin et al., 2012]. The use of context-based sources has the purpose of associating external attributes in order to enrich the description or the content of the media elements. In a lower abstraction level, the usual method for obtaining these attributes is via sensors (e.g. collecting signals related to identification, localization, pressure, skin conductivity, accelerometers, etc.) [Liu et al., 2009]. In a higher abstraction level, the spectrum of modeled attributes is broad, encompassing: environmental attributes (e.g. spatio-temporal contextualization) [Wu et al., 2007], social attributes (e.g. identification of postures, facial expressions and gestures in social interactions) [Jayagopi et al., 2012], among others.

Provided the inherent cost of generating annotations manually, the prospect of automatically generating them is tempting. However, automatic annotation methods pose several limitations as far as high level semantics is desired. Such limitations, to a large extent derived from the semantic gap problem, consist on the difficulty of mapping low level media features to high level meaningful concepts. As a consequence of this problem, efficient methods for automatic annotation are generally directed to specific datasets, thus suffering from low generalization. Additionally, even efficient methods may not be able to solve all inherent ambiguities present, for instance, in visual media, such as the geographical localization of a scene or identification of an entity, without employing evidences external to media element features.

The combined use of context information has been advocated as a promising solution to part of these problems, but a limitation on context-based methods is the need to instrument capture and access environments in order to obtain these information bits. Even though the broad use of mobile devices packed with different types of sensors has been creating opportunities for advancing research with certain types of attributes (e.g. localization, accelerometer, etc.), more far-reaching context information still requires dedicated infrastructure, so that significant amounts of data can be collected. However, such instrumented infrastructure will hardly be available in all unconstrained environments where amateur multimedia production takes place, at least not in all production stages. A recurrent alternative to overcome this limitation has been to abstract data from online social networks as "virtual" sensors, which generate lots of data streams, in order to derive a restricted set of context attributes, especially those derived from use and sharing of multimedia information. Aside from the restricted set of context information that can be obtained from these sources, proper use of this information is also challenging in some respects, provided the lock-in of information on hosting companies, partly due to privacy and revenue models involved.

Granularity

Annotations can be anchored or contextualized to reference content in varied granularity. In the case of simple media objects (e.g. those composed of a single media element), the annotations can be anchored to the media element as a whole, achieving coarse granularity (e.g. global descriptive metadata) or they can be anchored to individual fragments of media elements, achieving finer granularity, specifically to spatial fragments (e.g. a visual region from an image or video), to temporal fragments (e.g. a time interval from a continuous media element, such as audio or video) or to a combination of both (e.g. a hotspot over a visual object during a time interval). In case of complex media objects (e.g. those represented as containers which aggregate multiple media elements), annotations can also be applied to specific elements that compose the object (e.g. the annotation of audio track of a video, or the annotation of a media element inside a multimedia presentation). Several multimedia technologies offer the possibility of addressing media elements in fine granularity. This is true for annotation schemes (e.g. video segments and spatial decomposition in MPEG-7, segments and segment groups in TV-Anytime, as well as fine-grained abstract annotation models [Haslhofer et al., 2012]) as well as for multimedia presentation languages (e.g. anchors, tracks and named fragments in languages such as NCL [Soares-Neto et al., 2010], SMIL [W3C, 2008a] and the HTTP-HTML stack [Mannens et al., 2012]).

Coupling

Annotations can be tightly coupled, when they are aggregated in the same container as the media elements, or they can be loosely coupled, when they are aggregated in a document or repository that is external to the media container. Tight coupling is common in certain metadata formats, such as MPEG-7 [ISO, 2002b] and EXIF [JEITA, 2002], which directly encode metadata in media streams, and also in some unstructured annotation formats that yield annotated packages [Sadallah et al., 2012]. A direct advantage of tightly coupled annotations is that the annotations are kept alongside the media objects as it is transferred trough the stages of the multimedia production process. As a consequence of this advantage, annotations are always available without the need for some external resolution mechanism. On the other hand, decoupled annotations are more friendly to organization and retrieval in large collections, provided that they can be managed via efficient access mechanisms in external repositories without the need for data duplication and consequent consistency mechanisms.

Origin

Regarding when and where annotations are generated, the problem can be framed to the moments at which each annotation is created in the production process. Recalling the capture and access process previously defined, Figure 2.4 illustrates the dynamics of annotations generation,

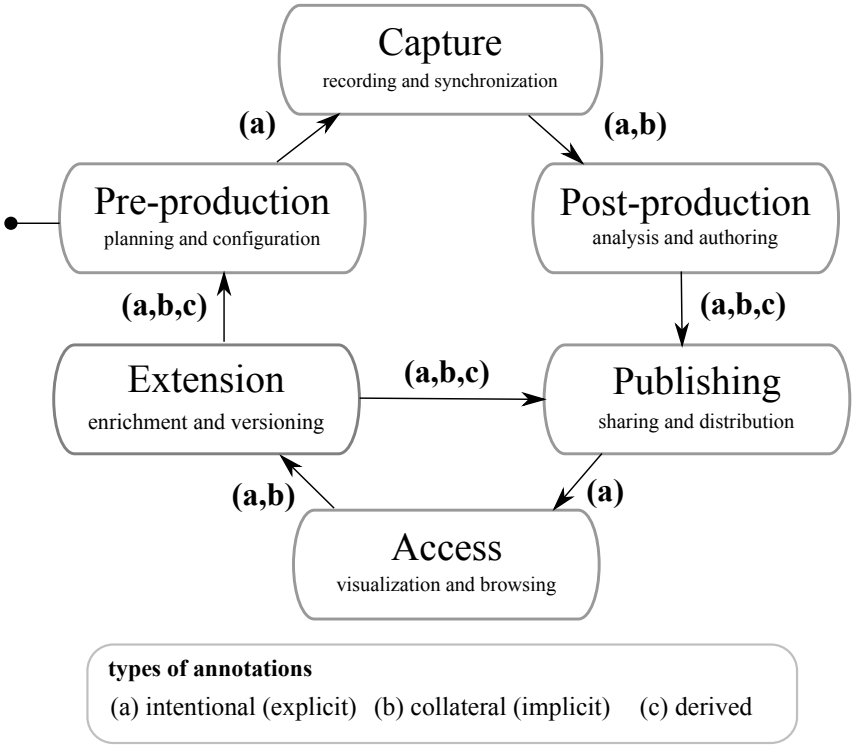


Figure 2.4: Annotations generated as output at each stage of a multimedia production process.

in which each arrow represents an output of annotations, possibly of various types, originated in a stage. In the pre-production stage, offline intentional annotations and documents can be applied to document who are the expected participants in the captured event, when the event occurred and what is its purpose, as well as general planning of the event as a whole [Linhalis et al., 2010a]. In the capture phase, online intentional annotations and collateral annotations can be constructed, for instance. Online annotations, in this case, may represent impressions and interpretations about the event (e.g. created by the participants of a lecture or talk, for instance) whereas collateral annotations may represent low-level metadata from the capture environment (e.g. from the capture devices or auxiliary sensors) or also synchronization events which will enable the reconstruction of the experience from multiple streams [Linhalis et al., 2010b]. In the post-production stage, several authoring-related annotations can be applied over the content (e.g. segment annotations, edit decision lists and so on) as well as the media assets can be processed in order to derive content-based annotations, either for editing or publication purposes. In the publishing stage, descriptive metadata is applied to the content for organization and retrieval purposes. In the access phase, besides intentional user annotations applied over the content, also collateral annotations generated as a consequence of the user interaction (e.g. playback commands, activation of fragments and links) can also be collected from interest measurement purposes [Gkonela and Chorianopoulos, 2012]. In the extension phase, the content receives additional layers of editing- and enrichment-related annotations. All these annotations, accumulated along the several phases up to this point can be fed to pre-production or publishing

stages in order to iterate the life cycle of the reference multimedia content.

2.4 Multimedia browsing

The issue of browsing and navigation of information has been extensively studied by information science and information retrieval researchers, provided the importance of studying human-information behavior as a means to bridge the gap between user's information needs and the meaning abstracted in the content [Järvelin and Ingwersen, 2010]. In the specific case of multimedia production, browsing is a key task to the management of media assets along the creation process. This is clear as far as a single (continuous) media asset gets too long: in this case, it becomes increasingly difficult to find points of interest in the contained information streams without additional mechanisms other than linear, VCR[2]-based, playback control. Analogously, as far as a multimedia repository contains too many media assets, it becomes increasingly difficult to find assets using ad-hoc organization mechanisms. Therefore the theme of browsing is closely related to — and, to a greater extent, dependent upon — the theme of multimedia annotations: most browsing strategies build upon the idea of exploring a collection of media assets by means of their annotations. In fact, in many circumstances, browsing tools act as direct "front-end" approach which enable users to take advantage of annotations.

Browsing multimedia information is a task that is recurrent along the whole multimedia production process, but it is of uttermost importance in the stages in which the involved actors engage in lean-back interaction with the content. This is particularly clear is the access phase, in which users may need to engage in browsing behaviors to explore collections of media elements (e.g. search and navigation inside a digital library or an online video repository) or even navigate to points of interest inside a single media element (e.g. navigating to a specific slide or chapter inside a lecture recording). This behavior has been observed, for instance, on studies about reuse of webcasts and video lectures [Dufour et al., 2011]. Browsing is also a key information seeking behavior in the post-production stage: producers may need to browse through all captured content, in order to review, document and structure the most important fragments that will compose the final product. For instance, Markkula and Sormunen [2006] has analyzed video seeking behaviors of journalists in TV production, having observed that browsing for videos occurs through the whole process, either as a technique for idea generation or as a means to gather material for authoring purposes. Notwithstanding the prominence of the access and post-production stages, browsing also occurs in other stages: for instance, Junuzovic et al. [2011] has demonstrated the usefulness of reviewing ("replaying") previous moments in a live video conference in order to recall missed moments, thus demonstrating the usefulness of

[2]VCR (Video Cassete Recorder) controls are classic operations for continuous media element control: it encompasses operations such as play, pause, stop, fast forward, rewind and slow motion, which are inspired by the tape-oriented nature of interactions with video or audio recordings using these devices. Most digital video players still adopt this metaphor as a primary playback functionality, in many cases only augmented by a timeline slider for random access and a duration tracker.

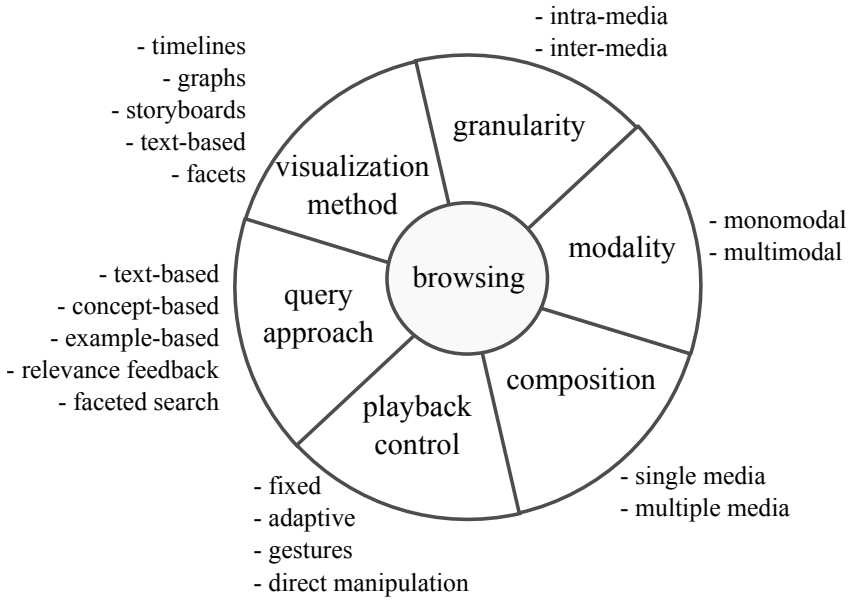


Figure 2.5: Framework for analyzing browsing approaches

browsing functionality in the capture stage.

The most important issues that characterize browsing models and tools are discussed in the following topics and illustrated in Figure 2.5.

Granularity

Inter-media browsing refers to the interactive exploration of multiple media elements organized in collections whereas *intra-media* browsing refers to the internal, non-linear, exploration of a single media element. Inter-media browsing is commonly achieved in multimedia digital libraries, and online repositories in general, where users can engage on finding specific media elements among a potentially large collection. Intra-media browsing is common in time-continuous media elements, where users can navigate to specific points of interest; browsing is also attainable with discrete visual media elements where the visual content does not fit a viewport (e.g. an image representing a map): in this case, browsing may occur via pan, drag and zoom controls [Kuijk et al., 2009].

Modality

Multimedia browsing tools can be either *monomodal* or *multimodal*, depending on the media types which the tool emphasizes access to. For instance, in the context of meeting browsers (e.g. multimedia browsing tools for meeting recordings), Whittaker et al. [2007] reports that many monomodal browsers are restricted to a single type of medium, for instance: audio (utterances) [Tur et al., 2010], video (keyframes and visual events) [Adcock et al., 2010], documents (e.g. ink-based notes, whiteboard content, shared documents, interaction logs,

etc.) [Monserrat et al., 2013] and text (e.g. speech transcription). On the other hand, multimodal browsers provide simultaneous access to multiple media streams in a synchronized way, for instance the synchronization of video and whiteboard content [Behera et al., 2007], or even additional information that complement audiovisual recordings such as, in the case of meetings, speaker turns, action items, agenda items and shared documents [Geyer et al., 2005].

Composition

In the context of intra-media browsing, a browsing tool can be focused on exploration of a *single media* element or of *multiple media* elements. Browsing a single media element encompasses for instance exploring a single video [Schoeffmann and Boeszoermenyi, 2009] or the timeline of an audio recording [Luz and Kane, 2009]. Multiple media elements are more relevant when browsing synchronized multimedia presentations: in this case, multiple media elements are orchestrated in a way that reconstruct their captured time relationships. Examples of such compositions are video presentations with synchronized slides [Chandra, 2011], multi-participant videoconference recordings [Junuzovic et al., 2011], multimodal meeting browsers [Behera et al., 2007], and multi-camera events, such as surveillance video browsers [Snidaro et al., 2012] and recordings from music concerts [Mase et al., 2011] and social events [Yu et al., 2010].

Playback control

In order to review a recording, a user may need to control the playback rate in fine-grained time resolution. Usual playback control methods are the classical VCR operations (play, pause, fast forward, rewind, etc.) augmented with a scrubbing method (i.e. a time slider for random access) — although familiar, there are situations in which more control is desired over the pace of presentation than what this set of controls can offer. Some research has been concerned with this problem, tackling it either by improving classical controls or proposing novel operations. On the side of improvements, augmenting a time slider with keyframes, selected in a fixed frame rate or with adaptive algorithms, has been an alternative [Matejka et al., 2013]. There are also improvements on fast forward and rewind operations, making playback speed adaptive to specific events of interest [Höferlin et al., 2010] (e.g. slow down when an attraction is reached in videos of scenic car driving [Cheng et al., 2009]), or making the presentation return to normal speed when associated media (e.g. subtitles) is traversed [Kurihara, 2012]. On the side of novel operations, there are interaction techniques for controlling media via gestures, which prove useful to support the prominent use of touch-based devices such as tabletops [Bertini et al., 2011] and smart phones [Huber et al., 2010]. Another line of novel operations consists on direct manipulation of objects in the scene, for instance for replaying a specific scene by dragging an object along its original trajectory [Santosa et al., 2013].

Query

In many situations, browsing multimedia information may encompass the formulation of queries, either for restricting the set of media elements that is presented (in the case of inter-media browsing) or for filtering the fragments of interest from a single media element (in the case of intra-media browsing). Blanken et al. [2007] classifies queries in multimedia systems in three classes: *text-based queries* are the usual approach, in which the user must formulate a free form text request that potentially match the annotations indexed in a media collection; *concept-based queries*, on the other hand, are composed of terms obtained from a controlled vocabulary, such as taxonomy or an ontology; and *example-based queries*, which are common for non-textual queries, i.e. where a sample media element is issued as request so that in response it is returned other media elements according to their similarity to the query. Alternatively, queries can be classified in metadata-based (which involves both text-based and concept-based) and content-based (which involves example-based and feature-based queries).

Due to inherent problems on abstracting meaning from multimedia information, it is not uncommon for a multimedia system to mix query modes. For instance, many systems combine text-based queries with example-based queries via the *relevance feedback* technique, e.g. the browsing interface proposed by Hopfgartner et al. [2009], which allows users to inform which items in a list of results is more relevant to a initial (text-based) query, thus filtering the result space to items similar to the informed one. On the combination of text-based and concept-based queries a recurrent technique is *faceted search* [Perugini, 2010], which allows the user to filter a search result space via multiple hierarchies of concepts (so called "facets"), each hierarchy representing a dimension. Using this technique, a user can, optionally, start a search with a textual query and, later, filter the results with several criteria that may be present in the retrieved media elements. Examples are filtering image collections by object, space, time and activity [van Zwol et al., 2010], or video interview collections by the occurrence of multiple discourse acts [Christel et al., 2010].

Visualization

Visualization is an important issue for browsing information in any granularity: for intra-media browsing it is critical to assist users in visually identifying fragments of interest; for inter-media browsing it is in most situations indispensable provided the need to visually present a collection of elements. Provided that a multitude of multimedia visualization approaches have been proposed in the literature, the following list of categories is not exhaustive but represent some of most frequent approaches:

- *Timelines*: this visualization takes advantage of time continuous nature of certain multimedia presentations. This is achieved by plotting time-centered information in a 1D or 2D visualization, for instance: timelines decorated with annotations of either low level or

high level media element features [Hauglid and Heggland, 2008], scrubbing time sliders augmented with keyframes or fisheye visualizations [Mertens et al., 2011], or plots of low-level signals along the video timeline [Hagedorn et al., 2008];

- *Graphs*: certain annotated multimedia datasets present association relationships between its underlying annotations, which allows visualizing the information as trees and more general graphs. Graph-based visualizations of low-level features, whose links are computed via a distance metric, has been demonstrated by Delest et al. [2006]. Systems that count on annotations with semantic relationships also can take advantage of graph-visualizations to allow navigation of related media elements, as demonstrated by Bertini et al. [2011]. On the issue of intra-media visualization, hierarchical video browsing has been demonstrated [Del Fabro et al., 2010] as a way to incrementally explore segments of interest in a video by traversing a tree visualization;

- *Storyboards*: for purposes of intra-browsing over visual media, storyboards are frequently adopted as a visual summarization method: the usual approach consists on generating a grid of keyframes that the user can quickly inspect and jump to the most relevant fragment. The core problem in generating such visualizations is adopting a proper criteria to generate a list of keyframes that effectively summarizes the media elements. This problem has been tackled for instance by segmenting the underlying visual structure of edited videos (e.g. scenes and shots) [Cooray et al., 2009], in adaptive ways by tackling semantic compression via action detection [Adams et al., 2008], and by detecting visual events [Hong et al., 2011], just to mention a few approaches;

- *Text-based*: many tools rely upon text-based annotations to provide visualizations. This can be achieved for instance by linking fragments of text to segments of a video (for instance, in the case of linked discourse transcriptions [Li et al., 2011]); or visualizations of text, for instance tag clouds to visualize the frequency of emotions in movies [Martins et al., 2011b] or user-assigned tags in photos [Girgensohn et al., 2010];

- *Facets*: besides their direct applications for query purposes, facets are also a way to orchestrate various visualization methods into a single browsing tool: assuming that each visualization represents a facet (or dimension), users can filter an information space via multiple criteria. This alternative has been explored in many multimodal browsing methods. Zhao et al. [2012] has demonstrated the combination of an annotated timeline with text-based annotations [Diakopoulos et al., 2009], whereas Haesen et al. [2013] has combined annotated timelines with graphs of related assets.

2.5 Multimedia authoring

Multimedia authoring is the activity of specifying the orchestration of multiple media elements with purposes of conveying a message [Bulterman and Hardman, 2005]. Thereby, concerning the final product of a multimedia production process, authoring is a core activity: even though it is more strongly related to the post-production stage, authoring encompasses also the integration of all information that is captured, created and annotated, with the purpose of creating new content. Authoring methods can be *manual*, *semi-automatic* or *automatic*.

During authoring, a user has the role of *manually* executing a set of canonical tasks [Cesar et al., 2009; Hardman et al., 2008] before obtaining the final product, of which the following are more related to media transformation:

- *select* the media elements which will compose the presentation, which can include search and inter-media browsing in local or remote media repositories;
- *fragment* selected media elements in order to specify only the relevant parts, which include intra-media browsing, annotations for organization and documentation purposes, and application of editing decisions as cuts over the content;
- *compose* the media elements in a presentation, which involve the specification of temporal, spatial and associative relationships, of visual and aural transitions between the elements and interactivity resources in the presentation.

Much research has demonstrated the utility of supporting producers through the authoring process, mostly by minimizing the need for user intervention in part of these tasks, in the case of semi-automatic authoring, or in all of these tasks, in the case of automatic authoring. On the side of methods that seek complete automatization of the authoring task, relevant groups are:

- *automatic summarization*: consists on producing shorter versions, in the temporal domain, of complex media objects, in order to highlight the most important fragments [Truong and Venkatesh, 2007];
- *automatic generation of media shows*: consists on producing multimedia presentations (e.g. slideshows or mosaics with a background audio track) from thematic images (e.g. photo collections) [Singh et al., 2011];
- *automatic mashups*: consists on the generation of a complex multimedia object from a collection of (potentially overlapping) media elements captured by multiple participants of an event (e.g. in musical concerts [Zsombori et al., 2011]);
- *reviews of editing histories*: consists on the creation of multimedia presentations which reconstruct, with adjustable granularity, the editing process of a document (e.g. for generation of guided walkthroughs over a learning resource [Fernquist et al., 2011]);

- *automatic generation of tutorials*: consists on the instrumented capture of screen-based tutorials in order to automatically create interactivity points in the final presentation (e.g. "pause" points when an external interaction is needed) [Pongnumkul et al., 2011].

Most of these methods will depend to a large extent on the existence of annotations to guide their algorithms. Such dependence leads automatic authoring methods to inherit most of the limitations of automatic annotation methods, i.e. their performance will largely depend on the quality of available annotations. Complete automation of the selection and fragmentation tasks, for instance, will depend on proper annotations which allow to infer which objects to select and what fragments of these objects to include in the presentation. Automation of composition tasks are an even harder problem: even though low level composition (i.e. via different multimedia composition formalisms) are well known by the community, high level composition (e.g. regarding aesthetic requirements, story consistency, editing grammars, among others), reveal themselves as very challenging because entail highly subjective requirements which may demand, for every application, fine-tuned user-centered research in order to properly guide design decisions.

Facing these limitations, semi-automatic methods with a "user in the loop" approach are a recurrent alternative, namely methods that provide smart support for users in specific tasks that comprise authoring. For instance, users can be guided on planning a story and on organizing shoot material as demonstrated by Adams et al. [2005] via application of predefined narrative templates. Similar approaches have experimented with automatic alignment between user-authored natural language narratives and video footage in order to support semi-automatic movie composition [Diemert et al., 2013; Liang et al., 2013]. Additionally, smart capture equipment, for instance based on smartphones [Wu et al., 2007], can assign contextual sensor-based information to captured material which can help the editing process later. On the side of content-based automatic annotations, presenter postures have been correlated to editing decisions on lecture videos [Wang et al., 2005]. On the side of intentional annotations, Ma et al. [2012] have experimented with mobile applications to log fragments of interest at capture time and, later, annotated fragments are merged in an editing tool that assists users on applying editing decisions. The value of intentional annotations have also been demonstrated by Sadallah et al. [2012]: after a user annotate a video, a non-linear interactive video is generated based on the annotations. All these methods advocate the key role of annotations, obtained either automatically or manually, on supporting semi-automatic authoring methods.

Regardless of the method employed (manual, semi-automatic or automatic), the final product of authoring is a complex multimedia object which aggregates multiple discrete (e.g. images and text) or continuous (e.g. audio and video) media elements whose presentation layout is specified via temporal, spatial and associative relationships between these elements. Examples of complex multimedia objects are videos (most of which encapsulates at least one visual and one aural track), simple multimedia presentations (e.g. slideshows containing a combination of continuous and discrete media with animations [Edge et al., 2013]) and hypermedia documents

(e.g. interactive multimedia documents [Jansen et al., 2012], hypervideos [Tiellet et al., 2010], etc.). Respected the critical role of interactive multimedia documents in this dissertation, Section 2.6 discuss the most important issues of this general multimedia format.

2.6 Interactive multimedia documents: models and languages

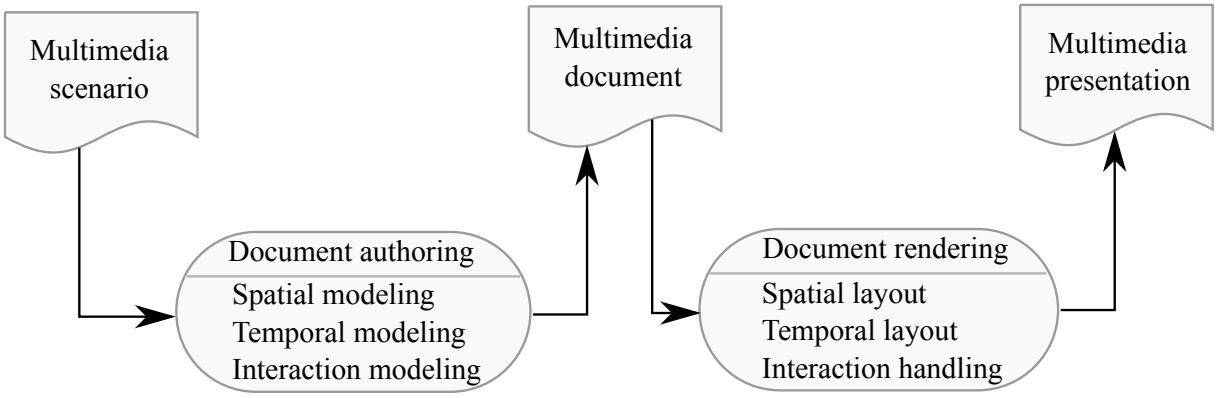


Figure 2.6: Multimedia documents: from scenarios to presentations. Adapted from Rogge et al. [2004] and Pellan and Concolato [2009].

An interactive multimedia document is the specification of a multimedia scenario in a formal authoring language which provides constructs to represent spatio-temporal constraints and interactivity behaviors. From a document engineering perspective, authoring a multimedia document involves a number of modeling and transformation tasks, as illustrated in Figure 2.6. The first model, a *multimedia scenario*, is an abstract representation (e.g. a user mental model or a set of computable rules) which represents a set of media elements and a set of abstract specifications involving temporal synchronization, spatial layout and interactivity behavior. A multimedia scenario has its abstract representation transformed into a concrete representation which is dependent upon a specific multimedia document model: in this case the abstract constraints are transformed into the equivalent temporal, spatial and interactivity constructs provided by the specific model, thus generating a multimedia document. The resulting *multimedia document* can then be played in a execution environment which will apply algorithms for spatial and temporal layout for rendering a *multimedia presentation*. When a multimedia document, optionally, includes a set of interactivity behaviors, it is called an interactive multimedia document and, in this case, the execution environment must be prepared to handle user interactions. When analyzing multimedia documents, it is important to consider the following issues, which are summarized in Figure 2.7: spatio-temporal layout and interaction models, representation, packaging and distribution issues.

From the standpoint of layout specification and rendering, *spatial layout* follows similar principles as those from a-temporal documents, for instance by positioning using either absolute,

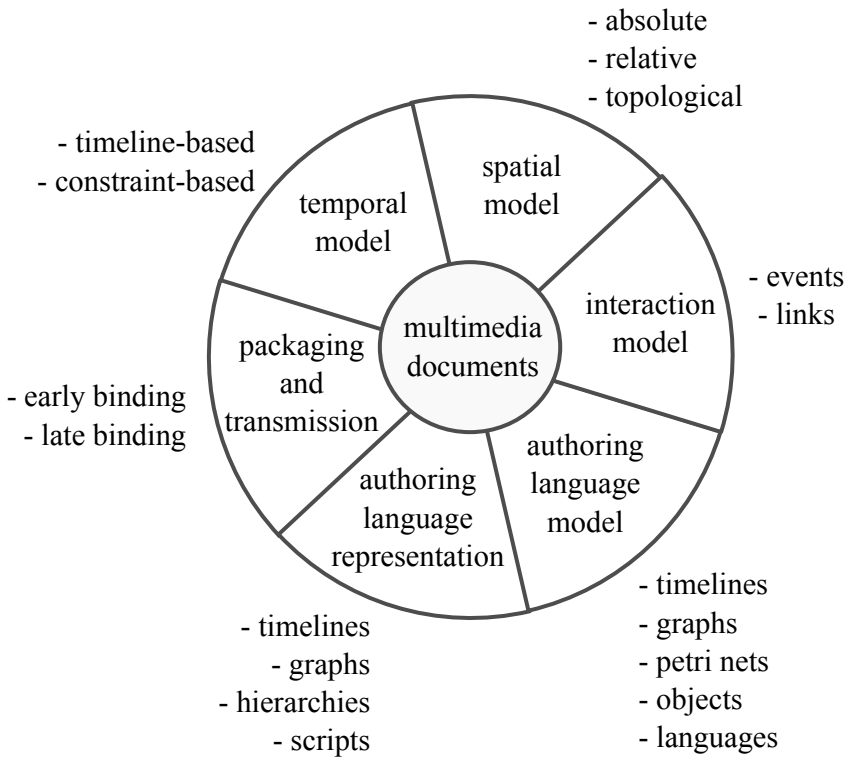


Figure 2.7: Framework for analyzing multimedia documents.

directional or topological relations between media elements and their compositions. Additionally, besides positioning, media elements and their spatial compositions can be styled via inline or external properties. *Temporal layout*, on the other hand, is a distinguishing characteristic of multimedia documents, and the choice of temporal model has profound implications on the enclosing document model. The literature reports several reference frameworks for multimedia synchronization that categorize models of temporal layout [Bertino and Ferrari, 1998; Blakowski and Steinmetz, 1996; Perez-Luque and Little, 1996]. In this discussion, we favor the framework proposed by Bertino and Ferrari [1998], given that it promotes a separation between abstract temporal models and their concrete specifications. In this framework, low-level formalisms for specifying temporal layout are classified in:

- *timeline-based model*: consists on specifying *quantitative* relationships between media elements over a single temporal axis, that is, a timeline. Such quantitative relationships can be relative (e.g. 6s after the beginning of the timeline) or absolute (e.g. at a wallclock timestamp);

- *constraint-based model*: consists on establishing primarily *qualitative* relationships between media elements. In this case, synchronization occurs via ordering relationships between presentation events represented as time points (e.g. start two videos at the same moment) or time intervals (e.g. start a video *after* the duration of a given image elapses), which makes this model be also known as *event-based*. Examples of qualitative relation-

ships are the 13 relationships defined by Allen [1983] (e.g. before, after, during, contains, overlaps, etc.). Additionally, in the case of constraint models based on intervals, qualitative relationships can be combined with quantitative offsets: for instance, to specify that a video should occur 10 seconds (quantitative) after another video ends (qualitative).

The simplicity of the timeline model, as an advantage, generally entails better execution performance, provided that it requires algorithms and data structures which aggregate potentially low computational cost. On the other hand, the use of a fixed time axis makes it unsuitable for presentations with unpredictable timing. Examples of such presentations are those whose flow depend on user interaction events (e.g. those provided by a "next" button in a video playlist or by a table of contents in an ordered collection of video clips): those events are external events whose occurrence is unpredictable and, consequently, it is not possible to anticipate, at authoring time, at which point of the timeline they will occur. Constraint-based models directly tackle this problem, provided that the use of qualitative relationships allows to plan the presentation of a media element relatively to the occurrence of an asynchronous (e.g. interaction-related) event. On the other hand, algorithms and data structures for realizing constraint-based timing tend to be more complex, potentially requiring additional heuristics in order to minimize runtime performance. Additionally, depending on the expressive power of an underlying constraints model, the document can present temporal inconsistencies which demand verification mechanisms [Gaggi and Bossi, 2011] to be applied before publishing — which, by its turn, can contribute with more overhead on the authoring process.

Representation of spatio-temporal relationships and interactivity occurs via an authoring language. From the standpoint of temporal specification, each authoring language customarily has an underlying *formal model*: e.g. timelines, graphs, objects, programming languages and petri nets [Bertino and Ferrari, 1998]. From a user perspective, authoring via a language occurs by means of one or more *representations*, either textual or graphical, of this language, possibly associated to a specific-purpose authoring tool. Bulterman and Hardman [2005] observed that authoring tools generally present a correlation between the choice of a formal model and the choice of a representation in the underlying authoring language, having identified four paradigms as frequent ones:

- *Timelines*: this paradigm represents a direct mapping of the timeline-based temporal model to an authoring interface. It represents points and intervals in a scheme of axis-based visualizations, thus providing an intuitive mechanism for users to visualize the (deterministic) temporal ordering of media elements. This approach is still widely used for authoring on multimedia objects without unpredictable events (e.g. in general video editors for professional or amateur use or even for authoring of interactive videos [Sadallah et al., 2012]). The simplicity and widespread adoption of the timeline paradigm has lead to attempts to integrate constraint-based time models in modified timeline visualizations [Cerqueira Neto et al., 2012];

- *Graphs*: graph-based visualizations for authoring are generally a direct mapping from graph-based formal models, such as in the cases of timed petri nets [Bouyakoub and Belkhir, 2011], timed automata [Jourdan, 2001], or general control flow [Meixner et al., 2012]. Graph-based formal models have been extensively studied in the multimedia synchronization literature because of their formal properties to specify and verify temporal behaviors of multimedia presentations. On the user side, authoring via graphs is generally seen as a low-level and very technical approach, not very intuitive for a wide spectrum of users: this is part of the reason that such systems have stayed on most part in academic tools;

- *Scripts*: consists on specifying synchronization in a procedural way, for instance via a programming language. This is an usual way of authoring complex multimedia presentations in many production systems, such as observed with ActionScript [Moock, 2007] in Flash[3] and with Javascript [ECMA-262, 2011] in Mozilla Popcorn[4]. As an advantage, counting on the expressive power of programming languages, these approaches are suitable to represent any temporal relationship required by a presentation. On the other hand, learnability of the method is a concern, provided that some classes of users, amateurs as an example, may be unwilling to learn a programming language in order to author multimedia presentations. This contributes to make direct authoring via scripts a niche for professional productions;

- *Structures* [Bouyakoub and Belkhir, 2011]: this paradigm consists on specifying timing relationships as composition and encapsulation/scoping abstractions over timed elements, encompassing visualizations such as hierarchies and general nesting of visual elements. This provides a more direct mapping to formal models that deal more closely with composition abstractions, such as object-oriented models. A classical example of this paradigm is the SMIL [W3C, 2008b] language, which allows the synchronization of media elements via "time containers", which are abstractions of composition of media elements with an associated scheduling semantics (e.g. parallel and sequential execution).

From the standpoint of *packaging and transmission*, a multimedia document can be self-contained or distributed. In the case of self-contained documents, both multimedia scenario (comprising spatial, temporal and interactivity relationships) and media elements are distributed as a single package. An example of self-contained document is a video container, which can include audio, video and text track streams multiplexed into a single stream. In the case of distributed documents, or distributed multimedia presentations, a multimedia document represents the relationships in the scenario whereas media elements are specified as references: consequently, there is a loose coupling between a multimedia scenario and underlying media elements. Furthermore, a distributed multimedia presentation is assembled only at presentation-time (late

[3] Adobe Flash technology. URL: `http://www.adobe.com/products/flash.html`. Accessed on July 2013.

[4] Mozilla Popcorn.js technology. URL: `http://popcornjs.org/`. Accessed on July 2013.

binding) whereas self-contained singly-streamed presentations are assembled at packaging-time (early binding). Regarding versioning and transformation of multimedia document, late binding presents more advantages, provided that enrichment and editing can occur on client-side without the need of passing through de-packaging/packaging stages, which can incur is significant overhead on the client.

Among the various multimedia document models proposed in the literature, in this dissertation there is a particular interest on multimedia document formats that are delivered via networked environments and, further, those that are concerned with distributed multimedia presentations:

- **Synchronized Multimedia Integration Language (SMIL)**: this is the language recommended [W3C, 2008b] by the W3C[5] for synchronized multimedia on the Web. Its spatial model allows specification of polygonal regions distributed via coordinates or topological relations. Its temporal model is hierarchic, that is, timing relationships are implied via hierarchies between document elements, which can be: media objects, which encompass continuous and discrete media elements, and time containers, which encompass abstractions of compositions (which can include either media objects or other time containers) with an associated scheduling semantics (e.g. sequential behavior, parallel behavior, etc.). The SMIL hierarchical time model is in fact a hybrid model, combining both timeline-based and constraint-based synchronization. Each node in the hierarchy, being either a media object or a time container, has its own (internal) timeline, thus timing specification inside a single composition can occur via a timeline-based synchronization. As a consequence, a SMIL document that takes advantage of timeline-based synchronization alone can be seen as an hierarchy of timelines, each timeline scoped to the enclosing composition. Furthermore, constraint-based modeling is also supported via event-based activation or deactivation of elements. Both timing models are combined via a set of rules that govern the update of the hierarchic timeline model upon the occurrence of events in the constraint model. Interactivity in SMIL is specified either via its constraint model, by activation/deactivation of elements via user-initiated events, or by linking functionality, which make SMIL a hypermedia language: links are spatio-temporally defined in a presentation and can be used to activate any referenceable element in the document. From the standpoint of transmission, as recommended in most W3C technologies, a SMIL document is transmitted in text-based format over the network and client platforms are responsible for rendering the multimedia presentation, an activity which includes fetching the referenced media elements;

- **MPEG-4 Binary Format for Scenes (BIFS)**: this is an ISO[6] standard [ISO, 2005] for distributed multimedia presentations. Spatial formatting is achieved via a hierarchical

[5]World Wide Web Consortium (W3C). URL: `http://www.w3.org/`. Accessed July 2013.
[6]International Standards Organization. URL: `http://www.iso.org/`. Accessed July 2013.

data structure, called scene graph, which is extended from the scene graph defined in the VRML (Virtual Reality Markup Language) [ISO, 1997] standard. Temporal formatting, for normative purposes, is timeline-based (via timestamps and reference clocks). Additionally, constraint-based synchronization can also be modeled via extensions, using a subset (FlexTime extensions) of Allen's temporal relationships, but in this case, before transmission, undeterministic events are enforced via network commands in a scheme that is very similar to a timeline model. Client-side interactivity in BIFS follows mostly the VRML events model. BIFS is a binary format which represents both a low-level authoring language and a transmission language (i.e. presentations are sent via a stream comprising scene and presentation commands). For declarative authoring, it defines two other languages: XMT-A, which is a direct mapping from the scene graph data model; and XMT-O, which provides higher level abstractions based on a subset of the SMIL language. For transmission purposes, both languages must be converted to binary format: in the case of XMT-A, this conversion is done in one step; in the case of XMT-O, a conversion to XMT-A precedes the conversion to binary format: such conversion steps can make very difficult to reverse the original declarative XMT-O document from a binary representation. Regarding transmission, a BIFS scene is transmitted in a scene stream, turning the client responsible for rendering the scene and fetching associated media streams. In fact, BIFS is both a binary representation format and transmission format, i.e. it is formed by commands that instruct either rendering and transmission of presentation units;

- **MPEG-4 Lightweight Application Scene Representation (LASeR)**: another ISO standard [ISO, 2009a] for distributed multimedia presentations. Spatial layout in LASeR is based on the W3C's SVG (Scalable Vector Graphics) recommendation [W3C, 2011d]: therefore, similarly to BIFS, scenes are described by a scene graph that compose document nodes. Differently from BIFS, in LASeR only 2D scenes are possible to model. A LASeR document extends SVG by including additional elements for continuous media, especially audio and video, which are imported from SMIL's Media Object Module. Temporal formatting occurs via a timeline model. Interactivity follows mostly the SVG events model. Regarding transmission, LASeR defines a binary encoding to which the SVG declarative document is converted. Additionally, the embedding of media elements directly in the SVG stream is supported;

- **Nested Context Language (NCL)**: this is a hypermedia language standardized [ITU-T, 2011] by ITU-T[7] with a focus on multimedia presentations over IPTV services. NCL is commonly defined as a "glue language" because it promotes the integration of arbitrary objects (e.g. continuous and discrete media elements, source code, other documents, etc.) in a single presentation. Spatial formatting occurs by the definition of polygonal regions which, similarly to SMIL, can be positioned via coordinates. For temporal synchronization,

[7] International Telecommunication Union. URL: `http://www.itu.int/`

the language counts on an elaborate hypermedia (graph-based) model which provides constraint-based synchronization via event-condition-action (ECA) rules: activation and deactivation of media elements occurs upon activation of links which, by its turn, can be activated via presentation, user interaction or contextual events. Besides arbitrary objects, the language also allows the definition of "contexts", an encapsulation of a group of media elements, which can be activated mostly as regular node in the presentation graph. Additionally, the language also supports scripting via the Lua[8] language, which potentially contributes with considerable expression power to its timing model. Even though it has been widely advocated as a language for TV environments, the expressive power of NCL has been also experimented in the Web environment [Melo et al., 2012].

Each of the aforementioned languages has been advocated for specific platforms: BIFS and NCL for TV-oriented platforms; SMIL for the Web platform, as a standard, and for the desktop, as a reference implementation (i.e. Ambulant implementation [Bulterman et al., 2004]); and LASeR for mobile-oriented platforms. LASeR and BIFS present simplified temporal models (purely timeline-based, in the case of LASeR, and timeline-based with weak constraint-based constructs, in the case of BIFS) which, at one hand, may potentially restrict the applicability of these languages in certain scenarios (e.g. those that require more expressive temporal contraints) but, on the other hand, can simplify authoring of presentations which are faced with less complex requirements. Many multimedia presentations can be properly represented via a timeline model, being a natural applications for timeline-based languages. This is the case, for instance, of presentations with a continuous media element as a "master track": a single video with a caption track [Laiola Guimarães et al., 2010] or even an annotated video with associated resources at specific time points [Sadallah et al., 2012]. In these cases, the whole presentation is already scheduled via the intrinsic timeline of the video, thus it would not take much advantage of a constraint-based model.

SMIL and NCL, on the other hand, present robust temporal models with considerable integration with hypermedia functionality, such as linking, which makes them serious contenders for applications with more complex timing and hypermedia needs. Both NCL and SMIL operate under constraint-based models which makes them suitable to deal with presentations with undeterministic events. Additionally, reuse features in these languages are flexible enough to allow integration of different time models inside a single presentation: in the case of NCL, via inclusion of declarative code (e.g. in other languages, such as SMIL) as media nodes and, in the case of SMIL, via inclusion of an external SMIL document (for instance by including a timeline-based presentation inside a constraint-based one).

[8]Lua Language. URL: `http://www.lua.org/`. Accessed July 2013.

2.7 Interactors and WaC: history, requirements and new contributions

In this section it is analyzed previous research upon which the contributions of this dissertation are built, encompassing the following themes: Interactors (Section 2.7.1) and WaC (Section 2.7.2). For each theme it is highlighted opportunities for improvements, new requirements derived from these opportunities and the ways in which these requirements are tackled in the dissertation.

2.7.1 Interactors

In capture and access applications, it is common to automatically generate an interactive multimedia document as an access interface to a captured experience (e.g. a videoconference, a meeting, a lecture, etc.). This multimedia document, usually called a multimedia session, aggregates all captured media streams as a single presentation. In certain scenarios, an author (e.g. a teacher publishing its lectures) may be interested in making adjustments in a multimedia document before publishing it, for instance by dropping certain segments or highlighting important information, without engaging with elaborate multimedia authoring tools. In order to meet these requirements, Interactors has been developed, along several iterations, as a family of multimedia operators to support amateur-oriented review, editing and enrichment of multimedia sessions. In order to analyze the historical evolution of Interactors, it is useful to organize it in two groups of results, each of them with a distinct focus:

Group 1: Inkteractors (Editing). The seminal group of operators, called Inkteractors [Pimentel et al., 2005], took advantage of recorded user interactions with boards using digital pen devices: first, all stroke generation and slide manipulation events were recorded with an instrumented capture environment; later, operators were applied over these logged events in order to customize the original board presentation [Cattelan et al., 2008b]. Such customization occurred via parameterization of certain attributes (e.g. authors, stroke formatting, spatio-temporal coordinates, etc.) of the captured interaction events. This was achieved via a taxonomy of operators that allowed two general groups of *editing* operations: expansion (e.g. generating more slides by varying the granularity of strokes) and filtering (e.g. removing slides by dropping strokes). As a result of applying an operator, expanded or filtered slide presentations were generated, as illustrated in Figure 2.8.

Group 2: Interactors (Enrichment, visualization and browsing). Provided that logged interaction events (e.g. user interactions with a board) have a spatio-temporal scope, it is useful for a user to navigate to specific points in the session using these events. This allows not only non-linear access to the session for skimming purposes (e.g. a student skimming only the slides in which a professor highlighted important information during the class) but also helps an author to identify the point most likely to be filtered or expanded. Targeting at this

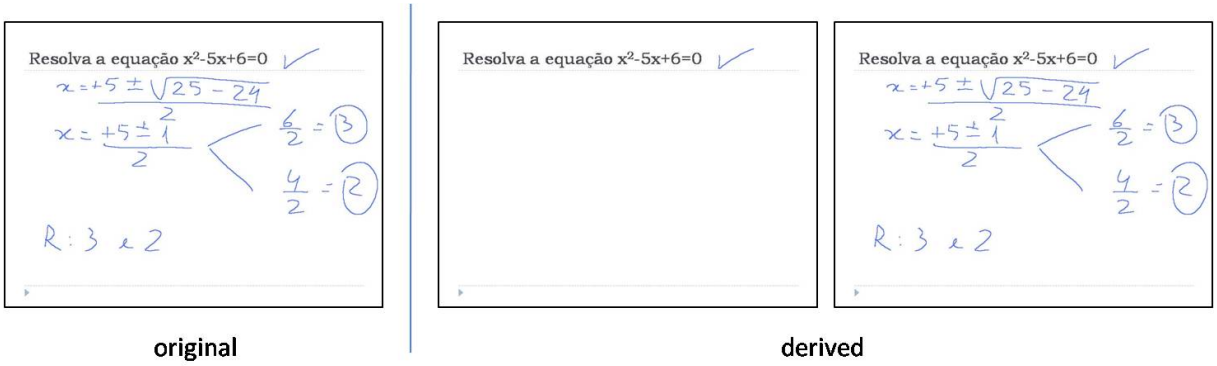


Figure 2.8: Inkteractors example [Cattelan et al., 2008b]. Inkteractors *IdleTime* applied over a lecture session in order to expand a slide into two new slides, thus segmenting the problem statement and the ink-authored solution.

opportunity, such interaction events were experimented as surrogates to points of interest in a session: an NCL multimedia document *enriched* with an interactive timeline could then be automatically generated that would allow these operations [Martins et al., 2011a]. Later, the Interactors model was extended to encompass not only ink-based events but also interactions via audio (AudioInteractors), text (TextInteractors), interactive surfaces (BoardInteractors) and video (VideoInteractors) [Vega-Oliveros et al., 2011a] — Table 2.2 summarize some of the defined Interactors and emphasizes their labels, their categories and the restriction applied to event space in order to obtain them. As the number of operators in a document grew, a better timeline-oriented visualization scheme was deemed necessary: for that purpose it was proposed the filtering of the visualization using a logic-based combination of the operators in the point-based temporal domain [Vega-Oliveros et al., 2011b], as illustrated in Figure 2.9.

Experimentation with both groups of results lead to identification of new requirements. In the following it is discussed the most important issues that were observed and their associated requirements:

- **Authoring requirements**: whereas in Group 1 there was a focus on *manual authoring*, i.e. on supporting a user in fragmentation operations, in Group 2 there was a focus on *automatic authoring*, i.e. on generating a final-form multimedia document for access purposes. As a consequence, Group 1 potentially supported the (manual) post-production and extension stages of a multimedia production process, whereas Group 2 potentially supported the (automatic) post-production and access stages of the process. Therefore, it is desirable a solution that integrates both automatic and manual authoring approaches via Interactors: such a solution should allow using the operators, in an integrated manner, in the post-production, access and extension stages of a production process.

  Dissertation contributions: support for multiple authoring approaches using Interactors is a longitudinal issue that orchestrates most of the dissertation contributions. The I+WaC-IE conceptual model (Section 4.2) provides a standard representation for interaction events

Table 2.2: Examples of Interactors.

Interactor label	Category	Restriction applied
filterByAttribute	All interactors	a list of media element attributes is defined
filterByAttributeValue	All interactors	a list of mappings between media element attributes and corresponding values is satisfied
idleMoments	All interactors	no interaction of a specific type has taken place
enterAudioMute	AudioInteractor	an audio element is muted
exitAudioMute	AudioInteractor	an audio element is unmuted
outstandingMoments	AudioInteractor	outstanding voice behavior (discussion, debates, etc.)
spokenMoments	AudioInteractor	voice interaction
voiceIncrease	AudioInteractor	consistent increase in the volume of one or more voices
changeBoard	BoardInteractors	a change of projected images over a board
boundariesDistance	Inkteractor	two or more ink strokes are drawn with a specified distance between their boundary boxes
changeOnArea	Inkteractor	an editing operation of ink stroke in a specified area over a board
changeOnAttribute	Inkteractor	an attribute of some media element was edited (e.g. the color of an ink stroke, the font of a text message)
changeOnAuthor	Inkteractor	a change of authors in some live editing operation over some media element
changeOnAuthorRole	Inkteractor	a change on the role of authors in some live editing operation
changeOnDrawingMode	Inkteractor	a change from drawing to erase mode
drawingDistance	Inkteractor	two or more ink strokes are drawn with a specified distance between their start and end points
filterByArea	Inkteractor	an ink stroke is drawn in the specified area
filterByAuthor	Inkteractor	a editing operation performed by a specific author
filterByAuthorRole	Inkteractor	an editing operation performed by a specific author role
textMoments	TextInteractor	a text message occurs

Figure 2.9: Interactors review and visualization prototype [Vega-Oliveros et al., 2011b]. In the background, an NCL document automatically generated from a captured session. In the foreground, the visualization query panel that allows filtering data points in the annotated timeline using set-based operations.

that can be used along all stages of the multimedia production process. Regarding automatic authoring, events represented with this model are transformed into an automatically generated SMIL document for flexible browsing of multimedia sessions. Regarding manual authoring, reuse features in SMIL documents were defined and experimented in a reference implementation: these features, when combined with document manipulation operators, allow non-destructive enrichment or editing of a multimedia presentation. On a more concrete level, all these issues are demonstrated in the I+WaC editor prototype.

- **Operator composition requirements**: in Group 1, editing operations could only be applied in isolation, that is, no semantics was prescribed to allow the composition of multiple operators, for instance via an algebraic expression. As a result of each operation a new document was generated. The most direct means of combining operations was for every operation to output a new document and feed this document to the next operation, similarly to a processing pipeline. Although simple to design, such approach is potentially inefficient provided that unnecessary document transformations could be performed because of a poor ordering of the operations. An algebraic composition, on the other hand, would potentially contribute with formal properties that subsidize optimizations: for instance, by merging overlapping fragments and optimizing the order of operations. As a first step, the operators on Group 2 tackled a first attempt towards an algebra: but, in this case, the operators were defined not for authoring purposes, but for filtering data points in a timeline-oriented

visualization. As a consequence, the algebra was defined based on axiomatic set theory and considered only time points — this would hinder the composition of operations in the continuous (interval-based) temporal domain. Tackling these issues requires the definition of an algebra that allows selection, fragmentation and composition of media elements in the continuous (interval-based) temporal domain.

Dissertation contributions: this requirement has been tackled via the definition of IAMm-DocT, the Interactors Algebra for Multimedia Document Transformation. The algebra includes operators for selection, fragmentation and composition of media elements in the continuous temporal domain, thus subsidizing both browsing tasks (by application of filters over temporal visualizations) and authoring tasks (by combining multiple annotations in order to generate new versions of multimedia documents).

- **Interaction style**: as proof-of-concept strategy to demonstrate the operators, prototypes were developed. Whereas in Group 1 prototypes emphasized a desktop-based interaction, in Group 2 prototypes emphasized a lean-back interaction style via interactive TV equipment, thus prescribing the use of a remote control as input device. After experimenting with several configurations of such environment, it was observed the need to efficiently visualize and select a large number of annotations, which created demands for lean-forward interaction styles with richer navigation and selection functionalities. Provided that the choice of interaction style influence the choices of target platform and underlying interaction techniques, tackling these issues requires design of infrastructure, tools and interaction techniques for lean-forward editing, enrichment and review of multimedia documents via Interactors operators.

  Dissertation contributions: these requirements has been tackled in a layered approach. On a lower layer, a SMIL presentation engine with live editing features has been designed and prototyped in a web environment. Such a building block contributes with an infrastructure for transformation of an active multimedia document for authoring purposes. On a higher layer, this infrastructure has been integrated in a web-based multimedia editing tool (I+WaCEditor, which allows editing, enrichment and sharing versions of a multimedia document via Interactors.

2.7.2 Watch and Comment (WaC)

When watching a video, it is common that users attempt to make comments about the underlying content. For instance, regarding an entertainment scenario, a user watching a video on TV may talk with someone in the same room about the program. Or, in a educational scenario, a student may bookmark or highlight some fragment of a video for keeping attentional cues. Taking advantage of users' commenting and collaborative/social behavior while accessing multimedia content, Watch and Comment (WaC) has been proposed as a paradigm for multimedia author-

ing [Cattelan et al., 2008a]: while accessing and making (collaborative) comments over a video, an interactive multimedia document is generated as a result of annotating. Consequently, editing a multimedia document could be associated to the general practice of annotating documents.

Initially experimented on TV watching scenarios, a prototype named WaCTool [Cattelan et al., 2008a] enabled users to apply multimodal comments (e.g. voice, text and ink annotations) and editing commands (e.g. seek, pause, skip, loop, etc.) in a video using the remote control (Figure 2.10). Every editing operator had an associated interactivity behavior: a group of seek operations generated a table of contents, a pause operation made the video be paused until resumed by the user, a multimodal comment also paused the video with a icon requesting if the user wants to see the comment, and so on. After an editing session, an edited/enriched version of the original video was generated as an NCL document. As an alternative for explicit editing commands, Inkteractors (refer to Section 2.7.1) were integrated into the tool, so that the operators could then be used to fragment the video: provided that a document was annotated using digital ink, a user could extract frames from the video using filtering and expansion operations. Additionally, users could collaborate with other viewers via text-based messages and share the edited content over a network.

Figure 2.10: An interactive video generated via WaCTool [Cattelan et al., 2008a] with icons notifying that the intended result of an editing operation can be activated in a video interval. From left to right: skip, loop and slow motion operations.

Later, the CWaC (Collaborative WaC) tool [Fagá et al., 2010] was proposed to allow not only collaboration via text messages but also synchronous multimodal annotation: in this case, geographically distributed users could annotate a single video at the same time (Figure 2.11). Collaboration issues were also developed in another extension tackling mixed collaboration: a group of distributed users watching TV could use personal mobile devices to bookmark [Teixeira et al., 2010b] or apply stickers [Teixeira et al., 2011] to a video: later, users gather at a location (e.g. a class) and these annotations are merged in a media center (e.g. a TV set-top-box); finally, as a result, an interactive multimedia document is generated to summarize the annotations.

Analysis of these research iterations allow the identification of new requirements concerning the following issues:

- **Authoring flexibility**: WaC, as a multimedia production technique, has been associated with manual authoring, either explicitly, in the cases where editing commands are applied,

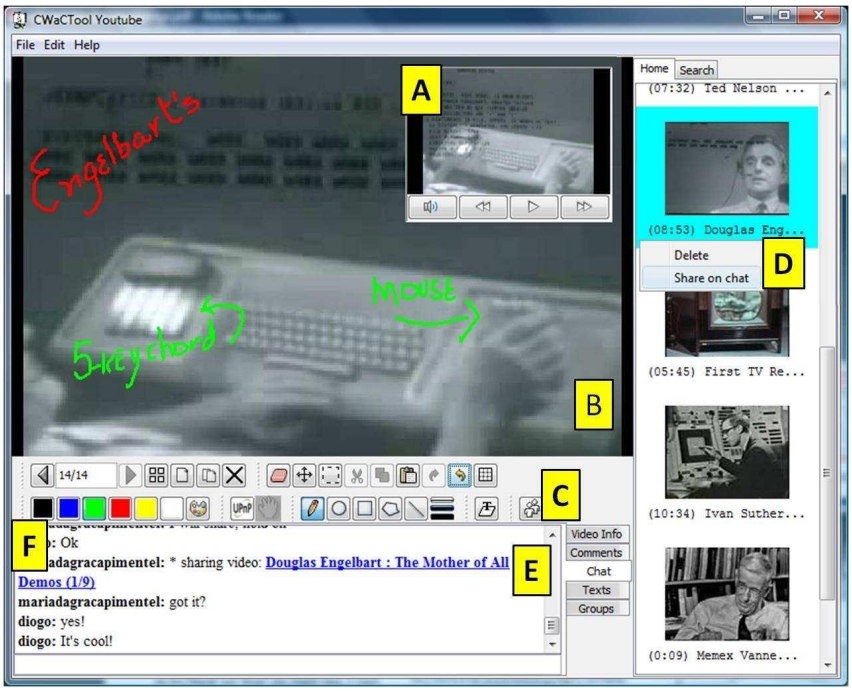


Figure 2.11: Collaborative annotation of video frames using CWaCtool [Fagá et al., 2010].

or implicitly, in the cases where annotations are used to add interactivity to a video. In the case of implicit authoring, an important contribution is taking advantage of natural commenting behaviors in order to author/enrich multimedia information. In the original model, every user annotation is directly included in the resulting document. But a user may annotate a document for various purposes, and some of these annotations may be for private use: therefore, a user may not be interested in including them in a customized document. Additionally, a user may be interested in generating different versions of a annotated document depending on the content of the annotations: for instance, when watching a TV program, a user annotates a fragment that would most likely be shared publicly whereas another fragment would be shared with a family member. Naturally, a feature-rich authoring tool could be used if more flexibility is an issue — but what is interesting about this problem is to keep the amateur-oriented simplicity of "authoring with comments" while allowing more flexibility whenever it is needed. Such issues demands design of models and interaction techniques to provide fine-grained selection and control of which annotations are included in a generated multimedia document.

Dissertation contributions: in this dissertation, this issue is tackled by integrating both Interactors and WaC in a single approach. Such an integration has been advocated before, with a different focus: Pimentel et al. [2007], on the original WaC proposal, envision the integration of Interactors and WaC via a timeline generation approach. According to the original idea, upon using ink operators (filters and expanders), visual annotated timelines should be automatically generated as a result. As a consequence of this functionality,

browsing functionalities could be dynamically incorporated to an (annotated) multimedia document. On the side of concrete realizations of this vision, previous work has tackled it only to a limited extent: the original WaC prototype included support for ink operators, but the operators did not lead to live document transformations as long as it allowed only extraction of video frames [Cattelan, 2009]. As long as live editing were present in the original WaC tool, it was restricted to enrichment commands (e.g. comment, pause, loop, skip, etc). Interactors research, on the other hand, has defined new operators but it has distanced from manual authoring. Besides, generation of timelines still do not tackle the problem of allowing fine-grained selection of which editing results to include in the final document.

The contributions of this dissertation not only tackle this integration in full extent but also advance it further: besides dynamic (live) generation of timelines for browsing purposes, it also allows temporal transformation of the original session, as a whole, in order to extract fragments. In order to realize these issues, the I+WaC-IE model provides a first level of integration by abstracting both implicit interaction events and intentional annotations in the same model. In a further level, the use of the IAMmDocT operators, associated to reuse extensions developed for SMIL, allows the specification of document transformations with specific annotations: these transformations subsidize non-destructive editing of the original document. On the side of interaction techniques, the I+WaCEditor realizes a prototype for browsing, enriching and editing a multimedia document using both approaches.

2.8 Final remarks

In this section it is reviewed the contents of the chapter and it is discussed the relationships of each theme with the research reported in this dissertation. The first and more general issue discussed, that of general *multimedia production*, is of uttermost importance provided that this dissertation contributions on document engineering have been directly applied to amateur productions. It is true that any production, regardless of being professional-oriented or amateur-oriented, can potentially take advantage from easier and smarter methods for capture, post-production or any other activity encompassed in a production process. But, as extensively discussed in this chapter, supporting amateur productions has a greater appeal because, on the absence of adequate computational support, the difficulty of the task might lead a user to avoid engaging in enriching or editing activities. It was demonstrated in this chapter that, among various abstractions for multimedia production, understanding the process from the perspective of capture and access provide several benefits for tackling user-oriented production workflows. First, it opens up possibilities for end-user enrichment of both professional and user-generated content. Second, capture and transfer of annotations along the production workflow potentially assists editing and enrichment tasks. In summary, a proper theoretical understanding of multimedia

production processes help not only to delimit all the issues and challenges involved in the research theme but also to situate the contributions in reference to this framework: in particular, the contributions of this dissertation allow contemplating issues in most stages of the reference multimedia production process, as will be more thoroughly discussed in the following chapters.

Regarding *multimedia annotation*, it is advocated in this dissertation that annotations (seen as either metadata or additional content) are key resources for supporting organization, reuse and extension of multimedia content. The literature has consistently demonstrated the importance of annotations to organize information. But in this dissertation, there is interest not only on organization benefits, but also on the role of annotations to take advantage of combined curation and extension of multimedia content. On the side of curation, an infrastructure for live document transformation, augmented with an authoring tool, allows enrichment of documents via temporal annotations, which optionally can serve as editing commands, over multimedia presentations. On the side of extension, enriched documents can be transformed, via a family of operators and a set of document reuse features, into new versions potentially customized in fine granularity.

As discussed earlier, *multimedia browsing* is an key activity in multimedia production, as far as it subsidizes processes of idea generation, review of information, gathering material for reuse, and so on. In this dissertation, as a supporting contribution to take advantage of annotations, intra-media browsing techniques are realized in the Interactors+WaC tool: these encompass interactive timeline visualizations (to navigate multimedia documents via temporal annotations), query features (to filter groups of annotations) and adaptive playback mechanisms (which allow to create playlists of fragments based on a list of annotations).

Towards supporting amateurs on creation and extension of multimedia content, much research on *multimedia authoring* has been concerned on finding a middle ground between automatic and manual methods by adopting "user in the loop" approaches. Some of these approaches have taken advantage of annotations, collected at various opportunities along the production process, in order to subsidize authoring. This dissertation follows along these lines, by allowing amateurs to take advantage of annotations in order to enrich and edit multimedia content without the need to engage in traditional authoring tools. Here the focus is less on demonstrating authoring tool functionalities but more on investigating document engineering issues that are involved in processes of reuse, enrichment and extension of multimedia content. For that purpose, a declarative language for authoring interactive multimedia documents is taken as basis, over which important extensions for live editing and reuse of document components were designed and experimented with. Further on, these extensions are a core foundation to realize other contributions: primarily, an algebra for document transformations and, secondarily, a demonstration infrastructure for editing multimedia documents via annotations.

These contributions are realized by building upon previous work on enrichment, editing and review of multimedia documents: Interactors and WaC approaches. In particular, it was identified and discussed a number of new requirements to these themes that motivate not only a stronger integration between them but also, more importantly, the need to extend them in new

important ways. Additionally, it was demonstrated how such integration and extension issues are tackled via the models, operators, document engineering methods and concept demonstration tools developed in this dissertation.

Chapter 3

ActiveTimesheets: extending Web-based multimedia documents with live editing and reuse features

As discussed in Section 2.6, many multimedia languages for authoring purposes have been proposed, each with its tradeoff regarding spatio-temporal models, interactivity and expression power. In this dissertation there is a particular interest in languages that fulfill a set of specific requirements for enrichment and editing tasks, primarily in the context of Interactors and WaC authoring approaches. In particular, there is an interest in languages that fulfill the following requirements:

(1) **Timeline-oriented browsing**: certain multimedia presentations can take advantage of timeline-oriented browsing for seeking to specific moments in the presentation duration. This is generally accomplished by associating elements in a time-based visualization with interactivity behaviors in a multimedia document, for instance in the case of lecture browsers (e.g. skipping in a multi-stream lecture based on an index of slide transitions), meeting browsers (e.g. jumping to a point where some term was mentioned, also via a temporal index), or annotated documents in general (provided that annotations are temporally-indexed). In these cases, there is a need to associate interactivity behaviors and other media elements to the "global timeline" of the presentation. Because these scenarios involve presentations with deterministic durations, a timeline model is a natural choice to represent them. But, regarding extensibility issues, adopting such a model would severely constrain the types of constructs that can be added later to the document via enrichment or editing operations. For instance, it would be more difficult to represent constructs based on undeterministic events (e.g. like the need to activate a link to proceed to another presentation fragment). In summary, a multimedia language that fulfills this requirement should count on a timing model whose features allow both timeline browsing and constraint-based synchronization;

(2) **Versioning by reference**: in some circumstances, a user may be interested in modifying some multimedia presentation authored by others, for instance a student applying annotations over a lecture recording or extracting fragments of a lecture for use in other contexts. As a consequence of such modifications, new versions of the original document are generated. Given that in these scenarios a user may not have modification permissions over the original document, new versions can be generated either via *copy* (i.e. a new copy of the original document is made and, over the copy, annotations are applied) or via *reference*, or *reuse*, (i.e. the original document is encapsulated, via a reference, inside an annotation document). Besides generation of redundant information, versioning by copy also makes it hard to keep the consistency between the original document and the derived version: if the original document is updated by its author, the copy might be left out-of-date in the absence of some consistency maintenance mechanism. Versioning by reference, on the other hand, overcomes this limitation as long as the original presentation is obtained from the original source upon rendering. In summary, a language that fulfills this requirement should provide features that allow reuse of whole presentations and of presentation fragments with fine granularity;

(3) **Dynamic client-side modifications**: in applications in which end-users can apply enrichment and editing operations over an active multimedia document, i.e., a document that is being played, the effect of each operation on the document spatio-temporal layout can be static or dynamic. Static operations do not immediately affect the spatio-temporal layout, being postponed in a queue until a layout update operation is triggered. This approach is especially advantageous for layouts that are computationally expensive to render in real time, for instance in resource-constrained access devices. However, from an end-user perspective, a disadvantage of this approach is that the effect of the operation is not perceived immediately. On the other hand, dynamic operations are those that trigger a spatio-temporal layout update in the document model. As a disadvantage, the real-time cost of recomputing the layout may be prohibitive in certain configurations (e.g. document size, access device, etc.), which might demand a number of optimizations to make operation execution times acceptable: for instance, determining that depending on the severity of the operation, the layout will be re-rendered only partially. Certain applications particularly take advantage of dynamic operations: for instance, time-indexed annotations, once applied, can be immediately used to trigger a seek in the document presentation; or the effect of an editing command can be immediately tested. However, these operations cannot be costly to the point that user experience is negatively impacted. In summary, a language that fulfills this requirement should support dynamic client-side document modifications in an efficient manner.

Regarding versioning and transformation of multimedia documents, late-binded distributed multimedia presentations, as long as they are assembled at rendering time, present more ad-

vantages, given that client-side enrichment and editing operations in these languages do not potentially incur in costly packaging cycles. As a consequence, distributed multimedia languages are more adequate to fulfill part of the issues stated in requirement *(3)*. Among the most relevant distributed multimedia languages (refer to Section 2.6), requirement *(1)* calls for a constraint-based model: as a consequence, as far as an expressive constraint-based temporal model is desired to fulfill a range of operations, SMIL and NCL are serious contenders to fulfill this requirement.

Because of its hierarchical temporal model, SMIL stands out for fulfilling requirement *(1)*, given that it brings more flexibility issues while maintaining simplicity. This advantage contrast with the difficulties of modeling interactive timeline visualizations to the global temporal scope of constraint-based presentations, for instance in NCL. This problem has been investigated before by Vega-Oliveros et al. [2010b], which simulated a presentation timeline in an NCL document. Dealing with the problem of automatically authoring meeting browsers with temporally indexed annotations, the authors simulate a timeline by creating, at authoring time, a "virtual node" that collects the activation intervals for all annotations: therefore, the virtual node acts as global index for the document, over which spatio-temporal and interactivity constraints are applied, for each index entry, in order to enable random seeking to annotations of interest. Definition of such constraints leads the resulting document to contain extensive lists of rules to enforce synchronization of all streams upon every seek operation. The same problem has been studied, also regarding NCL for automatic authoring of lecture browsers, by Viel et al. [2013], which also identified the need for elaborate document engineering patterns, static code generation, and considerable amount of scripting to fulfill such a use case. In summary, both solutions suggest, from a document engineering perspective, high perceived complexity and verbosity in the underlying NCL document. SMIL, on its turn, potentially simplifies these designs, given that its hierarchical temporal model allows manipulating the global timeline of a constraint-based presentation while still keeping the simplicity of the resulting document. Such advantage is more concretely demonstrated in the following chapters, upon definition of an algebra for document transformation (Chapter 4) and a tool for editing and extension of captured sessions (Chapter 5).

Despite this advantage, SMIL presents shortcomings to fulfill requirements *(2)* and *(3)*. Regarding client-side editing, many multimedia languages support dynamic modifications over an active document, such as NCL, BIFS and LASeR. SMIL, however, currently has no formal model that tackles the problem of dynamically applying client-side transformations in a document. As a first step, preliminary organization and cost analysis of possible editing operations in SMIL has been reported by Jansen et al. [2010], but effective means of performing these operations in the SMIL spatio-temporal model has not been thought about as much. On the side of reuse features, SMIL supports it only to a limited extent, for instance it is possible to reuse a whole document or the content of media elements (given that media elements are included as references). Such limited support contrasts with reuse features in NCL, for instance, which supports reuse for several entities in a document [Soares-Neto et al., 2010].

Given this rationale, SMIL still stands out for part of the modeling requirements tackled in this dissertation, because its hierarchical temporal model allows to reasonably accommodate timeline-oriented browsing, which is an important issue to minimize perceived complexity in the final document. However, some other requirements are not properly tackled by current SMIL models and implementations, for instance dynamic modification of documents and fine-grained reuse of document constructs. In order to take advantage of this language benefits whereas tackling its perceived limitations, this chapter reports a group of language extensions developed to enable live editing operations and granular reuse in a subset of the SMIL language (SMIL Timesheets). These extensions are demonstrated in ActiveTimesheets, a Web-based multimedia presentation engine. Such contributions are relevant not only for the dissertation, but for multimedia authoring in general, as long as it increases the range of scenarios in which the SMIL language can be applied.

The remaining of this Chapter is organized as follows. Section 3.1 surveys related work on live editing, reuse features and synchronized multimedia engines, comparing them with the contributions of the Chapter. Section 3.2 delimits the scope of the research by discussing a subset of the SMIL language that is taken as a basis to build the extensions. Section 3.3 presents a formalization of the timegraph model and its algorithms, which is the basis for developing the novel extensions. Section 3.4 discuss the live editing model, detailing its operations, structures and algorithms. Section 3.5 reports the integration of linking in ActiveTimesheets, which also has contributions regarding syntax for addressing media fragments. Section 3.6 details the syntax and semantics of the reuse extensions provided by ActiveTimesheets. Section 3.7 describes an implementation of the ActiveTimesheets language in a Javascript-based engine. Finally, Section 3.8 discuss the contributions of the Chapter and opportunities for future work.

3.1 Related work

3.1.1 Live editing of multimedia documents

In multimedia production, the term “live” has been traditionally associated to the temporal overlap between capture and distribution, namely: a *live production* is simultaneously captured and distributed whereas a *non-live production* has no such overlap. Live production is common on live TV when programs are simultaneously captured and broadcasted [Perry et al., 2009], and also on the Internet, for instance via live streaming technologies [Mazzola Paluska et al., 2010]. On the other hand, *live editing*, when applied to a multimedia stream over a network, consists on transforming a stream while it is being transmitted and/or rendered: it can occur at *server-side* (or broadcaster-side), at some *intermediate point* in the transmission medium, or at *client-side* (or viewer-side). As a consequence, live editing can occur either for live productions (e.g. live TV programs captured with multiples cameras that are switched, on broadcaster-side, into a live stream being broadcasted to viewers) or non-live productions (e.g. content adaptation,

in which inner streams are modified, at server-side or at some intermediate point, for instance to include advertisements). When applied to a *multimedia document*, a live editing operation, as a counterpart to authoring, can affect temporal, spatial and interactivity relationships of the document. Depending on the changes performed by a live editing operation, the temporal and spatial layout of the document must be updated, incrementally or as a whole, in order to reflect the changes.

Updates on *spatial layout* (e.g. adding, updating or removing visual elements), in particular, has been tackled in several fronts. Consequently, it is a well understood problem, with solutions that generally encompass definition of data structures that can be manipulated at document presentation-time. This is the case of several visual formats, such as X3D [ISO, 2010] and SVG [W3C, 2011d], which define a scene graph for rendering purposes. In these formats, the scene graph can be manipulated and, as a consequence, the resulting graphics is partially or totally repainted. In the case of SVG, whose document structure is based on a general scene graph (which is populated with shapes and compositions), the document is converted, at runtime, into a DOM (Document Object Model) [W3C, 2004c], an object-oriented tree which can be manipulated via scripting. A DOM is also used for HTML (Hypertext Markup Language) [W3C, 2012d], which allows dynamic updates on the spatial layout of the document: certain modifications on the DOM can yield a partial or full update in another data structure, the document rendering tree, which describe the painting order of every visual element in the document. In all of these formats interactivity can also be dynamically manipulated, for instance by adding event listeners and links. Additionally, all these formats are Web-based, thus traditionally focus on client-side live editing. However, server-side live editing can also be achieved in the cases in which scripting is supported, for instance via execution of asynchronous network calls.

Updates on *temporal layout* are particularly relevant for languages that count on synchronization constructs. This is the case of MPEG-4 BIFS [ISO, 2005], for instance, which provides a command-oriented live editing approach. In this format, multimedia presentations authored in a declarative language (either XMT-A or XMT-O) are converted to a binary format (BIFS) which includes a low-level scene description language and a command-based language. Server-side live editing is expressed in the BIFS language, via editing commands sent by the broadcaster through the presentation stream. As a consequence, a scene graph in the presentation stream can be manipulated in fine granularity, by modifying nodes and their attributes, and even whole scenes can be replaced. Client-side live editing can be triggered via user interaction events that trigger special-purpose scene graph nodes (e.g. conditional, scripting and server-command nodes), which can yield a local update (i.e. modification of the currently rendered scene) or a remote update (i.e. a roundtrip to the server that responds with a editing command in the presentation stream). A similar approach is adopted by LASeR [ISO, 2009a], which also adopts a scene graph model, based on SVG, and a binary format for transmission (Scene Aggregation Format, or SAF). Client-side editing is performed via interaction events and a special group of graph nodes that extend the SVG set (conditional and updates nodes). Server-side editing occurs

via commands sent through the binary SAF stream.

In NCL [ITU-T, 2011], live editing has been designed for TV platforms, therefore it also occurs via a command-oriented approach. In this language, declarative documents are transformed, at runtime in the client, into a graph-based data structure (Hypermedia Temporal Graph, or HTG), which is employed for planning the synchronization and the transmission of media elements. Server-side live editing occurs by sending commands which are conforming to the document syntax: as a consequence, the document itself is changed by each command, being necessary to translate the changes to the HTG according to predefined command-specific rules. Client-side live editing, on the other hand, is not directly supported via commands. This limitation has been studied by Pimentel et al. [2010], which extended an NCL execution environment in order to allow client-side editing via custom high-level operations. In this solution, after an editing operation is applied, the document is replaced by a new version that is then formatted from scratch, consequently incremental updates are not supported.

The SMIL [W3C, 2008b] format, differently from BIFS, LASeR and NCL, has no support for live editing operations. SMIL is a language based on XML [W3C, 2008] but, differently from other structured web documents, such as HTML and SVG, a SMIL DOM modification does not lead to updates in the document temporal layout. Spatial layout, on the other hand, might be updated depending on the underlying execution environment (for instance, when HTML is a host language for SMIL [Gaggi and Danese, 2011]). According to the SMIL recommendation, at runtime, the temporal layout of a document is transformed into a graph data structure, called *timegraph*, composed of media elements and their compositions. The timegraph for a document is essentially decoupled from the document DOM, as a consequence any update in the DOM is not naturally reflected in the timegraph. As a naive solution to this problem, upon detecting a DOM modification, an execution environment could save the timegraph state, rebuild it and restore its state: however, besides being potentially costly, this alternative can also disrupt the seamlessness of the presentation. Consequently, solutions that allow incremental updates over a document are a desirable feature. As far as document modification is officially supported in SMIL, a limited range of behaviors can be configured via its State Module [Jansen and Bulterman, 2009], which allows definition of document variables whose values can be assigned at runtime, for instance via interaction events or via external applications (in the latter case, depending on specific support in the execution environment). However, such variables can be used only in XML elements and attributes where a value substitution is applicable, for instance as value in node attributes or as content in text nodes. Consequently, the State Module does not allow more complex operations, such as adding and removing nodes and attributes in the document.

The need for arbitrary modifications in SMIL has been preliminary studied by Jansen et al. [2010], which proposes an abstract classification of editing operations and their expected costs regarding timegraph updates. Their study organizes SMIL functionalities into four “clusters” (Selection, Adaptation, MediaItem and Structure), of which timegraph updates are expected only on the MediaItem (in some cases) and Structure (in all cases) cluster. The case study

conducted by the authors demonstrate modifications on the Selection, Adaptation and MediaItem clusters, the latter with a set of operations that does not generate timegraph changes. Given that it does not present methods to tackle the MediaItem (as a whole) and the Structure clusters, the case study does not tackle the problem of updating temporal layout. Additionally, their classification lack cost estimation in several important operations, such as timing manipulations (e.g. live speed modifications), and have uncertain estimations for fundamental timing attributes (e.g. `begin`, `end` and `dur`). Even though very insightful, their model, as advocated by the authors, is regarded as a first abstract step towards designing SMIL live editing functionality. For that reason, this chapter builds upon this classification and report the ActiveTimesheets live editing model that, instead of the full SMIL language, encompass incremental temporal updates over a SMIL-based temporal language. The model is properly defined and the possible editing operations are demonstrated with their associated behaviors and costs. Additionally, the model is concretely demonstrated in the ActiveTimesheets engine for Web-based multimedia documents.

3.1.2 Reuse features in multimedia documents

Reuse in general computer languages is advocated not only to reduce development costs but also as a means to improve maintainability of applications. In the case of imperative languages, such concerns have led to development of libraries, toolkits, components, frameworks and design patterns, which take advantage of variable encapsulation so as external code can be used as building blocks. In the case of declarative languages, the importance of reuse is also warranted by means of specific language features that support such activity.

In HTML, for instance, reuse of external content can occur via the `link` tag which allows, for instance, reuse of external style information in CSS (Cascading Style Sheets) [W3C, 2011a] or reuse of external procedural code in Javascript. The `iframe` tag allows the inclusion of whole external webpages in a framed spatial region. Additionally, reuse of external media content is allowed via the `img`, `audio` and `video` tags, for instance by including the same media element multiple times in a single document. However, HTML does not provide general means to declarative reuse fragments of HTML documents, neither internally nor externally. A potential reuse feature, fragment identifiers, i.e., addressing named document fragments via a hashtag in a URI (Universal Resource Identifier) [IETF, 2005], is supported in this language. However, this resource is used mostly for specifying intra-document link targets, and has no application for reuse.

Reuse of media content can be further extended in HTML by means of the Media Fragments URI [W3C, 2012b] scheme. This W3C recommendation provides a multidimensional scheme to address discrete and continuous media elements. Therefore, each media element can be fragmented via a URI syntax which encompasses the following dimensions: spatial (e.g. a region of an image or a video), temporal (e.g. an interval in a video or an audio file), track (e.g. only the audio track of a video file) and id (e.g. named fragments inside a media element). The Media

Fragments recommendation provides only the URI scheme for addressing fragments, being up to the user agent to decide how the fragment will be resolved. Depending on how the user agent support the mechanism, interesting reuse scenarios can be achieved in HTML, for instance including a cropped version of an image or only a clip from a video.

In the case multimedia-oriented languages, SVG, for instance, allows intra-document reuse of geometrical shapes via the combined use of definition (`def`) and reuse (`use`) tags. Inter-document reuse is also possible, by means of references to external XML namespaces. An extensive support for reuse is provided by NCL, which allows document constructs to be reused in fine granularity. For instance, intra-document reuse is supported for various constructs, such as media content, layout, structure and rules; inter-document reuse is also supported, for instance by nesting NCL documents and importing specific objects from external documents.

In SMIL, reuse is limited to media content, which is possible by using the same media URL in multiple media definitions (e.g. two `video` tags sharing the same URL in the `src` attribute). In the case of continuous media content, each media definition can, optionally, be customized to a fragment in the underlying media content by clipping a subinterval of its intrinsic timeline (e.g. via `clipBegin` and `clipEnd` attributes). Some reuse features are hard to devise in SMIL, given that the language present a tight coupling between structure and temporal layout: i.e., the document structure of the document `body` occurs mostly via time containers, which also enforce the temporal layout. As a consequence, an attempt to reuse a fragment in the body of the document imply a reuse of the underlying temporal layout. Despite these intrinsic limitations, SMIL has a lot of room to incorporate reuse features in the language.

SMIL Timesheets [W3C, 2012a], a temporal language based on SMIL, for example, provides a few reuse features. Every timesheet document defines a root element, `timesheet`, whose attribute `src` can refer to an external document. The reuse feature is realized to the extent that multiple `timesheet` elements can be nested: as a consequence, whole external documents can be embedded using the `src` attribute in the inner `timesheet` elements. Additionally, this language, being purely temporal, employs a pattern-based binding mechanism to associate temporal specifications to spatial fragments. This is is achieved via the `select` attribute in the `item` element. A consequence of this mechanism is that the same spatial fragment can be used in multiple `item` elements, consequently the language also allows reuse of spatial structure. Besides these improvements, it still shares some of SMIL limitations, for instance not allowing reuse of individual elements in the temporal document.

This chapter reports contributions on this issue, by proposing features that allow document reuse (import a whole external document), reuse of fragments both from internal and external documents, and clipping of compositions (e.g. fragmenting time containers in a subinterval upon reuse). These contributions are demonstrated at the language level (via new syntax constructs) and at the execution level (by demonstrating their impacts in the timegraph).

3.1.3 Synchronized multimedia on the Web

The adoption of multimedia technologies on the Web is highly dependent on whether user agents (web browsers) support them. Even though technologies such as SMIL have been recommended by the W3C for considerable time, up until a few years ago, user agents have delegated most of multimedia presentation functionalities to external, self-contained execution environments that could be embedded in a Web document via plugins. This delegation was part because Web browsers did include native support for multimedia presentation formats, for various reasons, such as licensing issues over media formats, complexity of multimedia presentation formats, among others. This has lead to the widespread adoption of technologies such a Flash[1], Silverlight[2] and Java[3] as long as general multimedia (e.g. audio, video, 2D and 3D animations) content was needed in a Web page. Tackling at potential portability limitations and a lack of open standards in these technologies, HTML5 [W3C, 2012d] includes a number of extensions toward native multimedia functionalities on the Web: continuous media playback, inline vector-based graphics, state machine -based raster graphics, access to multimedia capture devices (microphone and webcam), just to name a few.

Despite these advances, HTML5 still lacks support for synchronized multimedia presentations. Even though continuous media streams can be included via `audio` and `video` tags, or via self-contained plugins controllable by scripting, there are very limited resources for specifying the temporal layout of media elements. In most of the cases, declarative synchronization constructs are supported only in simple layouts in which a single continuous element is a master track, for instance in subtitles (`track` tags) or in lipsync enforcements (media controllers specified via the `mediagroup` attribute), consequently limited to a timeline model. Such limitations have lead to development of multimedia synchronization technologies that take advantage of native multimedia support in user agents. One popular example is Popcorn.js[4], a procedural approach (scripting-based) that enables synchronization of arbitrary content (e.g. images, frames, document fragments, etc.) to the intrinsic timeline of a continuous media element. As an alternative to a procedural language, timeline synchronization can also be achieved in a declarative document in XML: in this case, a scripting-based interpreter generates the presentation schedule. A limitation of this solution is that it is still based on the timeline model, as a consequence the expression of more complex temporal relationships (e.g. in a presentation with multiple videos and no master track) demands a lot of application-specific scripting code that increases the complexity of the resulting document.

As an alternative to script-based authoring, some researchers have attempted to adapt declara-

[1] Adobe Flash technology. URL: `http://www.adobe.com/products/flash.html`. Accessed on July 2013.

[2] Microsoft Silverlight Technology. URL: `http://www.microsoft.com/silverlight/`. Accessed July 2013

[3] Java Media Framework API, URL: `http://www.oracle.com/technetwork/java/javase/tech/index-jsp-140239.html`. Accessed July, 2013.

[4] Mozilla Popcorn.js HTML5 Media Framework. URL: `http://popcornjs.org/`. Accessed July, 2013

tive multimedia languages to the HTML5 technology stack. Here these approaches are classified in two categories: *i) spatio-temporal engines*, which take as input a declarative multimedia document, encompassing both spatial and temporal layout, and generate as output a multimedia presentation represented in HTML+CSS+Javascript; and *ii) temporal engines*, which take as input a multimedia document encompassing only temporal layout whereas spatial layout is authored separately in HTML+CSS.

Regarding *spatio-temporal engines*, an example is WebNCL [Melo et al., 2012], a Javascript-based presentation engine that enables embedding of NCL documents in Web pages. Upon a script-based initialization, the engine translates spatial layout to HTML+CSS code that is dynamically inserted into the HTML DOM; temporal layout is controlled via an event-based formatter in Javascript. A related but slightly different approach is followed by NCL4Web [Silva et al., 2013], which statically translates an NCL document into a bundle of HTML5+CSS+Javascript code. In this case, the translation process is not performed via procedural code, but via a declarative XSLT document. Another example is SmilingWeb [Gaggi and Danese, 2011], a scripting-based engine that at runtime also tackles both spatial and temporal layout: taken as input a document, in a subset of the SMIL language, it dynamically translates the spatio-temporal layout of the document to HTML+CSS+Javascript.

As an advantage, spatio-temporal engines are more loyal to a pure design in the underlying declarative languages, i.e. NCL and SMIL, both of which have been originally designed for spatial and temporal layout. But, regarding their execution in the HTML5 stack, this can be also a disadvantage, since expressing spatial layout in SMIL or NCL inhibits an author to take advantage of the (potentially more expressive) spatial layout features of HTML. This has also implications for live editing, i.e. an editing operation would need to trigger an update in the generated code: a naive solution, but also potentially costly, consists on re-generating the whole document, as long as a more performance, incremental, updates would require elaborate document transformation mechanisms to map source and target languages. Additionally, such an approach would duplicate robust live editing features that are already supported in HTML via DOM manipulations. All of these issues may be reasons for these approaches, as far as their documentation goes, do not officially support live editing operations.

Temporal engines have potential to overcome part of these issues as long as they extend the HTML5 stack with timing functionality, i.e. they take advantage of HTML spatial layout features whereas extend the language with temporal layout features. XHTML+SMIL [W3C, 2002], for instance, extends HTML tags in order to include most of SMIL timing-related constructs via namespaces: in this case, the multimedia document is authored only in HTML. As a disadvantage, this solution entails a syntactic coupling between spatial layout and temporal layout (i.e. temporal constructs are applied directly to spatial tags) which is not always a feasible document pattern. The Ambulant Annotator [Laiola Guimarães et al., 2010] is a solution implemented via scripting that allows subtitling web videos using the smilText [W3C, 2008e] module. As a disadvantage, the engine focuses on demonstrating a specific SMIL

module in single-video multimedia presentations, which makes it unsuitable with multiple video presentations with more elaborate synchronization constraints. Syntactic decoupling between spatial and temporal layouts has been tackled by SMIL Timesheets [W3C, 2012a] language, which is inspired by the idea of structure-style separation in CSS: temporal layout definition is kept separate from the HTML spatial structure, either in the HTML header or in a separate file. As a consequence, little modification is needed on HTML syntax. In order to integrate both layouts, CSS selectors allow temporal tags to be associated with spatial tags. This language has been implemented in the Timesheets.js [Cazenave et al., 2011] engine, which supports most of the SMIL Timing module.

Besides taking advantage of HTML spatial layout features, temporal engines potentially make live editing operations more manageable. This is because spatial editing is delegated to the underlying host language whereas only temporal editing should be a concern. But such separation also brings disadvantages, since spatio-temporal consistency may be more difficult to attain as long as both dimensions are kept decoupled. For instance, additional mechanisms are necessary to make sure that spatial modifications are updated in the temporal layout as well. Ambulant Annotator, for instance, support presentation-time editing, as long as it enables subtitling a video during playback. But this feature is restricted to timed text functionality and the does encompass important types of operations, such as removal of media elements. Timesheets.js partially support live editing, as an experimental feature, with a number of limitations. First, of all possible language nodes, only time containers can be added (but not removed or modified) via a public API. Second, a time container is added directly to the internal model, keeping the original in-memory XML document unaltered, which can lead to runtime inconsistencies between the two models, in case timing re-computations are needed (for instance, when some script alters the timesheet DOM). Third, spatial modifications in the HTML DOM (for instance adding nodes) are not reflected in the temporal document, which may lead to inconsistencies in the spatio-temporal binding mechanism. Fourth, the architecture of the engine does not separate concerns of parsing, scheduling and the timegraph data model, which is part of the reasons for the difficulty of providing live editing in a broad and consistent manner in this engine.

In order to present solutions to part of these problems, this chapter presents a SMIL Timesheets engine (ActiveTimesheets) with advantages when compared to previous attempts. First, the engine supports live editing of all constructs directly in the document, via a DOM API, and the changes are automatically translated to the document temporal layout, by means of timegraph manipulations. Second, ActiveTimesheets presents a few alternatives for achieving spatio-temporal consistency in case of live modifications in the spatial document. Fourth, the architecture of the engine is modular and defines a flexible timegraph model which can easily be manipulated and modified.

In addition to tackling these limitations, ActiveTimesheets also brings additional advantages. First, the engine extrapolates the SMIL Timesheets recommendation by including not only SMIL's Timing module, but also an extended version of the Linking module [W3C, 2008c], an

extended version of the Time Manipulations module [W3C, 2008f], and the MetaInformation module [W3C, 2008d], all of which are here advocated as very important to a proper HTML-SMIL integration. Second, the engine supports the HTML5 Media Fragments [W3C, 2012b] recommendation which, besides allowing a SMIL document to be addressed in fine granularity, contributes to make a SMIL document a first-class media object in a HTML5 page. Finally, the engine incorporates a number of SMIL custom extensions to allow reuse of constructs in this language.

3.2 Subsetting and extending the SMIL language

SMIL 3.0 is an extensive W3C recommendation that encompasses a variety of multimedia presentation issues: spatial and temporal layout, content adaptation, metadata, data models, subtitling, linking, etc. Definition of different profiles for different usage contexts is a recommended practice for this language, which in fact includes a few predefined ones, such as a talking book profile and a mobile profile. In order to achieve such configurability, the SMIL recommendation is currently organized in 12 modules that can be combined in specific profiles and in different host languages. SMIL Timesheets, for instance, take advantage of this configurability by defining a new language built upon three SMIL modules.

On defining the scope of ActiveTimesheets, there is an interest in choosing a specific combination of SMIL modules that support the requirements identified in the introduction of this chapter. For this purpose, ActiveTimesheets has SMIL Timesheets as a basis and includes additional modules. However, integration of additional SMIL modules in the Timesheets language is not always a seamless process, especially in the case of modules that depend on SMIL's spatial layout. Also, the presence of new tags that are Timesheets-specific add more complicating factors to this integration. In order to clarify these issues, in the remaining of this section it is provided an overview of the original SMIL timesheets language (Section 3.2.1), followed by a brief discussion of additional SMIL modules and extensions which are aggregated in the ActiveTimesheets language (Section 3.2.2). After properly contextualizing the functionalities present in ActiveTimesheets, the most important issues of the language are discussed in more detail in the remaining of the chapter.

3.2.1 The SMIL Timesheets language

SMIL Timesheets, or simply Timesheets, is a markup language that allows SMIL's temporal layout model to be incorporated in various a-temporal XML host languages. A natural host language for Timesheets is XHTML, whose feature-rich spatial layout and style model can be combined with SMIL's temporal synchronization in order to author web-based multimedia documents. The design principle of Timesheets is inspired on the separation between style and structure provided in CSS — but, in this case, the principle is adapted to denote separation

between temporal structure and spatial structure. Consequently, a spatial document (in XHTML, for instance) can be assigned different separate, loosely-coupled, timesheets in different circumstances. Even though Timesheets has been designed for XML-based languages, it can be easily integrated in non-XML based languagues, such as HTML5. Currently, as far as public documentation goes, Timesheets has not been implemented in any major user agent, consequently its functionality, as long as interoperability is a concern, is commonly implemented via scripting.

The Timesheets language reuses all SMIL timing modules, in addition to the *Basic Animation* and the *Prefetch* modules, most of them modified for purposes of simplicity. Additionally, it reuses the *CSS selectors* [W3C, 2011b] pattern matching syntax in order to connect temporal specifications to spatial document fragments. Over this foundation it defines new tags: the `timesheet` tag, which represents the root of a timesheet document; and the `item` tag, which represents a simple or composite temporal specification applied to one or more HTML elements. In the following, by means of a sample Timesheets-based multimedia document, the most important characteristics of the language are discussed.

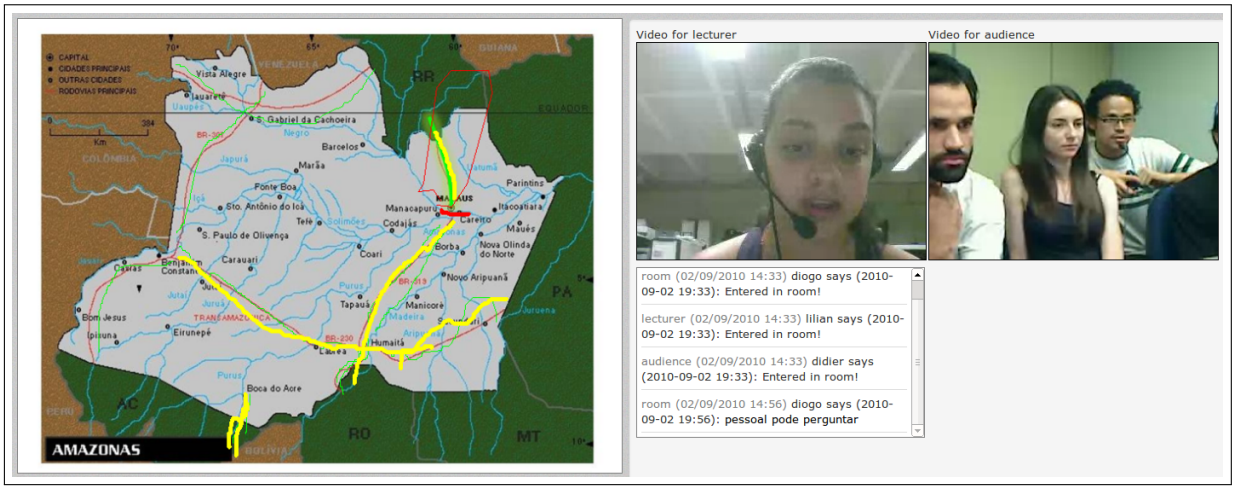


Figure 3.1: Example of a SMIL presentation for a distributed talk. Multiple media streams (audio, video, images and text) are synchronized into a session.

Figure 3.1 presents a screenshot of the example multimedia document, which was generated from a distributed talk recorded via a webconferencing environment[5]. The document aggregates all media elements captured during the talk, including a video for each participant, a whiteboard session with slides and ink annotations and a log of chat messages.

[5]DiGaE [Linhalis et al., 2010a] (Distributed Gathering Environment) is a capture environment for collaborative meetings that can be used either in instrumented rooms or in webconferencing mode. When used in instrumented rooms, DiGaE environment has support for synchronization of video cameras, microphones, electronic whiteboards and projectors; RFID readers can be used for participant identification. When used for webconferencing purposes, a special configuration called DiGaE Home provides a web-based tool to capture audio and video streams from desktops and laptops, as well as other communication tools such as instant text-based messaging and synchronous software-based whiteboard. After a capture session is finished, DiGaE exports the session into a package containing all media elements and an XML file describing all synchronization events. This export package was used as input to automatically generate the multimedia document in Figure 3.1

Listing 3.1: HTML-based spatial layout for the presentation in Figure 3.1

```
1  <html>
2  (...)
3  <head>
4   (...)
5   <link href="(...)/timesheet/1304.xml" rel="timesheet" type="application/smil+xml">
6   (...)
7  </head>
8  <body>
9  (...)
10 <div>
11  <div class="slideshow media_element">
12   <img id="image_1" src="(...)/media/session-1_slide-1.png"></img>
13   <img id="image_2" src="(...)/media/session-1_slide-2.png"></img>
14   <img id="image_3" src="(...)/media/session-1_slide-3.png"></img>
15   (...)
16   <img id="image_81" src="(...)/media/session-1_slide-81.png"></img>
17  </div>
18 </div>
19 <div id="secondary_media_container">
20  <div id="scrollable_media_list">
21   <div class="media_element">
22       Video for lecturer
23       <video id="video_82" src="(...)/media/session-1_lecturer.webm"></video>
24   </div>
25   <div class="media_element">
26       Video for audience
27       <video id="video_83" src="(...)/media/session-1_room.webm"></video>
28   </div>
29   <div class="media_element chat_window">
30       Chat messages
31       <ul>
32           <li id="chat_message_84">room (02/09/2010 14:33)  Entered in room!</li>
33           <li id="chat_message_85">lecturer (02/09/2010 14:33) Entered in room!</li>
34           (...)
35           <li id="chat_message_94">audience (02/09/2010 14:57) Left the room!</li>
36       </ul>
37   </div>
38  </div>
39 </div>
40 (...)
41 </body>
42 </html>
```

The multimedia document corresponding to this presentation is composed by a spatial document in HTML5 and a temporal document in SMIL Timesheets. The spatial document (Listing 3.1) is a regular HTML5 document which includes the external timesheet via a `link` tag (line 5) which specifies the type of document as a `timesheet` with a MIME type `application/smil+xml`. This is the recommended way of embedding a timesheet in non-XML languages, which is the case of HTML5; it also requires that the `stylesheet` relation (attribute `rel`) be supported by the user agent, if document validation is a concern. For XML-based languages, e.g. XHTML5, instead, the recommended way to include a timesheet is via a `<timesheet></timesheet>` block in the header of the document, with tags properly

scoped to the SMIL namespace. The body of the document contains fragments of declarations of media elements in the presentation: slides (lines 12-16), videos (lines 23 and 27) and chat messages (lines 32-35).

Listing 3.2: Timesheets-based temporal layout for the presentation in Figure 3.1

```
1  <timesheet xmlns="http://www.w3.org/ns/SMIL">
2    <par id="presentation">
3     <seq begin="3" id="slides">
4        <item dur="40" id="meimage_1" select="#image_1"/>
5        <item dur="40" id="meimage_2" select="#image_2"/>
6        <item dur="60" id="meimage_3" select="#image_3"/>
7        (...)
8        <item dur="1" id="meimage_81" select="#image_81"/>
9     </seq>
10    <par id="videos">
11       <item begin="24" end="1477" id="mevideo_82" select="#video_82"/>
12       <item begin="22" end="1477" id="mevideo_83" select="#video_83"/>
13    </par>
14    <par id="chat_messages">
15       <item begin="0" end="1518" id="mechat_message_84" select="#chat_message_84"/>
16       <item begin="1" end="1518" id="mechat_message_85" select="#chat_message_85"/>
17       (...)
18       <item begin="1469" end="1518" id="mechat_message_94" select="#chat_message_94"/>
19    </par>
20    (...)
21   </par>
22 </timesheet>
```

The temporal layout of the presentation is specified by the Timesheets document illustrated in Listing 3.2. Every Timesheets document starts with a `timesheet` element as root. This element, on its turn, can contain an arbitrary combination of any of the following as children: *a)* any SMIL time container, i.e. a temporal composition with associated scheduling semantics, which can be any of `par` (multiple elements executing in parallel), `seq` (multiple elements executing in sequence) and `excl` (only an element of a group executing at any single time); or *b)* `item` elements, which represents a timing specification associated to a spatial element. In Listing 3.2, the timesheet body starts with an upper `par` container, which is used as a convenience for grouping the elements into a composition named "`presentation`". This time container specifies the parallel execution of three other compositions: a `seq` container (lines 3-9), specifying the sequential execution of a group of images; a `par` container (lines 10-13), specifying the parallel execution of two videos; and another `par` container (lines 15-18), specifying the parallel execution of a list of chat messages.

Every time container in Listing 3.2 aggregates a number of *item* nodes, which is an element specific to the Timesheets language. It can contain any of the following types of attributes: *a)* attributes defined in the SMIL Timing module (e.g. `begin`, `end`, `dur`, `repeatDur`, `endsync`, etc.); or *b)* the `select` attribute, whose value is a CSS selector that associates one of more spatial elements to the timing specification. Therefore, in the slide presentation (lines

4-8) the attribute `dur` in each `item` is meant to present each slide in the specified duration, as scheduled by the sequential temporal semantics of the container. In the cases of video and chat media elements, instead, `begin` and `end` attributes are used to denote their respective activation intervals. In the particular case of the example document, every `item` element is associated with a single element in the HTML document because the respective CSS selectors operate over element identifiers. But it is also possible for a selector to match multiple elements: in this case, the effect is the same as of scheduling the results, in document order, in a `seq` container.

It is worth noting that the `item` element, besides simple timing specifications (as demonstrated in the example), can also express structural compositions. This is achieved by appending to it another element (either a time container or another `item` element) as a child. The underlying restriction is that the parent `item` represents a selection scope for the child, i.e., a child can only operate over spatial elements already selected by the father.

3.2.2 The ActiveTimesheets language

ActiveTimesheets is derived from SMIL Timesheets, since it adapts some of its original modules and extends it with additional ones. Of all modules originally included in Timesheets, only the ones related to Timing and Synchronization are reused without modifications, the remaining ones are replaced or adapted as follows:

- **Basic Animations**: this module is not included in ActiveTimesheets in favor of similar functionality in other languages. Given that ActiveTimesheets adopts HTML5 as a host language, a number of CSS3 features can be used for frame-based animations. Additionally, *Basic Animations* is part of SVG, which can be included via inline declarations in HTML5 documents: this opens up possibilities of incorporating animated content inside SVG documents, in case time-based animations is an issue. In summary, the HTML5-related stack of languages already has varied support for animations via constructs that are expressive enough to satisfy, if not all use cases, at least an ample range of them. Consequently, in ActiveTimesheets, animations are a concern delegated to spatial formatting;

- **Prefetch**: this module prescribes a declarative method to preload streams from continuous media elements prior to their activation. In ActiveTimesheets this issue is also delegated to spatial formatting, as HTML5, for instance, can perform a similar task via the `preload` attribute of continuous media elements. In case specification directly in a timesheet document is a requirement, the syntax for the `prefetch` attribute can be enforced in the spatial document by means of dynamic modification in the DOM.

In addition to modules included in Timesheets, ActiveTimesheets also integrates other modules from the full SMIL language. However, integration of some of these modules is not always seamless, given that some of them were not designed with the same requirements as those of ActiveTimesheets, such as live editing and separation between temporal and spatial

layout. Consequently, these modules are included with modifications to make them suitable to the particularities of the ActiveTimesheets language. In the following, it is discussed which modules were included and which modifications were deemed necessary:

- **Timing manipulations**: this SMIL module encompass playback control functions which can be applied to individual timed elements, via attributes related to speed, acceleration control and reverse playback. In ActiveTimesheets, this module is included because it provides an important feature when performing intra-browsing tasks in multimedia sessions, e.g. for playing a lecture in a faster speed. A limitation of this module is that timing manipulations are governed by a group of equations originally designed for static documents, i.e. it assumes that all manipulations are specified prior to document rendering. As a consequence, live editing of timing manipulation attributes is not possible via these equations. ActiveTimesheets solves this problem by adapting the original equations to piecewise functions;

- **Linking**: this module provides spatio-temporal linking functionality to SMIL. In this dissertation it is advocated that a proper integration between SMIL and HTML5 should necessarily offer full support for hypermedia links. However, the original Linking Module assumes that SMIL is being used also for spatial formatting, which is not the case of ActiveTimesheets. This problem is solved in ActiveTimesheets by re-factoring the Linking module and extending it with additional functionality, such as integration with media fragments and linking to time containers;

- **Metainformation**: this module provides constructs for attaching metadata to SMIL elements, namely via the *meta* tag, for a single property-value pair, and the *metadata* tag, for a composite metadata record. This module was included in ActiveTimesheets given that it represents a powerful resource for associating annotations directly to multimedia documents. The choice of formats and syntaxes for metadata elements is left to user agents, being the most commonly used ones those based on RDF (Resource Description Framework) semantics [W3C, 2004a] with XML syntax [W3C, 2004b]. Additionally, the decision of which of the document elements should support `metadata` elements as children is up to language designers. ActiveTimesheets includes this module in a way that metadata can be assigned to any element in the document;

- **Media clipping**: this module defines a syntax for fragmenting continuous media elements in the temporal domain. It was included in Timesheets because it has several applications regarding reuse of media elements. As a limitation, this module defines clipping only to simple media objects, but in some situations clipping over time containers, which represent a composition of media elements, is also a desirable feature. In order to solve this problem, in ActiveTimesheets the SMIL timing model is extended in order to allow clipping time containers as well.

Finally, ActiveTimesheets also includes functionality not present neither in the Timesheets language nor in any SMIL profile. These novel extensions are the following:

- **Live editing**: an ActiveTimesheets document can have any of its elements and attributes edited while the document is active, i.e. during runtime. As a consequence of an editing operation, the temporal layout of the document is automatically updated. Additionally, for performance reasons, it is important that only the affected elements are updated, instead of a full schedule computation. Moreover, some changes on spatial layout (e.g. adding or removing nodes) can also lead to changes in the temporal layout, depending on which selection patterns were employed in `item` elements in a ActiveTimesheets document. For these cases, ActiveTimesheets also provide methods to keep spatio-temporal consistency on live editing;

- **Reuse features**: differently from other SMIL languages and engines, ActiveTimesheets provides constructs for reusing SMIL elements in fine granularity. For instance, the ActiveTimesheets language allows that a document reuses whole other documents in their specification, or any addressable element inside an external document;

- **Media fragments**: closely related to reuse, the need to address fragments of a SMIL document (for instance, for linking purposes) is an important issue to make such presentations first-class citizens in the Web. ActiveTimesheets include functionalities for that purpose, which allows elements and intervals from a SMIL presentation to be URL-referenced in fine granularity;

Table 3.1 compares ActiveTimesheets with other temporal-based SMIL languages and engines. As the table demonstrates, ActiveTimesheets presents a set of extensions that potentially make it suitable for a great range of use cases. Whereas it does not support the full range of SMIL timing constructs (such as declarative sync error tolerance), it not only supports most of SMIL Timesheets modules but also extend the language in important ways. For instance, support for live editing, fine-grained reuse and media fragments are features fully provided only by ActiveTimesheets. Additionally, in some cases, such as media clipping, basic linking, and time manipulations, the original functionality is extended to make it more conforming to novel extensions or the HTML5 environment. The remaining of this chapter discuss these extensions and novel features.

3.3 The ActiveTimesheets timing model

In order to render, or format, a SMIL document temporally, i.e. to compute its temporal layout, it is important to have an underlying formal representation that allows to describe the semantics of its temporal model. Even though ad-hoc scheduling strategies, such as the hashtable-based mechanism proposed by Gaggi and Danese [2011], may suffice a number of formatting

Table 3.1: Comparison between ActiveTimesheets and other SMIL-based temporal languages and engines.

Module/features	SMIL	XHTML+SMIL	SMIL Timesheets	Timesheets.js	ActiveTimesheets
Media					
MediaClipping	+	+	-	-	+ +
Timing					
TimingAttr	+	+	+	+	+
RepeatTiming	+	+	+	+	+
EventTiming	+	+	+	+	+
SyncBaseTiming	+	+	+	+	+
SyncBehavior	+	-	-	- +	-
TimeContainerAttr	+	+	+	+	+
BasicTimeCont	+	+	+	+	+
BasicExclTimeCont	+	+	+	+	+
ContentControl					
PrefetchControl	+	-	-	-	+
Linking					
FragmentIdentifier	+	-	-	+	+ +
LinkingAttributes	+	-	-	-	+
BasicLinking	+	-	-	-	+
Metainformation					
Metadata	+	-	-	-	+
Time Manipulations					
Speed	+	+	-	-	+ +
Novel extensions					
Live editing	-	-	-	- +	+
Reuse extended	-	-	-	-	+
Media fragments	-	-	-	-	+

+ supported - unsupported -+ partially supported ++ supported with extensions

needs, some document manipulations may demand more sophisticated models. One of these manipulations is live editing, which requires that a document modification be translated to the underlying rendering model in a simple and efficient manner. To that end, it is important that the rendering model can be properly mapped to the document model.

The semantics of the SMIL Timing model has been described via various formalisms, for instance, automata [Bertolotti and Gaggi, 2007], timed petri nets [Bouyakoub and Belkhir, 2011], logic rules [Bossi and Gaggi, 2007], just to name a few. Most of these formalisms have been employed for verification purposes, i.e. to detect, generally via simulation, whether the temporal layout of a SMIL document contains any inconsistency. Some of them, for instance the model proposed by Sampaio and Courtiat [2000], also allows the generation of a schedule for the presentation and, therefore, can be applied not only on verification but also on rendering. Despite their advantages, a downside of using these models is that their semantics, from an authoring perspective, are very dissimilar to the document language. This can be a complicating factor for live editing operations, because it may be hard to map a modification in the SMIL document to a corresponding modification in the rendering model.

The SMIL recommendation, on the other hand, promotes a graph-based model, called *timegraph*, to render a document. A timegraph is very similar to the hierarchic syntax of its

SMIL document and, as a consequence, it simplifies the mapping of constructs in the document to the corresponding nodes in the timegraph. Provided these advantages, the timegraph is the model of choice for building the editing operations described in this chapter.

A complicating factor to grasp the SMIL timing model is that its recommendation [W3C, 2008g] does not include any formal model which allows a concise and accurate definition of its temporal semantics. Instead, these issues are described mostly via natural language descriptions and, in rare occasions, pseudo-code. Documentation of the timegraph model is no exception, since it is only mentioned, out of context, in a few sections. In face of these issues, this section attempts to provide a more structured overview of the timegraph model. The basic concepts of the timegraph model has been abstracted not only from the SMIL's literature but also from souce code analysis of several SMIL implementations such as Ambulant and Timesheets.js. Based on these sources, a group of basic principles for the timegraph model was derived and used as a basis to formalize the model reported here. Thus, it is important to stress that the ActiveTimesheets timegraph model has many differences to the timegraph model of related engines, as it has been highly influenced by design decisions such as live editing and reuse features.

3.3.1 The timegraph model

The timegraph model is an object-oriented graph-based data structure which is used to control the temporal rendering of a ActiveTimesheets document. Nodes in the timegraph represent elements whereas edges represent temporal relationships between elements. Formally, a timegraph is defined in Equation 3.1, where:

$$TG = (N, R) \tag{3.1}$$

- N is a set of nodes representing time elements;
- R is a set of edges representing temporal relationships.

A time element is specialized to various types, depending on the temporal semantics of the language construct it abstracts. Figure 3.2 presents a hierarchy of the possible time element types that may be present in a timegraph.

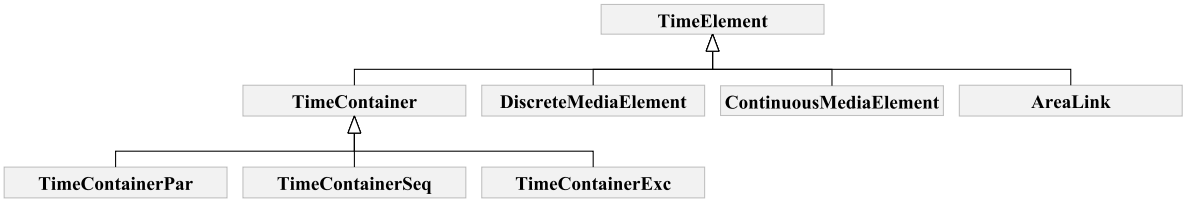


Figure 3.2: Class diagram of time element types.

The most abstract type is *TimeElement*, which represents an upper element which has an internal timeline. From this element can descend *TimeContainer*, which represents a temporal composition, media objects as *DiscreteMediaElement* and *ContinuousMediaElement*, and *AreaLink*, which represents a link (which is SMIL also has a temporal scope, as discussed later in this chapter).

Each timegraph node n is represented by the tuple given in Equation 3.2, where:

$$n = (I, E, d_i, t, s) \tag{3.2}$$

- $I = [b_I, e_I)$ is the internal timeline, where b_I and e_I are time points;
- $E = [b_E, e_E)$ is the external timeline, or active interval (ai), where b_E and e_E are time points;
- d_i is the element implicit duration;
- t is the current element time;
- s represents the element state;

A timegraph node has two temporal scopes, each one represented by a separate timeline. The internal timeline is used for scheduling the children of the node, i.e., it is a temporal reference for temporal aligning every child. The external timeline, on the other hand, defines the active interval of the element, which is used in the schedule of its parent. The local timeline and the external timeline of an element are potentially different, due to offsets applied by the `begin` attribute and duration restrictions imposed by the `end` and `dur` attributes, for instance. Each timeline is represented by an right-open interval, meaning that the element duration finishes as soon as the upper bound of its active interval is reached. In order to derive the current time of the element, t, a time function, F, is used to make conversions between the external timeline and the internal timeline or, in other words, to hierarchically convert the time of the reference clock to the time of a specific element. The element implicit duration, d_i, is the natural duration of the element, such as the intrinsic duration of a video. Discrete media elements, on the other hand, do not have a natural duration, consequently their implicit duration is zero. The implicit duration of time containers is dependent on their children and container-specific semantics, as will be discussed later.

Figure 3.3 presents a graphical representation of the timegraph corresponding to the timesheet previously introduced in Listing 3.2. In this example, a timegraph node was created for every time container node (in the Figure, represented by the codes *par* and *seq*). Additionally, every `item` node also generated a single timegraph node, which is either a discrete media element (*dm*) or a continuous media element (*cm*), depending on type of media detected by parsing the element. Each node has an internal id (notation `<type>:<id>`) and stores its own scheduling information. In the example, such information encompasses, for each node: *i)* an active interval

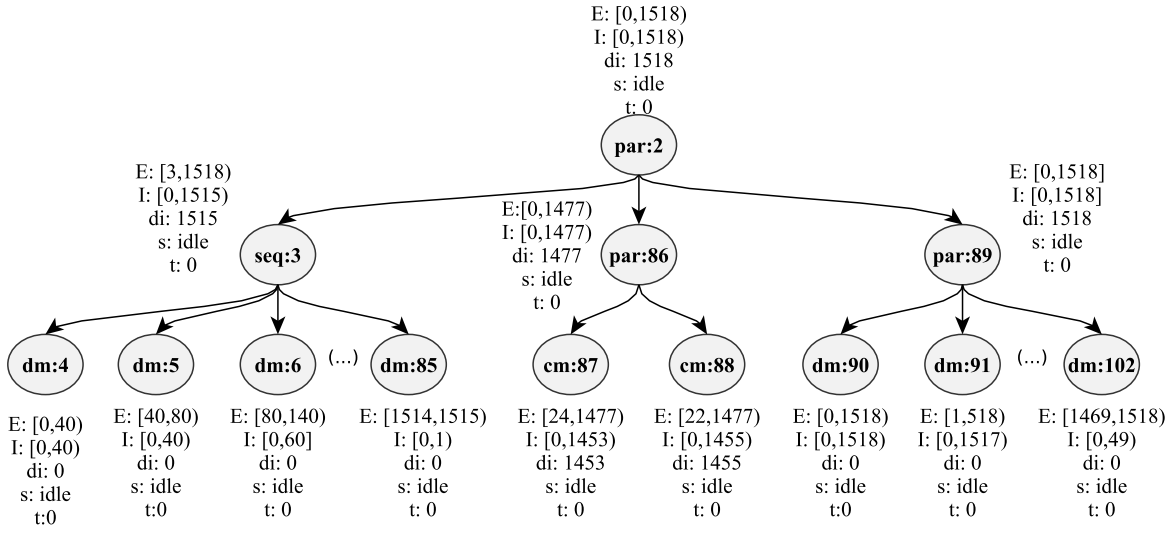


Figure 3.3: Timegraph corresponding to Timesheets document in Listing 3.2

(*E*), which represents the element external timeline (in the example, given in seconds); *ii)* an internal timeline (*I*); *iii)* the implicit duration of the element (*di*); *iv)* a state (*s*), which tracks the lifetime of the element; and *v)* the current time of element (*t*), a time point relative to the element active interval. In the example, the active intervals for all elements are already computed (given that all intervals are resolved), and the document has not been started yet or has been reseted (given that all elements are in *idle* state).

Given that the timegraph realizes a hierarchical temporal model, the active interval of each child node is relative to the internal timeline of its parent. As a consequence, the active interval of node `dm:5`, for instance, starts $40s$ after the start of the active interval of node `seq:3` which, by its turn, starts $3s$ after the start of its parent, `par:2`. By traversing the graph from `dm:4` up to `par:2`, computing the local offsets while doing so, it is possible to derive that `dm:5` starts $43s$ after the presentation starts. The algorithm for converting local time to global time will be presented further in this chapter.

The states a time element can assume are represented in Figure 3.4. An element initially enters an *idle* state after its timing has been computed for the first time. From the *idle* state, an element can transition to a *playing* state only if it is activated, which can occur via a composition event, a internal event or an external event. External events (e.g. buffer underrun) can trigger a transition to a *stalled* state, whereas an internal event (e.g. end of the intrinsic duration) can trigger a transition to the *frozen* state (which, for a video, corresponds to pausing it in the last frame). A transition to *idle* state can be triggered from any other state as soon as the element receives a *deactivate* event.

In many respects, a timegraph is structurally similar to the document DOM, since in many cases it is possible to build a correspondence between DOM nodes and timegraph nodes. Such correspondence is recurrent, in particular, when `item` elements consistently select a single

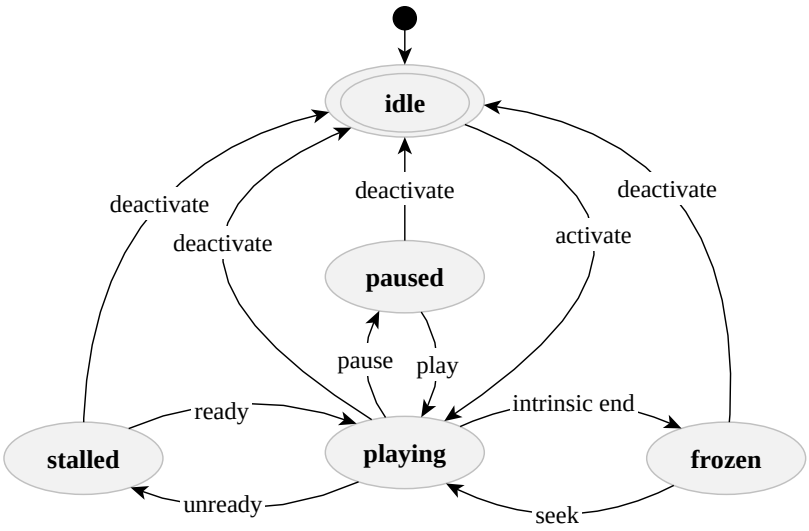


Figure 3.4: State machine for a time element.

element from a spatial document. This is the case, for instance, in the example presented in Figure 3.3. At first sight, this correspondence might suggest that building a timegraph consists only on augmenting the DOM model of the SMIL document with scheduling information. But for many other configurations there is no such one-to-one correspondence between DOM nodes and timegraph nodes. This occurs mostly in the cases in which: *i)* the item element matches multiple elements; or *ii)* the item element is used as structural composition.

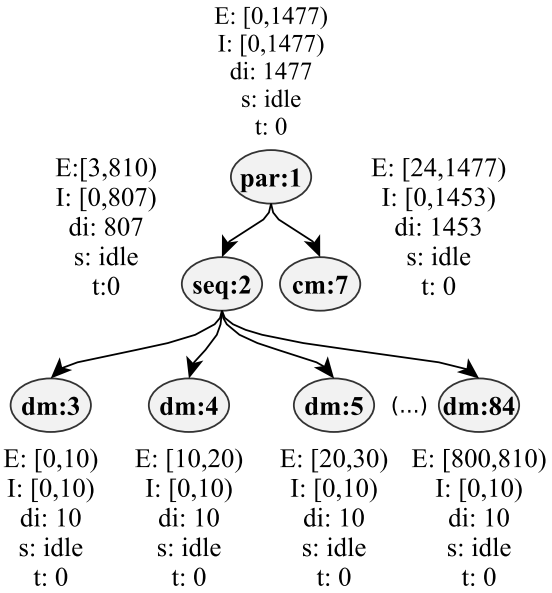
As an example of this pattern, consider the document fragment in Figure 3.5a. In this fragment, still applying to the spatial document in Listing 3.1, an *item* element (line 2) is selecting all images in the document (via the selector `img`, which will match all images). According to the semantics of the item element, when multiple elements are matched by a selector, then these elements are scheduled as a `seq` container. Additionally, the timing attributes of the `item` element are distributed among the children, which in the example means that each children will be active for 10 seconds. In summary, in this example the timegraph has more nodes than the timesheet DOM, consequently augmenting the DOM with timing information is not a robust alternative to a timegraph.

Similarly to nodes, timegraph edges can also assume different types, thus representing a variety of temporal relationships (Figure 3.6). In the examples introduced so far, only parent-child (composition) relationships have been demonstrated, which makes the resulting timegraphs be represented as trees. More elaborate structures can be formed depending on the types of relationships that are expressed in the document, of which the following are the possible ones:

- *composition relationship*: this is a relationship between a time container and another time element. The semantics of this relationship implies that the temporal scope of the child is constrained to the timeline of the father, i.e. the relationship forms a hierarchy of timelines;

```
<par>
   <item begin="3" select="img" dur="10" />
   <item begin="24" end="1477"
      select="#video_82"/>
</par>
```

(a) Timesheet



(b) Timegraph

Figure 3.5: Fragments of document and corresponding timegraph with an `item` element selecting multiple spatial elements.

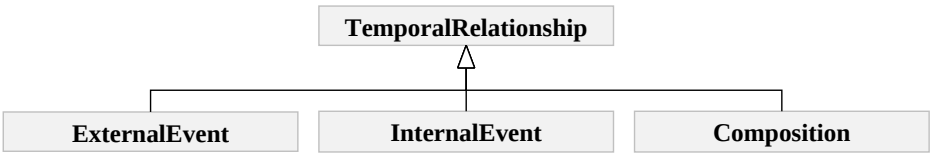


Figure 3.6: Hierarchy of temporal relationships.

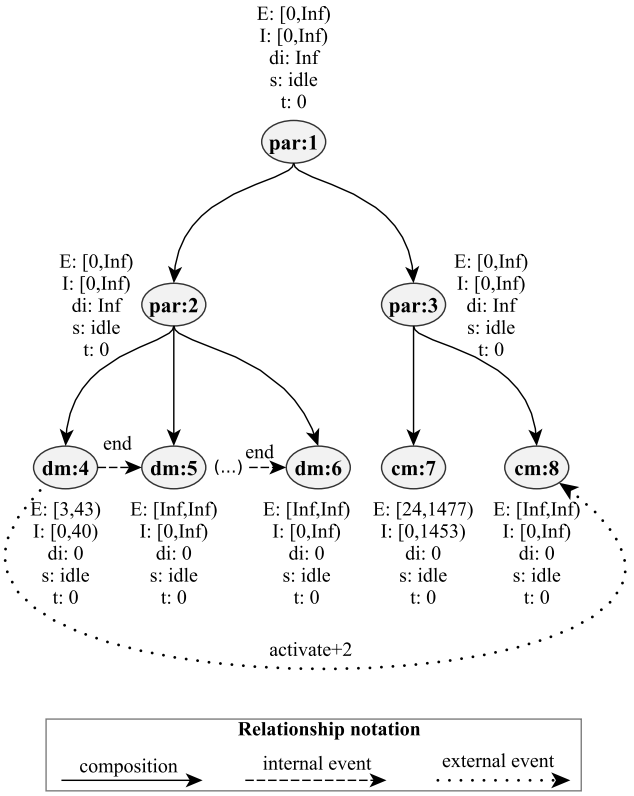
- *internal event relationship*: also called syncbase relationship, this is a event-based synchronization relationship between time elements. The semantics of this relationship establishes a time element can be activated or deactivated upon events triggered by other time elements (e.g. an element begins when other finishes);

- *external event relationship*: this is a synchronization relationship between a time element and the external document environment. Examples of external events are user interaction (e.g. via DOM events or via links) and environment-related events (e.g. abort or suspend the presentation due to network-related issues).

Composition relationships characterize the hierarchic nature of the SMIL timing model and take full advantage of its per-element timeline model. Internal and external events, by their turn, contribute with constraint-based synchronization. All of them, combined, make SMIL a hybrid temporal model. Figure 3.7 demonstrates the influence of internal and external events in the timing of a document. In Figure 3.7a the time container `slides` represents a slide show with

```
1  <par>
2   <par id="slides">
3    <item begin="3" dur="40" id="meimage_1"
4     select="#image_1"/>
5    <item begin="meimage_1.end" dur="40"
6     id="meimage_2" select="#image_2"/>
7    (...)
8    <item begin="meimage_80.end" dur="1"
9     id="meimage_81" select="#image_81"/>
10  </par>
11  <par id="videos">
12   <item begin="24" end="1477"
13    id="mevideo_82" select="#video_82"/>
14   <item begin="meimage_1.activate + 2"
15    end="1477" id="mevideo_83"
16    select="#video_83"/>
17  </par>
18 </par>
```

(a) Timesheet



(b) Timegraph

Figure 3.7: Fragments of document (and corresponding timegraph) containing composition relationships, internal events and external events.

automatic transitions, i.e. each slide from the second on is activated as soon as its predecessor finishes. This is achieved by specifying, as the value of each element's `begin` attribute, a syncbase event corresponding to the end of the predecessor element (e.g. line 5). The same pattern is applied to all slides in the sequence. This is reflected in the timegraph by means of an internal event relationship (dashed edge between nodes *dm:4* and *dm:5*, for instance). An example of external event relationship results from interpretation of the time container `videos`: the element in line 14 has its begin attribute specifying that it should be activated two seconds after the element `meimage_1` triggers its *activateEvent* (which occurs, for instance, when a user clicks over the element). This is reflected in the timegraph via a external event relationship (dotted edge between nodes *dm:4* and *cm:8*).

Internal and external events influence the scheduling of the timegraph. In fact, the mechanism of interaction between events and scheduling accomplishes the integration between timeline-based and constraint-based synchronization in SMIL. An example of this integration is provided in Figure 3.7b: the active interval for element *dm:5* is unresolved, i.e. the interval have in both extremities an *Inf* value, which stands for Infinity. This is because it is not possible to determine the active interval of the element until the internal event, specified in its begin attribute, occurs. Such indeterminate timing is propagated to all ascendant nodes via composition relationships, up

to the root[6]: as a result, all nodes, including the root node, have indeterminate end in their active intervals. The same scheduling pattern occurs in node *cm:8* and its ascendants. Each type of relationship is represented in the timegraph with a different notation: composition relationships with a solid line, internal event relationships with a dashed line and external event relationships with a dotted line. This is the notation that will be used thorough the dissertation to graphically represent temporal relationships. When one of the event-based relationships in the timegraph occurs, the schedule of the affected nodes should be recomputed in order to resolve their active intervals (this process, called resolution, will be discussed further in this Chapter).

3.3.2 Timegraph processes and their orchestration

The lifecycle of a timegraph involves the following processes: *i) construction*, in which the timegraph is built from the document syntax; *ii) scheduling*, which consists on computing the active interval of each element in the timegraph; *iii) sampling*, which consists on updating the internal time of each element as long as the timegraph is active; *iv) resolution*, which consists on updating the timegraph schedule when some timing constraint is solved. Figure 3.8 summarizes how these processes are orchestrated in the ActiveTimesheets engine.

The integration of all processes is abstracted as a state machine. The engine state machine starts with a full scheduling of the timegraph which, when ready, puts it in the *idle* state. After that, the engine can be started, which will sample the timegraph continuously. Upon any resolution event, the associated constraint is solved and then the schedule is partially updated. After that, the engine can resume the sampling of the timegraph. Sampling can be halted by a pause operation and terminated by a stop operation. In the following Sections, the most important issues of each of these processes are discussed in more details.

3.3.3 Timegraph construction and scheduling

In order that temporal rendering can occur, the timegraph has to be built via the process described in Figure 3.9. In ActiveTimesheets, rendering of temporal layout occurs as soon as the spatial layout finishes. At this point, all spatial nodes are potentially resolved, so timesheet documents are fetched and then parsed via recursive descent over the timesheet DOM. The output of parsing a timesheet is a timegraph. In case more than one timesheet is defined for the document, all of their corresponding timegraphs are combined into single timegraph, using a implicit `par` container to achieve the integration. In order to track the origin of each timegraph in the implicit container, a reference to the original DOM is maintained for each subgraph in the implicit container.

[6]A timegraph that contains only composition relationships is in fact a tree: in this case, the root of the timegraph is the only node that has no composition ancestor. As the other types of relationhips, i.e. internal and external events, are not used for traversal purposes (but only for resolution and schedule update purposes), hereafter “root node” denotes the root of the subgraph formed by composition relationships alone.

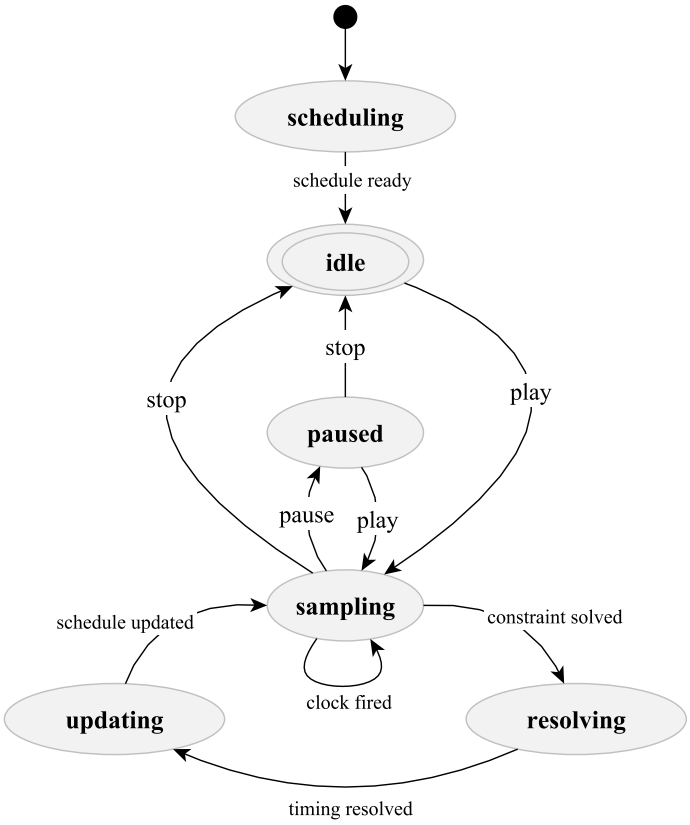


Figure 3.8: ActiveTimesheets engine state machine.

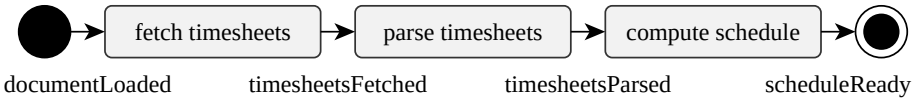


Figure 3.9: Activity diagram for timegraph construction.

In its initial state, the timegraph has no schedule, i.e. the intervals of all its nodes are unresolved and they are initialized to an idle state. The phase of schedule computation proceeds in depth-first traversal of the timegraph and computes the active intervals of each node based on the semantics of their timing attributes and temporal relationships. Such computation involves an elaborate group of rules with exceptions and special cases, of which a simplified[7] variant is summarized in Algorithm 3.1.

Algorithm 3.1: Simplified timegraph node full scheduling

[7]For simplicity purposes, only the fundamental timing attributes (`begin`, `end` and `dur`) have been regarded in this Definition. For specialized timing attributes, such as `endsync`, `repeatDur`, `min`, `max`, etc., their default values have been assumed. For more information on the temporal effects of these specialized attributes refer to the SMIL timing and synchronization specification [W3C, 2008g].

1: Let $this$ be the timegraph node
2: **function** TIMEELEMENT.COMPUTESCHEDULE( )
3: Let B, E, and D be the `begin`, `end` and `dur` attributes of $element$, respectively
4: Let $this.i_a$ be the element active interval
5: $this.d_i \leftarrow$COMPUTEFULLIMPLICITDURATION( )
6: $b \leftarrow$ COMPUTEBEGIN(B)
7: $d \leftarrow$ COMPUTEDURATION(D, E, b)
8: $this.i_a \leftarrow [b, b + d)$
9: **end function**

For clarity purposes, Algorithm 3.1 has been organized in an object-oriented structure, taking advantage of functions which represent methods of *TimeElement* objects, each of these functions encompassing a separate component in schedule computation. The most important and global method, `computeElementSchedule`, determines the timing of an element, i.e. the begin moment and duration: with this information the active interval of the element can be determined. Specific schedule rules are factored in different functions: `computeBegin` to process `begin` attribute semantics, `computeDuration` to process `dur` attribute semantics, and `computeFullImplicitDuration` to determine the implicit duration of the element according to its type-specific temporal semantics. Once the basic timing units are computed, the algorithm can determine the active interval of the element, which is its main result. In case indeterminate timing attributes are present, to the element is assigned an infinite explicit duration, which can result either in a left-bounded active interval (e.g. an element that may never finish), provided that the begin attribute has determinate timing, or a unbounded active interval (e.g. an element that may never begin), provided that the element begin is indeterminate. Naturally, the active interval boundaries are defined as soon as the relevant attributes are resolved, as it will be discussed further in the Chapter. In the following, the main functions called in this algorithm are presented in more detail.

Algorithm 3.2: Computation of element begin offset

1: **function** TIMEELEMENT.COMPUTEBEGIN(B)
2: Let b be the computed begin
3: **if** B is undefined **then**
4: $b \leftarrow 0$
5: **else if** B is constraint-based and unresolved **then**
6: $b \leftarrow \infty$
7: **else if** B is constraint-based and resolved **then**
8: $b \leftarrow t$, where t is the constraint solution
9: **else if** B is timeline-based **then**
10: $b \leftarrow B$

```
11:     end if
12:     return b
13: end function
```

The computation of the begin offset of an element is demonstrated in Algorithm 3.2. The basic idea of the algorithm is to determine the element offset by checking if the begin attribute is defined, resolved or unresolved. Depending on which case has been selected, an appropriate value is chosen as begin. Such value can be resolved or unresolved. Resolved values are normal time units (e.g. an explicit begin offset or a solved constraint) whereas unresolved values are indicated by a special value, $Infinity$ or ∞. A consequence of an unresolved begin offset is that the active interval of the element will be unbounded until the associated constraint is solved.

Algorithm 3.3: Computation of element full implicit duration

```
1: Let this be the timegraph node

2: function TIMEELEMENT.COMPUTEFULLIMPLICITDURATION( )
3:     Let d_i be the implicit duration
4:     if this is a continuous media element then
5:         d_i ← dur, where dur is the intrinsic duration of the node
6:     else if this is a discrete media element then
7:         d_i ← 0
8:     else if this is a time container then
9:         Let ctype(this) be the time container type
10:        for each child node as c do
11:            c.COMPUTESCHEDULE( )          ▷ a recursive schedule computation in the child
12:        end for
13:        this.LAYOUT( ) ▷ Temporally layout the children according to container-specific semantics
14:        if ctype(this) is par or seq then
15:            d_i ← e_max, where e_max is the maximal interval bound[8]
16:        else if ctype(this) is excl then
17:            if a child has been activated then
18:                d_i ← e_cur, where e_cur is the upper bound of the active child interval
19:            else
20:                d_i ← 0
21:            end if
22:        end if
23:    end if
```

[8]The maximal interval bound of a time container is the upper interval bound of the last active children in the container schedule, i.e. the maximum upper bound among all children active intervals

24: **return** d_i
25: **end function**

The full implicit duration of an element is a precondition to the computation of the element actual duration. This is a very important step in the algorithm, given that in this step occurs the recursive call that will make the scheduling process propagate to the whole timegraph. As demonstrated in Algorithm 3.3, the element full implicit duration is assigned according to the types of the element, e.g. discrete media element, continuous media elements or time container. Time containers represent a more elaborate case because, besides inherent differences in temporal semantics between different types of containers, their implicit duration depend on the implicit duration of their children. But, for a child active duration to be available, its schedule must be computed as well. Then, for every time container a recursive call to `computeSchedule` needs to be issued in order to obtain this information.

Algorithm 3.4: Computation of element active duration

1: Let $this$ be the timegraph node
2: **function** TIMEELEMENT.COMPUTEDURATION(D, E, b)
3: Let d be the computed duration
4: **if** E is constraint-based and unresolved **then**
5: $d \leftarrow \infty$
6: **else if** E is constraint-based and resolved **then**
7: $d \leftarrow t - b$, where t is the constraint solution
8: **else if** D is timeline-based **then**
9: $d \leftarrow D$
10: **else if** D and E are undefined **then**
11: $d \leftarrow this.d_i$
12: **end if**
13: **return** d
14: **end function**

Once the implicit duration is available, the element actual duration can be computed, as demonstrated in Algorithm 3.4. The algorithm combines the semantics of `dur` and `end` attributes, having the element implicit duration as a fallback in case none of these attributes have been defined. Similarly to begin values, duration values can also be either unresolved or resolved, depending if the involved attributes are constraint-based or not. When a duration element is unresolved, the element active interval is at least right-unbounded.

Once timing has been computed, the timegraph is unlocked and then can be activated in the presentation. In order to generate a schedule, necessarily all timegraph nodes will be traversed,

so the complexity of the operation is always the worst case (recall Equation 3.1 for notation): i.e. $O(|N|)$ in time and $O(|C|)$ in space, where $C \subset R$ is the set of composition relationships. In ActiveTimesheets, full timing computation should occur only at timegraph initialization. Future schedule updates, such as those derived from event resolution and live editing, should lead to partial or incremental updates in the schedule, respectively. Such update procedures will be discussed in the next Sections.

3.3.4 Timegraph sampling

As long as a timegraph is active, i.e. as long as its root node is active, its timing needs to be periodically updated. This process, called *sampling*, consists on performing a depth-first traversal of the timegraph whereas assigning a local time, or current time, to each visited node. The result of the sampling process is a *sample*, that is, a specific timing state[9] of the timegraph. This process periodically generates a sample (e.g. at every 40 ms, or 24 frames per second) which is triggered by a global reference clock, based on system time. At every update, the current clock time is converted to the local timeline, or local temporal scope, of each element. The worst case of the sampling process occurs when all nodes in the timegraph are active, therefore its space and time complexities are similar to that of scheduling. However, differently from scheduling, the sampling process occurs only over active nodes, i.e., it does not descend into inactive children, thus whole deactivated branches are pruned from the traversal. Consequently, in certain conditions, which are highly dependent on the temporal semantics of the presentation, the sampling process can be optimized.

For each visited non-root node, the local time t is given by the function in Equation 3.3, where[10]:

$$t = T - b - O \tag{3.3}$$

- t is the local time for the element;
- T is the parent simple time, i.e., the current time of the immediate parent;
- b is the element begin time;
- O is the synchronization offset, i.e., a perceived synchronization error between the timelines of the element and its parent; thus, it discounts environmental issues, such as network-related delays, that might lead two related nodes to slip.

An exception for this Equation is the root node since, besides having no parent, its internal and external timelines coincide. Then, the local time of the root node is the same as that of the

[9] A sample of a timegraph, also called timing state, represent instances of the attributes of all of its nodes in a given time. In other words, a sample is a set of structures in the form given by Equation 3.2.

[10] Here, for simplicity, it is assumed the use of the fundamental timing attributes. In case of `repeat*` attributes, which may influence the timing computation, their default values are assumed.

reference clock. For other elements, upon updating the current time, sampling also implies a verification on the current execution state of each composite node. This means that, based on the new local time, children of a time container may be activated or deactivated, depending on their schedule. This verification occurs before the traversal descend into the children, consequently the traversal might have branches pruned or included at this point.

Figure 3.10 demonstrates, in 5 clock steps, the sampling process of the example timegraph previously introduced in Figure 3.5b. In the example, branches that were visited during the current sample have their nodes emphasized in a darker background shade. In this timegraph, when $t = 1$, only the root node is visited and, due to conformance of its scheduling, activated. In a further moment, when $t = 3$, additional branches, i.e. *seq:2* and *dm:3*, are visited and activated, consequently their states are changed to *playing*. Notice that the element time (t) is relative to each node internal timeline (I); as a consequence, when `seq:2` is activated, it has $t = 0$. Further updates will keep adapting the traversal to other elements of the `seq` container (`dm:4`, `dm:5`, and so on) as soon their active intervals are reached by their parent time. Meanwhile, at $t = 24$, the continuous media element *cm:7* is also activated. At *t=900*, *seq:2* and their children have already reached their durations, so as the only active elements are the root and the continuous media element. This timegraph sample persists until the duration of the timegraph is reached at $t = 1477$, when the whole timegraph is then deactivated (i.e. the upper bound of the root element is reached).

3.3.5 Timegraph resolution

The *resolution* process, which occurs asynchronously to the sampling process, updates the scheduling of a timegraph based on internal and external events. As demonstrated in the timegraph model, nodes whose synchronization are constraint-based have unresolved timing, the resolution occurring upon event triggering. Such resolution follows the basic scheduling rules previously presented in Algorithm 3.1. However, differently from scheduling, the resolution process implies a partial scheduling of the timegraph, i.e. a scheduling only over the potential nodes affected by the resolution, as demonstrated in Algorithm 3.5.

Algorithm 3.5: Simplified timegraph node partial scheduling

1: Let $this$ be the timegraph node

2: **function** TIMEELEMENT.COMPUTEPARTIALSCHEDULE( )
3: Let B, E, and D be the `begin`, `end` and `dur` attributes of $element$, respectively
4: Let $this.i_a$ be the element active interval
5: $this.d_i \leftarrow$COMPUTEPARTIALIMPLICITDURATION( )
6: $b \leftarrow$ COMPUTEBEGIN(B)
7: $d \leftarrow$ COMPUTEDURATION(D, E, b)

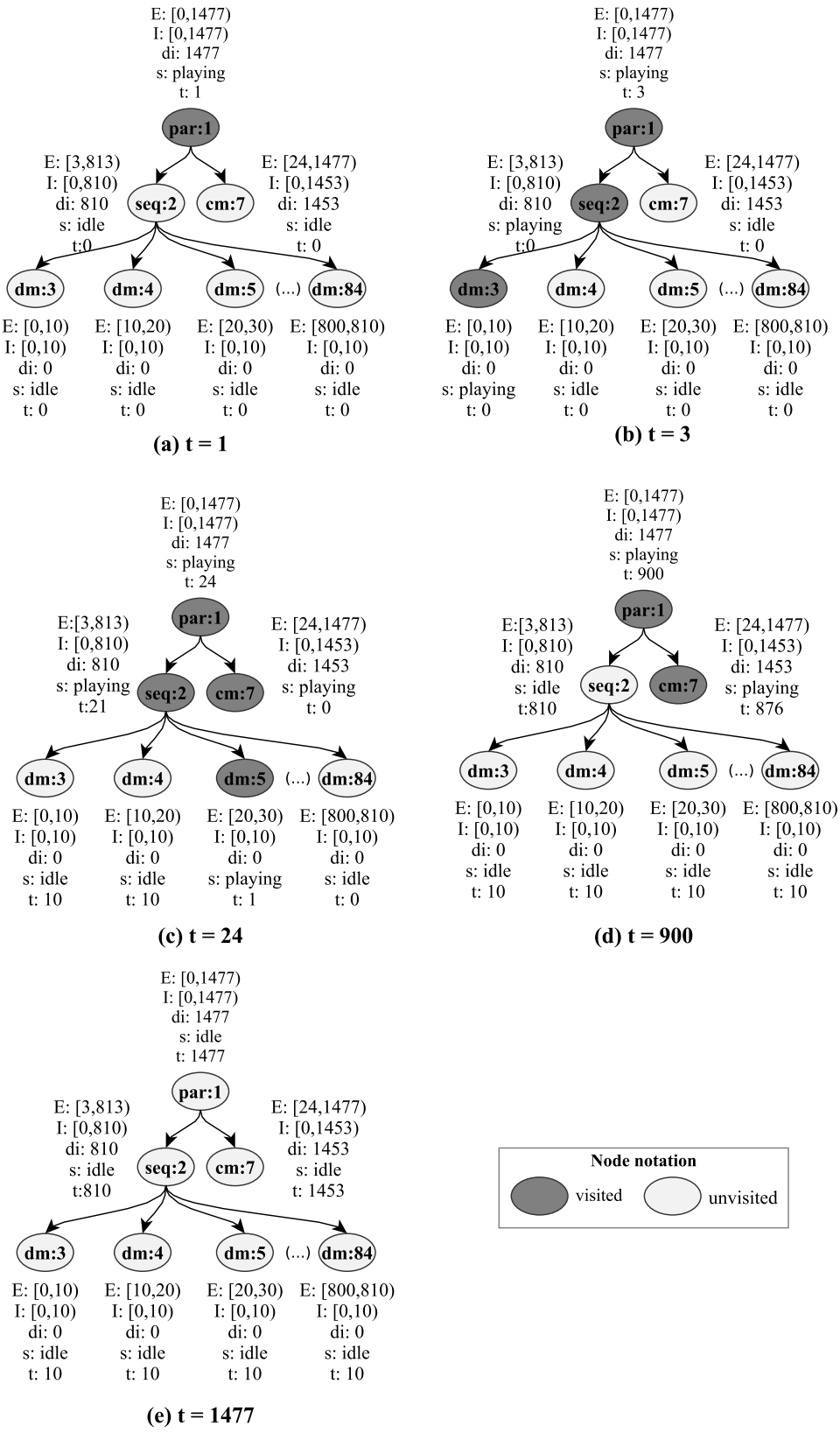


Figure 3.10: A few sampling states from a timegraph.

```
8:      this.i_a ← [b, b + d)
9:      if this.parent is not the root node then
10:         this.parent.COMPUTEPARTIALSCHEDULE( )
11:     end if
12: end function
```

In fact, the partial scheduling algorithm is an extension of the full variant. One important modification is the definition of the function `computePartialSchedule` which, besides computing the scheduling of the node, also recursively propagates the resolution to ancestor nodes in the timegraph (lines 9-10). Another important modification is the definition of the call to the function `computePartialImplicitDuration` (line 5), defined in Algorithm 3.6.

Algorithm 3.6: Partial computation of element implicit duration

```
1: Let this be the timegraph node

2: function TIMEELEMENT.COMPUTEPARTIALIMPLICITDURATION( )
3:     Let d_i be the implicit duration
4:     if this is a continuous media element then
5:         d_i ← dur, where dur is the intrinsic duration of the node
6:     else if this is a discrete media element then
7:         d_i ← 0
8:     else if this is a time container then
9:         Let ctype(this) be the time container type
10:        this.LAYOUT( ) ▷ Temporally layout the children according to container-specific semantics
11:        if ctype(this) is par or seq then
12:            d_i ← e_max, where e_max is the maximal interval bound[11]
13:        else if ctype(this) is excl then
14:            if a child has been activated then
15:                d_i ← e_cur, where e_cur is the upper bound of the active child interval
16:            else
17:                d_i ← 0
18:            end if
19:        end if
20:    end if
21:    return d_i
22: end function
```

[11] The maximal interval bound of a time container is the upper interval bound of the last child in the container schedule, i.e. the maximum upper bound among all children active intervals

This modified algorithm for implicit duration replaces the full version and eliminates recursive schedule computation in the children. As a result, the only elements that will have their schedule recomputed are the current node and all its ancestors. Therefore, the complexity of a partial scheduling, started by the resolution of a specific node n, is dependent on the depth of the node, i.e. $O(depth(n))$. The worst case for partial update occurs when the deepest node of a degenerate timegraph is resolved. A degenerate timegraph is a list, i.e. a succession of singleton compositions. In this case, $depth(n) = |N|$ and, consequently, the complexity is the same as of full scheduling.

As an example of timegraph resolution, Figure 3.11 revisits Figure 3.7b by sampling selected key states of its execution. In the Figure, besides emphasis on active branches, timing metadata changes in response to events are also emphasized, in boldface. Notice that some elements have indeterminate timing: this is the case of node `dm:5`, whose begin depends on a internal event triggered by `dm:4`. In the cases in which the timing of an element is resolved by an internal or external event, the external timeline of the element is unresolved until the event happens: this is represented, in the timegraph, by an *Inf* (standing for "Infinity") value in the interval, which can occur either in the lower or upper bound of the interval, depending on the timing specification. Notice that the internal timeline of an unresolved element, on the other hand, can be resolved, if the element has a resolved explicit duration (this is the case for all unresolved elements in the example).

At sample *(a)*, when $t = 30$, the timegraph has activated the nodes scheduled up to this point and no event-based synchronization has been captured so far. At $t = 35$, a user interaction event is simulated over *dm:4*, as a consequence the timing of node *cm:8* is resolved: its active interval gets resolved to the specified constraint ($activateEvent + 2 = 35 + 2 = 37s$). Additionally, the resolution propagates to ancestor nodes, which also resolves the active interval of node *par:3*. However, the last ancestor, the root node, is not resolved yet, because it still have an unresolved child, i.e., *par:2*. At $t = 45$, an internal event from *dm:4* has been already triggered, thus the resolution of dependent node *dm:5* has already occurred; however, such resolution does not propagate to any ancestor, as all of them until have unresolved children. At $t = 1394$, node *dm:6* is resolved, because of an internal event triggered by its immediate sibling. As a result, the resolution propagates to the immediate parent, node *par:2*, and to the root node that at this point has all its children resolved. The root active interval, however, has its upper bound resolved to the duration of the right branch, which is longest than that of the left branch. At this point, the whole timegraph is resolved and no scheduling updates will occur as long as the reference clock keeps moving forward: the timegraph will follow the present schedule until its duration is reached.

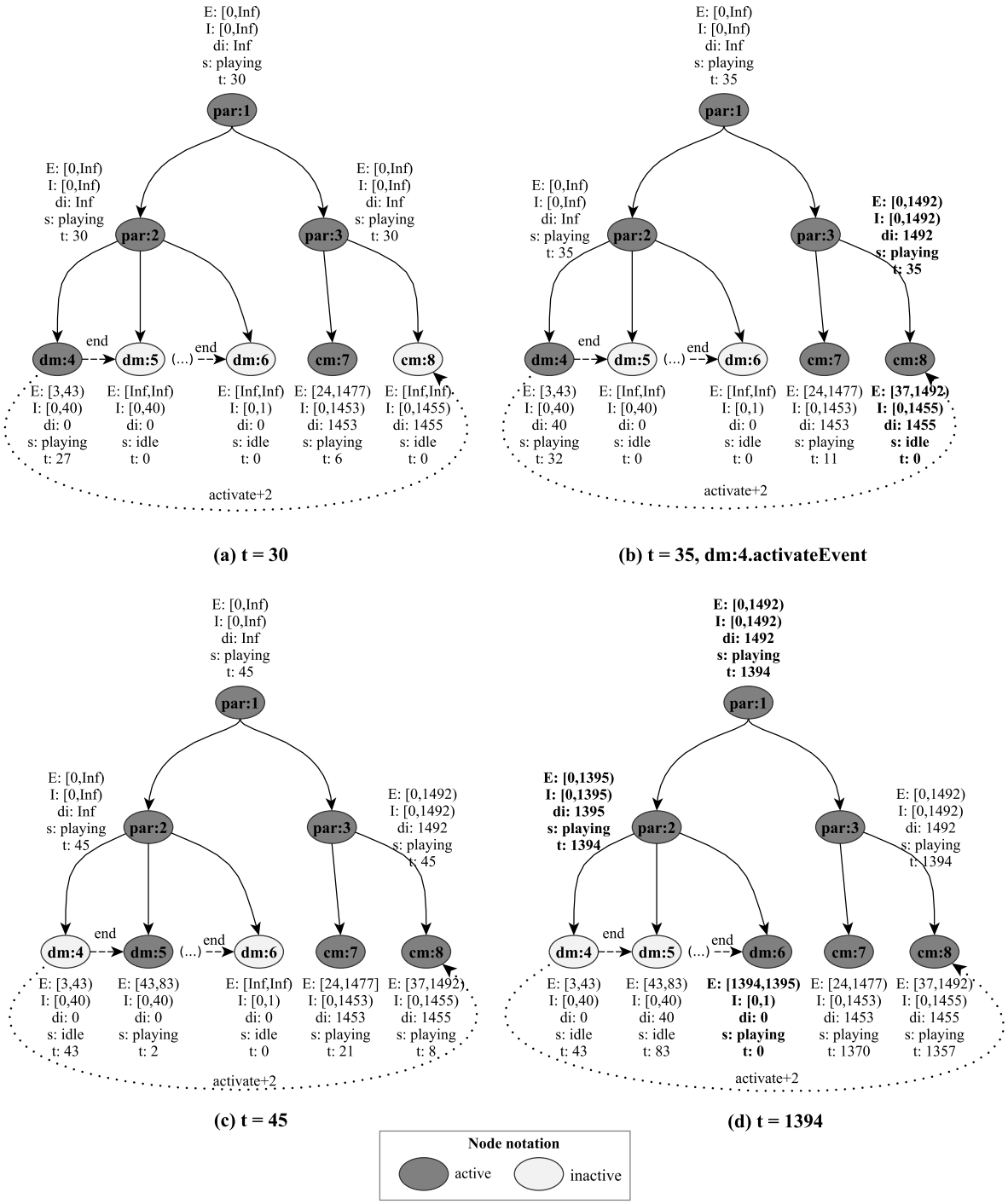


Figure 3.11: A few resolution states from a timegraph with undeterministic timing.

3.4 Live editing operations

Editing a document while it is active has several implications to scheduling, sampling and resolution of the timegraph. From the standpoint of scheduling, manipulating nodes may require a change in the timegraph in order to accommodate the schedule of the modification. Such schedule changes also affect sampling, as some of these changes, for instance time manipulations, require that parameters used in the sampling process be adapted to reflect the changes. Additionally, other groups of changes may add or remove indeterminate timing attributes to the document, which must also be consistently mapped to the timegraph.

As a consequence, performing editing operations in the document requires some mechanism to make the document and the underlying rendering state consistent. From the standpoint of document editing, the usual mechanism for manipulating structured documents, in the Web platform, is via the DOM. This API, which is supported by most scriptable user agents, allows some external program to manipulate document elements and attributes using object-oriented methods. Therefore, the problem of updating the rendering of a structured document is out of the scope of the DOM API. In the particular case of HTML, for instance, DOM implementations in user agents automatically enforce DOM modifications in the rendering data structures. Analogously, any rendering engine whose input document is edited via DOM operations should also provide some custom mechanism to keep the DOM and the rendering data structures in a consistent state. This is the case of the ActiveTimesheets engine, which relies on DOM operations as a live editing API. Some alternative solutions to the DOM-timegraph consistency problem are:

a) a change notification mechanism, which allows that every operation in the document syntax can be captured and treated so as to satisfy timegraph consistency;

b) a DOM inheritance strategy, which extends the DOM by overloading a selection of methods that affect the timegraph;

Alternative *(a)* is the less obtrusive, given that it would require only the definition of handlers for every notification of a modification. Additionally, the regular DOM API could be used to edit the document and have its rendering updated in a seamless manner. However, such notification mechanism should be consistently supported across user agents, which is not a reasonable assumption, given that such a feature is not part of the current DOM recommendation [W3C, 2004c]. Alternative *(b)*, on the other hand, may be more obtrusive because it would require the use of a custom API provided by the ActiveTimesheets engine. Additionally, it would include rendering, an out-of-scope functionality, into the the DOM API. On the side of advantages, it is more portable, as it does not depend on specialized feature support across user agents. Additionally, it stands out on flexibility, because it allows more fine grained control over the update behavior, for instance by defining new operations for batch-based updates, which could significantly reduce the cost of timegraph processes in certain situations. Finally, the DOM API is the only route where modifications to the temporal document are guaranteed to be intercepted,

consequently it is the best candidate, as far as the technology goes, to ensure that temporal rendering gets consistent with the temporal document.

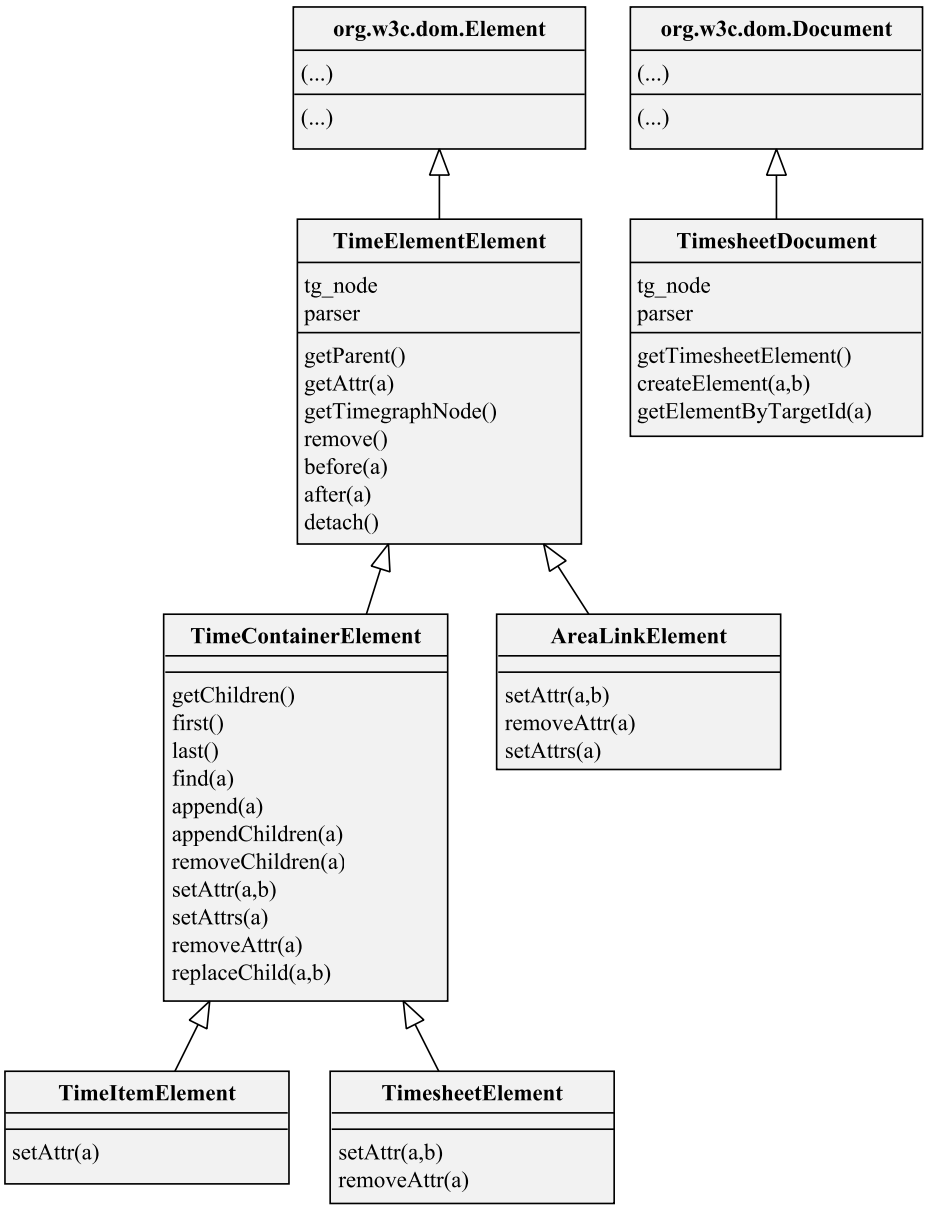


Figure 3.12: ActiveTimesheets DOM API

In ActiveTimesheets, both modes of DOM-timegraph consistency are feasible. However, in order to obatin more flexibility, the approach presented here is based on an inheritance mechanism, i.e., ActiveTimesheets defines a custom DOM API that extends the original DOM with new functionality (Figure 3.12). The element *TimesheetDocument* extends a DOM document (*org.w3c.dom.Document*): it allows not only retrieval operations over the document but also creating new ActiveTimesheet nodes. Upon creation of a new node, not only a DOM node is created, but also a new timegraph node corresponding to the DOM specification (attribute *tg_node*). The nodes created by *TimesheetDocument* belong to a hierarchy of objects, each

one of them representing one possible object in the ActiveTimesheets language. All of these elements derive its functionality from an abstract element, *TimeElementElement*, which is a specialization of the general element (*org.w3c.dom.Element*) node in the DOM model. This class also associates the DOM element to corresponding timegraph node which, depending on the composition structure abstracted by the syntax, may be a single node or a sub-timegraph. All of the other ActiveTimesheet elements are concrete specializations of *TimeElementElement*, of which the *TimeContainerElement* class has an important role: it not only allows manipulation of `par` and `seq` nodes, but also other elements which are composite structures: *TimeItemElement* and *TimesheetElement*. This is because, at the timegraph level, both `item` (in the composite case) and `timesheet` elements are represented as implicit time containers, as it will be demonstrated further in this Section. The class *AreaLinkElement*, which represents `area` links in ActiveTimesheets, is not a composition structure, consequently it is derived directly from *TimeElementElement*.

On defining the underlying operations that satisfy DOM-timegraph consistency, it is important to distinguish the operations that affect the timegraph and the extent of these modifications, i.e., which timegraphs processes it affects. Table 3.2 lists all editing operations in ActiveTimesheets language and discriminate which of them require timegraph updates. Here timegraph processes are discriminated as scheduling, sampling and resolution, which were already discussed. Additionally, there are formatting operations, which encompass any operation whose effect in the document layout can be perceived by a user. Not all of operations will require timegraph updates: this is case, for instance, of a-temporal attributes such as `sourceLevel` and `alt`, whose effects are perceived in the presentation but do not affect temporal layout. The most critical operations are the ones that affect timegraph schedule, since a change in schedule will also affect sampling, formatting and, potentially, resolution, in case indeterminate timing is incorporated. There are also some operations that may only affect sampling: this is the case of `destinationPlayState`, which will create conditions that are evaluated only at sampling time. A rare situation is an operation that affects only resolution: this is the case of `href`, whose second-order effects may lead to constraint-based resolution of the timegraph due to link activation.

In the remaining of this Section the discussion will focus, first, on element and attribute manipulation in a general form, in Section 3.4.1. Then a more elaborate editing operation, regarding timing manipulations, is discussed in Section 3.4.2. Operations in other modules deserve separate Sections, because they involve more elaborate consistency mechanisms or include novel language extensions: this is the case of the Linking group, which will be discussed in Section 3.5, the MediaObject group, whose editing of clipping attributes is discussed in Section 3.6.2, and part of the Timesheets group, whose editing is discussed in Section 3.6.1.

Table 3.2: List of ActiveTimesheets editing operations and their effects on timegraph processes. Due to space constraints, process labels are shortened as follows: SCHDL = scheduling, SMPL = sampling, RSLT = resolution, FRMT = formatting

Component	Name	SCHDL	SMPL	RSLT	FRMT	Group
attribute	accesskey				X	Linking
attribute	actuate			X	X	Linking
attribute	alt				X	Linking
attribute	destinationLevel				X	Linking
attribute	destinationPlaystate		X		X	Linking
attribute	href			X	X	Linking
attribute	nohref				X	Linking
attribute	sourceLevel				X	Linking
attribute	sourcePlaystate				X	Linking
attribute	tabindex				X	Linking
attribute	target				X	Linking
element	area	X	X	X	X	Linking
attribute	clipBegin	X	X		X	Media Object
attribute	clipEnd	X	X		X	Media Object
element	meta					Metainformation
element	metadata					Metainformation
element	prefetch				X	PrefetchControl
attribute	speed	X	X		X	Time Manipulations
attribute	begin	X	X	X	X	Timing and Synchronization
attribute	dur	X	X	X	X	Timing and Synchronization
attribute	end	X	X	X	X	Timing and Synchronization
attribute	endsync	X	X	X	X	Timing and Synchronization
attribute	repeat	X	X	X	X	Timing and Synchronization
attribute	repeatCount	X	X	X	X	Timing and Synchronization
attribute	repeatDur	X	X	X	X	Timing and Synchronization
attribute	syncBehavior		X		X	Timing and Synchronization
attribute	syncBehaviorDefault		X		X	Timing and Synchronization
attribute	syncMaster	X	X	X	X	Timing and Synchronization
attribute	syncTolerance		X		X	Timing and Synchronization
attribute	syncToleranceDefault		X		X	Timing and Synchronization
element	excl	X	X	X	X	Timing and Synchronization
element	par	X	X	X	X	Timing and Synchronization
element	seq	X	X	X	X	Timing and Synchronization
element	item	X	X	X	X	Timesheets
element	timesheet	X	X	X	X	Timesheets
attribute	select	X	X	X	X	Timesheets
attribute	beginInc	X	X	X	X	Timesheets
attribute	src	X	X	X	X	Timesheets

3.4.1 Editing elements and attributes

In the ActiveTimesheets DOM API, modification of elements and attributes consists on, first, performing the modification in the extended DOM and, after, translating the modification to the timegraph. As the timegraph is updated, its schedule needs to be updated as well. All these steps are performed during a single atomic operation. As Table 3.2 illustrates, editing operations, regarding document components, can be divided in attribute editing and element editing.

Algorithm 3.7: Element addition on a time container

```
1:  Let this be the current element
2:  function TIMECONTAINERELEMENT.APPEND(element)
3:      Let D be the extended DOM node
4:      Let N_parent be the wrapped timegraph node
5:      Let N_child be element's timegraph node
6:
7:      D.SETATTRIBUTE(name, value)
8:      N_parent.APPENDCHILD(N_child)
9:      if tag(element) is one of par|seq|excl|item|area then
10:         N_parent.UPDATESCHEDULE( )
11:     end if
12: end function
```

Regarding element editing, Algorithm 3.7 demonstrates how elements are added to a document. Inserting of a new node is demonstrated, in the Algorithm, by manipulating a composite element, a time container, by means of the method `append`. First, the attribute is assigned to the extended DOM node. Second, the new node is appended to the wrapped timegraph node. Finally, the wrapped timegraph node has its scheduling updated in order to reflect the changes, via the operation *updateSchedule*. In fact, the schedule is only updated if the node that was added affect the schedule: this is the case of a restricted set of elements, in particular `par`, `seq`, `excl`, `item` and `area` tags. Other tags, such as `metadata` and `prefetch`, or any other, have no influence on scheduling and, therefore, do not require an update.

Algorithm 3.8: Element removal on a time container

```
1:  Let this be the current element
2:  function TIMECONTAINERELEMENT.REMOVECHILD(child)
3:      Let N be the wrapped timegraph node
4:      Let D be the extended DOM node
5:      Let N_child be child's timegraph node
```

```
    D.REMOVECHILD(child)
    N.REMOVECHILD(N_child)
    if tag(child) is one of par|seq|excl|item|area then
        N.UPDATESCHEDULE( )
    end if
end function
```

Removal of a node follows a similar procedure, demonstrated in Algorithm 3.8. The method `removeChild` of a time container takes as argument to child DOM node to remove. The procedure will first remove the child DOM node, and then remove the child timegraph node. After that, the container timegraph node has its schedule updated, in case the removed node is timed element, via `updateSchedule`.

Regarding attribute editing, Algorithm 3.9 demonstrates how attributes are edited in the case of `item` elements.

Algorithm 3.9: Attribute modification on `item`

```
Let this be the current element

function TIMEITEMELEMENT.SETATTR(name, value)
    Let D be the wrapped DOM node
    Let N be the wrapped timegraph node
    D.SETATTRIBUTE(name, value)
    if name is one of begin|dur|end|...|speed then      ▷ a check on allowed attributes
        Let A ← this.PARSER.ATTRIBUTE(name, value, D)   ▷ reparse the attribute
        N.SETATTRIBUTE(a.name, a.value)
        if name labels a formatting attribute then
            N.UPDATEFORMATTING( )
        else if name labels a scheduling attribute then
            N.UPDATESCHEDULE( )                         ▷ update timegraph schedule
        end if
    else if name is select then
        Let N_old be the current timegraph node
        Let N_new ← this.PARSER.TIMEELEMENT(D)          ▷ reparse the time element
        Let N_parent ← N_old.parent
        N_parent.ADDCHILDAFTER(N_new, N_old)
        N_parent.REMOVECHILD(N_old)
        N_new.UPDATESCHEDULE( )                         ▷ update timegraph schedule
    end if
end function
```

In Algorithm 3.9, the `item` element has been chosen as an example because its syntax, especially its `select` attribute, leads to a different treatment when compared to other nodes, consequently is allows to demonstrate more issues of editing operations. The first important action taken by the algorithm is updating the element DOM node (line 5) by assigning the new attribute to it. In case the new attribute is not `select`, it is parsed and assigned to the timegraph node (lines 7-8). After that, the timegraph has its scheduling and/or formatting updated (recall that scheduling updates always imply formatting updates, but the contrary is not true). Compared to other attributes, the `select` attribute can have more profound consequences over the timegraph. This is because the `item` element can represent, depending on the selection expression, a single media element or a composite structure of media elements. Consequently, depending on the change in the expression, the node needs to be replaced by another different one (for instance a *DiscreteMediaElement* replaced by a *ContinuousMediaElement* or by a *TimeContainerSeq*). Consequently, upon a modification in the `select` attribute, the whole node needs to be reparsed (line 16). After that, the node is replaced in its parent container, concluding with an incremental schedule update.

An important requirement for a schedule update is that only the subgraph affected by the operation must be recomputed. This is achieved by operation `updateSchedule`. A possible strategy to such a partial update problem is to take advantage of the partial schedule update operation presented in Algorithm 3.5. This operation, designed originally for timegraph resolution, will propagate the new scheduling effect up to the root of the timegraph. This behavior is expected because unresolved timing attributes always imply unresolved timing in their parents, consequently the whole chain of ascendant nodes is unresolved. Differently from timing resolution, updates originated from live editing do not always require that schedule updates be propagated up to the root node. In fact, only those ancestor nodes who have their active interval modified by the editing operation need to be updated. Figure 3.13 adapts the timegraph previously presented in Figure 3.5b in order to illustrate this situation.

The timegraph in Figure 3.13 introduces a composition (*par:85*) which aggregates part of nodes corresponding to slides in the original presentation. In fact, this composition illustrates how a `par` time container can simulate the behavior of a `seq` container in certain situations. Upon insertion of a new node (*dm:86*) in this container, the timegraph schedule should be updated. In order to do that, after inserting the new node, the time container attempts to update its internal schedule. But, as *dm:86* active interval is already contained in the active interval of *par:85*, then no change will be observed in the container active interval. As a consequence, no other update is expected in any further ancestor and the update procedure can stop at this point.

Algorithm 3.10: Incremental timegraph node scheduling

1: **function** TIMEELEMENT.UPDATESCHEDULE( )

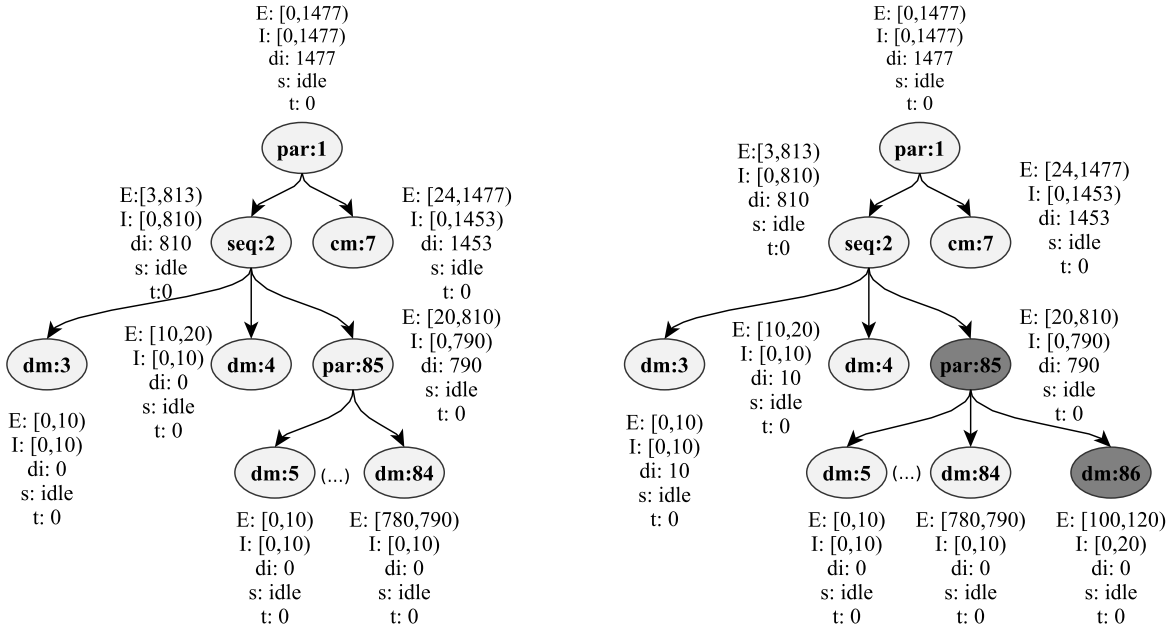


Figure 3.13: Example of an incremental timegraph schedule update due to live editing

2: Let B, E, and D be the `begin`, `end` and `dur` attributes of $element$, respectively
3: Let $this.i_a$ be the element active interval
4: $this.d_i \leftarrow$COMPUTEPARTIALIMPLICITDURATION( )
5: $b \leftarrow$ COMPUTEBEGIN(B)
6: $d \leftarrow$ COMPUTEDURATION(D, E, b)
7: Let $i_{new} \leftarrow [b, b + d)$
8: **if** $this.i_a \neq i_{new}$ **then**
9: $this.i_a \leftarrow i_{new}$
10: **if** $this.parent$ is not the root node **then**
11: $this.parent$.UPDATESCHEDULE( )
12: **end if**
13: **end if**
14: **end function**

An incremental scheduling process which performs this optimization is demonstrated in Algorithm 3.10. The algorithm is very similar to partial scheduling (Algorithm 3.5). The main difference stands on an additional verification (line 10) to check if the active interval of the current element has changed. In case it has changed, the scheduling update will propagate to the immediate parent, if any. Otherwise, the update will stop at this node. The example in Figure 3.13 presents, in fact, a best case scenario for the algorithm, i.e., the scheduling is propagated to the immediate parent of the newly inserted node. In the case of attribute modification, the best case

scenario would occur when the edited attribute do not change the element interval: in this case, the change would not even be propagated to the parent. The worst case scenario would be the same as of that of partial schedule update: in case the timegraph is degenerate, the complexity would be similar to that of a full schedule update. However, for practical purposes, big SMIL documents are unlikely to lead to degenerate timegraphs, as long as meaningful presentations are concerned. Thus, it is expected that in the majority of situations, the worst case is the same as partial schedule update, i.e. $O(depth(n))$. Additionally, whenever an ancestor node does not change its active interval, the complexity can be smaller, namely $O(depth(n) - depth(m))$, where *n* is the modified node and *m* is the last ancestor who needed to be updated.

Another performance improvement measure is to reduce the number of timegraph updates. This measure is particularly useful in certain timegraph manipulation patterns, for instance: *i)* when multiple elements are inserted into a single time container; and *ii)* when multiple timing-related attributes of an element are changed at once. In situations like these, the modifications occur in a sequence and, for every operation, if the algorithms previously presented are used, an incremental schedule update is triggered. A feasible solution for this problem consists on defining an operation which performs the sequence of DOM modifications in batch and, after that, generate a single incremental timegraph update to consolidate the changes. Algorithm 3.11 presents an example of such operation in the case of batch insertion of elements.

Algorithm 3.11: Batch insertion of time container children

```
1:  function APPENDCHILDREN(children[ ])
2:      Let D be the extended DOM node
3:      Let N be the wrapped timegraph node
4:      Let recomp ← true
5:      for all child in children[ ] do
6:          Let N_child be child's timegraph node
7:          D.APPENDCHILD(child)
8:          N.APPENDCHILD(N_child)
9:          if tag(child) is one of par|seq|excl|item|area then
10:             recomp ← true
11:         end if
12:     end for
13:     if recomp then
14:         N.UPDATESCHEDULE( )
15:     end if
16: end function
```

In Algorithm 3.11, the function `appendChildren` receives an array of nodes to be inserted

in the DOM. The algorithm traverses the array, inserting each DOM node and its corresponding timegraph node in the corresponding structures. If any of the inserted nodes require a timegraph update, this operation will only be triggered after all the nodes have already been inserted. In ActiveTimesheets, similar operations are defined for removal of nodes, and addition, update and removal of attributes.

3.4.2 Dynamic time manipulations

In regular SMIL documents, the timing of all elements progress at the same rate, i.e., that of the global sampling clock. Therefore, the only timing difference between active elements in a document is given by the offsets of their timelines in the hierarchy. Timing manipulations is a module that allows individual time elements to progress their timing in a different rate from that of the global sampling clock. The module has been strongly associated to time-based animation features in SMIL, particularly because the timing manipulation attributes allow to control the speed and acceleration of animations, for instance. But timing manipulations are also useful for non-animated presentations, for instance, for changing the playback rate of a time container and having the playback rate of all its children changed as well. Naturally, a similar effect could be achieved by modifying the rate of the reference clock used in the sampling process. But, in this case, the modification would be effective for the whole presentation, which is not desired in all situations. For instance, in complex presentations with multiple media elements without a "master track", it is useful to have the ability to modifying the speed of individual media elements or compositions.

The Timing manipulations is the normative method to change the playback speed of elements in SMIL. This is achieved by assigning speed and acceleration attributes to individual media elements. For instance, if an element has a speed value of 2, its timing progresses at the double of its parent rate. The inclusion of timing manipulations in the SMIL timing model has several consequences on timegraph sampling, scheduling and live editing. In this Section the discussion is focused on the manipulation of the `speed` attribute, which is the one supported by ActiveTimesheets; other timing manipulation attributes, such as `accelerate`, `decelerate` and `autoreverse`, are not included in this model. Including them, in a future version, will require computations different than the ones presented here. First, the manipulation of timing in static documents is discussed, in order to contextualize the issue. After, a solution for timing manipulations in dynamic documents is presented, as it is incorporated in the ActiveTimesheets timing model.

Manipulations in static documents

Regarding effects of timing manipulation on timegraph sampling, if an element has been assigned a speed value s, then the timing of the element is computed via the function given in Equation 3.4, where:

$$\begin{aligned} t^f &= t \cdot s \\ &= (T - b - O) \cdot s \end{aligned} \tag{3.4}$$

- t^f is the filtered element time, i.e. the element time after timing manipulation;
- t is the local element time, given previously by Equation 3.3;
- T, b, O are the parent time, begin offset and slip factor, also defined in Equation 3.3;
- s is the element speed.

Therefore, in the presence of timing manipulation each element must have its own time function which is used in the sampling process. Additionally, a consequence of timing manipulation in the SMIL hierarchical model is that speed rates are cascaded top-down through the element hierarchy. Consequently, the speed of the element is perceived in the presentation as the combination of the local speed and the speed of its parent, i.e. if an element has local speed 3 and the parent has local speed 2, then the cascaded speed of the element is $2 \cdot 3 = 6$. In summary, for every element, the cascaded speed is given by Equation 3.5, where:

$$s^c = s_p^c \cdot s \tag{3.5}$$

- s^c is the element cascaded speed;
- s_p^c is the parent cascaded speed;
- s is the element speed.

The cascaded speed is directly accounted for in the hierarchical time function presented in Equation 3.4, because local time is computed based on parent time. However, continuous media elements, such as audio and video objects, have an internal sampling process whose execution is independent of the timegraph. As a consequence, for such media objects, what applies is the cascaded speed.

Regarding effects on timegraph scheduling, the modification of an element speed also changes the element duration in certain circumstances. If the element duration has been defined in the element internal timeline, e.g. via `dur` attribute or via implicit duration, i.e., if the duration is *locally scoped*, then the element actual duration must be scaled, or filtered, in order to reflect the new speed. This is not the case for a duration defined by the `end` attribute, given that this attribute, alongside the `begin` attribute, is measured in parent time, thus it is not affected by speed changes. In this case, the duration is said to be *globally scoped*.

Figure 3.14 illustrates the behavior of a locally scoped duration by simulating the progression of an element time as a function of its parent time. Suppose the element has an implicit duration lasting $15s$. If no speed modification is applied to the element, i.e., when $s = 1$, then the element

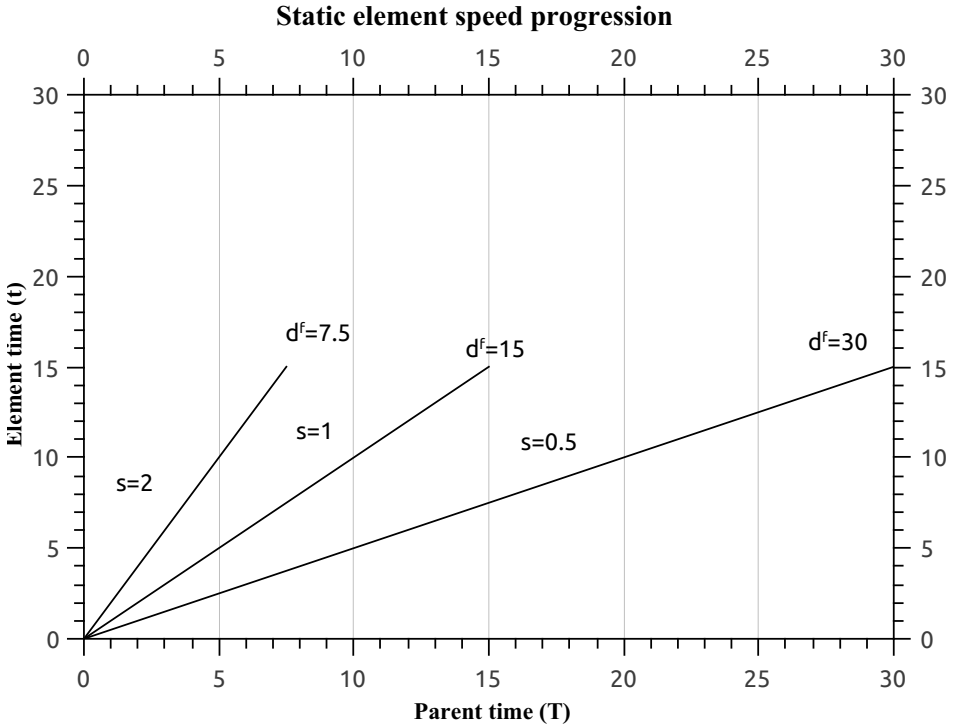


Figure 3.14: Effects of different speed manipulations on element implicit duration

filtered duration is the same as its implicit duration. On the other hand, when $s = 2$, the element filtered duration is 7.5, i.e., half the implicit duration. Analogously, if the element speed is half the sampling rate, i.e., $s = 0.5$, then the element filtered duration is the double of the implicit duration. A consequence of this behavior is that, in face of timing manipulation, the internal and external timelines of a node may differ in their durations. Additionally, as long as the scheduling process depends on the element external timeline, the effects of timing manipulations must be treated during scheduling.

For each element the filtered duration is given by Equation 3.6, where:

$$d^f = \begin{cases} \dfrac{d}{s} & \text{if } s \neq 0 \text{ and } d \text{ is locally scoped} \\ d & \text{if } s = 0 \text{ or } d \text{ is externally scoped} \end{cases} \tag{3.6}$$

- d^f is the element filtered duration, i.e., the duration in the external timeline;
- d is the element unfiltered duration, i.e., the duration in the internal timeline;
- s is the element speed.

Because the duration is filtered, the element external timeline, or active interval, should be filtered as well. Let b be the element begin offset. The filtered active interval is given by Equation 3.7.

$$E^f = [b, b + d^f) \tag{3.7}$$

A limitation of the timing manipulation model presented so far is that it only works for static documents, i.e. those at which the `speed` attribute is defined before rendering. If a live editing operation manipulates the `speed` attribute during document runtime, the time function given by Equation 3.4 is not enough to encompass this change. This is because Equation 3.4 models the element time as a linear function of the parent time. However, when the speed attribute changes, the time function also changes, and this change must be accounted for in the element duration. A solution for this problem is discussed in the next Section.

Manipulations in dynamic documents

When a timing manipulation attribute is edited while a document is active, the implicit duration of the element changes as well. As a consequence, the timegraph schedule should be updated in order to reflect this change. The incremental schedule update process previously presented in Algorithm 3.10 suffice this requirement only to a limited extent. Even though no change is needed in the strategy of the algorithm, the method for computing the element implicit duration should be adapted. In order to understand which changes are necessary, Figure 3.15 illustrates the behavior of the time function of an element in face of live timing manipulations.

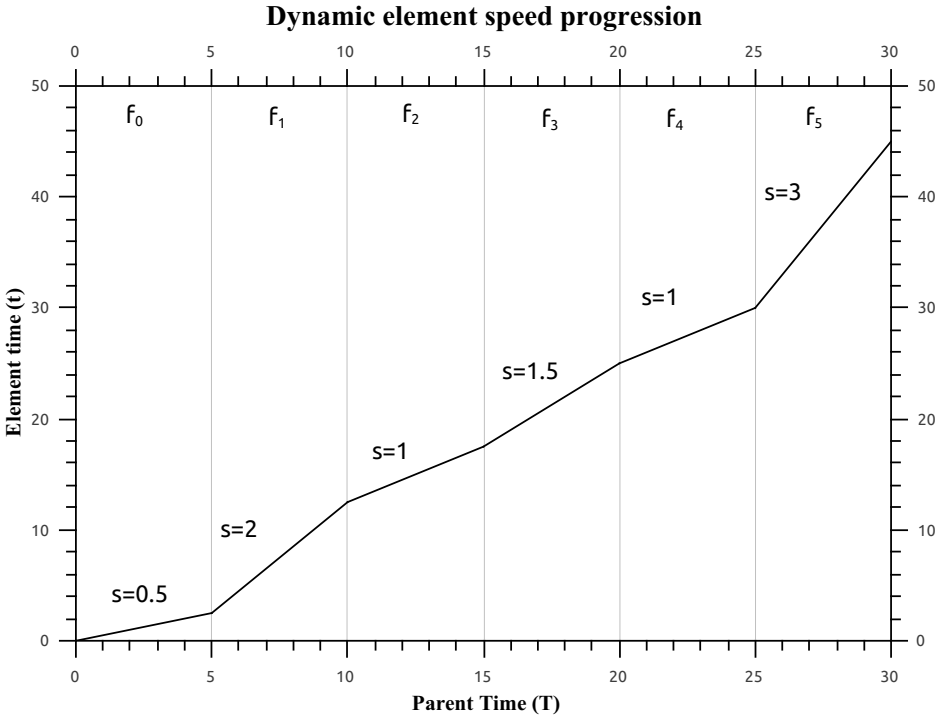


Figure 3.15: Effects of live timing manipulations on an element implicit duration

Figure 3.15 simulates the execution of an element during an interval of $30s$, counted in

the temporal scope of its parent container (T). During this interval, the speed of the element is changed at periodical intervals, i.e., at every $5s$, to a random value. As a consequence, the element is started at $T = 0$ with $s = 0.5$ and continues at this rate until $T = 5$. At this moment, the speed changes, so as $s = 2$, continuing at this rate until $T = 10$. More speed changes follow and, for each one of them, the element time (t) progresses at a faster or slower pace, depending on the value of s that was assigned. As a consequence of all speed changes, when $T = 30$, the element time reaches an accumulated duration of $t = 45$. Such temporal behavior, caused by speed manipulations, will have the following consequences for the element timegraph:

- *sampling*: every speed change also changes the time function of the element, i.e., it changes how the local time of the element should be computed from the point of the change onward. In order to solve this issue, it is necessary to define a time function that properly describes the behavior of timegraph nodes when their timing has been manipulated;

- *scheduling*: every speed change also updates the estimated implicit duration of the element. This is because a speed change splits the implicit duration of an element in two partitions: *i)* the duration that have elapsed before the change; *ii)* the remaining duration that is estimated based on the last speed. In order to solve this issue, it is necessary a method to accurately compute the implicit duration of an element in face of timing manipulations.

In order to tackle all these issues, a first step is to generalize the function that describes the timing of the element. As the plot suggests, the time function is a piece-wise defined function, where each piece, labeled $f_0, ...f_5$ in the plot, is bootstrapped by a new speed value. Additionally, each of the pieces is a linear function described via Equation 3.4. The first individual function, f_0, starts at the coordinates $(0, 0)$, whereas the next individual function, f_1, starts when f_0 finishes, i.e., f_1 is translated to the final coordinates of f_0. The same behavior is observed for the subsequent functions. Accordingly, in order to define the piecewise time function, it is necessary to define in advance each of its pieces. The first step to do so is to find a general expression that describes a translated form of Equation 3.4. The general formula for translating a linear function $f(x)$ is given by:

$$g(x) = f(x - h) + v \tag{3.8}$$

Where $g(x)$ is the translated function, x is a time value in the domain of the function, and (h, v) is the coordinate where the function will be translated to: h represents the horizontal translation whereas v represents the vertical translation. Let $S = ((s_0, h_0, v_0), (s_1, h_1, v_1), ..., (s_n, h_n, v_n))$ be a sequence of speed changes, where each speed s_k is assigned at the coordinate (h_k, v_k), with $0 \leq k \leq n$. As a consequence of these changes, the filtered time of an element will be described by a piecewise function with n pieces, given by $t_0^f, t_1^f, ..., t_n^f$. Applying the general translation form to Equation 3.4, leads to the filtered time function for an individual piece k as follows:

$$t_k^f(T) = [(T - b - O) - h_k] \cdot s_k + v_k \qquad \text{if } T \in (h_k, h_{k+1}] \tag{3.9}$$

As a piece of the function, Equation 3.9 is defined for only a partition of the global function, which in the equation is a left-open interval between the current speed change and the next speed change. Therefore, building the global function upon these definitions gives the piecewise-defined filtered time function of Equation 3.10.

$$t^f(T) = \begin{cases} t_0^f : [(T - b - O) - h_0] \cdot s_0 + v_0 & \text{if } T \in [h_0, h_1) \\ t_1^f : [(T - b - O) - h_1] \cdot s_1 + v_1 & \text{if } T \in [h_1, h_2) \\ t_2^f : [(T - b - O) - h_2] \cdot s_2 + v_2 & \text{if } T \in [h_2, h_3) \\ \quad \vdots & \quad \vdots \\ t_n^f : [(T - b - O) - h_n] \cdot s_n + v_n & \text{if } T \in [h_n, b + d^f) \end{cases} \tag{3.10}$$

In Equation 3.10, the last piece, t_n^f, gives the element local time at any parent time moment. This is because the domain interval of the last piece is the one that overlaps the element current time. As the time function is dynamic, new pieces may be added every time the speed changes, thus n is also variable during the element execution. Accordingly, the upper bound of the domain of the last piece is variable, i.e. it depends on the element filtered duration, d^f, that should be estimated at every speed change. In order to achieve that, the element external timeline, $E = [b, b + d^f]$, needs to scaled from the element internal timeline, $I = (I_b, I_e)$. This can be done by replacing the bounds of the internal timeline in the first and last pieces of Equation 3.10. In order generalize this idea, first Equation 3.11 isolates parent time, T, from a general piece of Equation 3.10.

$$T_k = \frac{t_k^f(T) + (b + O + h_k) \cdot s_k - v_k}{s_k} \qquad \text{if } T_k \in (h_k, h_{k+1}] \tag{3.11}$$

If $t_k^f(T)$ is replaced by the simplified variable t, it is possible to define a function that converts local time to parent time, of course valid only for this piece.

$$T_k(t) = \frac{t + (b + O + h_k) \cdot s_k - v_k}{s_k} \qquad \text{if } T_k \in (h_k, h_{k+1}] \tag{3.12}$$

Let T_0 and T_n be the first and last pieces, respectively, in the form of Equation 3.12. Let $I = (I_b, I_e)$ be the element internal timeline. The filtered implicit duration is given as the difference between the last and first pieces, when the local times are parameterized to I_b and I_e, respectively. This is represented in Equation 3.13.

$$\begin{aligned} d^f &= T_n(I_e) - T_0(I_b) \\ &= \frac{I_e + (b + O + h_n) \cdot s_n - v_n}{s_n} - \frac{I_b + (b + O + h_0) \cdot s_0 - v_0}{s_0} \end{aligned} \tag{3.13}$$

In order to demonstrate the applicability of the aforementioned equations in a concrete element, Equation 3.14 instantiates Equation 3.10 to the the example in Figure 3.15. This equation gives the final function for parent to local time conversion (i.e. the function after all speed changes have been performed). In order to subsidize this process, Table 3.3 tabulates the relevant information from Figure 3.15.

Table 3.3: Tabulation of translation points and speed for piece in the function of Figure 3.15

f_x	h	v	s	B	O
f_0	0	0	0.5	0	0
f_1	5	2.5	2	0	0
f_2	10	12.5	1	0	0
f_3	15	17.5	1.5	0	0
f_4	20	25	1	0	0
f_5	25	30	3	0	0
-	30	45	3	0	0

$$t^f(T) = \begin{cases} t_0^f : [(T-0-0)-0]\cdot 0.5 + 0 & \text{if } T \in [0,5) \\ t_1^f : [(T-0-0)-5]\cdot 2 + 2.5 & \text{if } T \in [5,10) \\ t_2^f : [(T-0-0)-10]\cdot 1 + 12.5 & \text{if } T \in [10,15) \\ t_3^f : [(T-0-0)-15]\cdot 1.5 + 17.5 & \text{if } T \in [15,20) \\ t_4^f : [(T-0-0)-20]\cdot 1 + 25 & \text{if } T \in [20,25) \\ t_5^f : [(T-0-0)-25]\cdot 3 + 30 & \text{if } T \in [25,d^f) \end{cases} \tag{3.14}$$

This function can be used to determine the local time based on parent time. Suppose the sampling process needs to sample the element at $T = 22$. In order do that, the sampling algorithm must take advantage of last defined piece in Equation 3.14 at this moment. This would correspond to the piece t_4^f, given that $T = 22 \in [20, 25)$. Naturally, regarding the simulated element lifetime, this would be the last known piece, provided that in a simulated situation, the piece t_5^f would not have been created yet. Consequently, the element current time local time is given by:

$$t_4^f(22) = [(22-0-0)-20]\cdot 1 + 25 = 22 - 20 + 25 = 27s$$

This same procedure should be done at any time during the element active interval, respecting the restriction that always the last known piece must be used. Next, Equation 3.15 gives the functions for local to parent time conversion at every piece.

$$
\begin{aligned}
T_0(t) &= \frac{t + (0+0+0)\cdot 0.5 - 0}{0.5} && \text{if } T_0 \in (0, 5] \\
T_1(t) &= \frac{t + (0+0+5)\cdot 2 - 2.5}{2} && \text{if } T_1 \in (5, 10] \\
T_2(t) &= \frac{t + (0+0+10)\cdot 1 - 12.5}{1} && \text{if } T_2 \in (10, 15] \\
T_3(t) &= \frac{t + (0+0+15)\cdot 1.5 - 17.5}{1.5} && \text{if } T_3 \in (15, 20] \\
T_4(t) &= \frac{t + (0+0+20)\cdot 1 - 25}{1} && \text{if } T_4 \in (20, 25] \\
T_5(t) &= \frac{t + (0+0+25)\cdot 3 - 30}{3} && \text{if } T_5 \in (25, d^f]
\end{aligned}
\tag{3.15}
$$

When the element is activated its time function has only one piece, f_0. After the first speed change, the time function will have two pieces, f_0 and $f1$. This continues until the element has five pieces. Thus, for each modification, the time function has a different state. Consequently, the filtered duration will be different for each state. Supposing the example element has an internal timeline $I = [0, 40]$, then Table 3.4 gives the estimated filtered duration for the element at every state, computed based on the formulas in Equation 3.15.

Table 3.4: Estimated filtered duration for each state of the time function of the element in Figure 3.15

f_x	h	v	s	d^f
f_0	0	0	0.5	$T_0(40) - T_0(0) = 80$
f_1	5	2.5	2	$T_1(40) - T_0(0) = 23.75$
f_2	10	12.5	1	$T_2(40) - T_0(0) = 37.5$
f_3	15	17.5	1.5	$T_3(40) - T_0(0) = 30$
f_4	20	25	1	$T_4(40) - T_0(0) = 35$
f_5	25	30	3	$T_5(40) - T_0(0) = 28.33$

As Table 3.4 demonstrates, upon inclusion a piece in the time function, the expected filtered duration assumes a different value. This value is dependent on the coordinates of the change (i.e. h and v measured in parent time and local time, respectively) and the new speed. When the time function has only one piece, f_0, the duration is estimated to $80s$, i.e. the duration with a speed of 0.5. Upon the next speed change, to 2, the expected duration is estimated via the second piece, f_1, being now 23.75. Such estimate considers the elapsed time and the remaining time at the current pace. This behavior is repeated as long as new speed changes occur, consequently new function pieces are added. When the last piece is added, f_5, with a speed 3, the element is expected to have a duration of $28.33s$. This estimate considers the internal elapsed time, $25s$,

more the remaining time at a speed of 3, $\frac{(40-30)}{30} = 3.33$, which gives $28.33s$.

The equations defined so far are used for various timegraph processes. Equation 3.10 is used for sampling, since it enables converting parent time to local time. Equation 3.13 is used for scheduling, as it allows to compute the external timeline, or active interval of an element. And Equation 3.12 is applied to linking resolution, an issue that will be discussed later in the Chapter, as it allows converting local time to parent time. In the following, the most important algorithms to tackle these issues are presented. First, the original definition for a timegraph node should be updated in order to include a time function for each element. Let n be a timegraph node, then the extended timegraph node definition is given by Equation 3.16, where:

$$\begin{aligned} &n = (I, E, d_i, t, s, F) \\ &F = [F_0, F_1, ..., F_k] \\ &F_i = (h_i, v_i, S_i) \qquad 0 \leq i \leq k \end{aligned} \tag{3.16}$$

- $F = [F_1, .., F_k]$ is the element speed function, composed of k pieces, one for every speed change. Each piece i is composed by the function horizontal translation, h_i, vertical translation, v_i, and speed that lead to its creation, S_i;
- S is the current element speed, i.e., the speed assigned at the k^{th} function piece.

Based on this redefined structure, Algorithm 3.12 presents parent to local time conversion.

Algorithm 3.12: Parent to local filtered time conversion

1: **function** TIMEELEMENT.CURPARENTTOLOCAL(T, B, O)
2: Let F_k be the last function piece
3: **return** $(T - B - O - F_K[h]) \cdot F_k[S] + F_k[v]$
4: **end function**

The basic idea of Algorithm 3.12 is to use the last piece of Equation 3.10 in order to perform the conversion. As the function is dynamically constructed, the last piece must obtained at every call. A similar idea is used in Algorithm 3.13.

Algorithm 3.13: Local to parent filtered time conversion

1: **function** TIMEELEMENT.CURLOCALTOPARENT(t, B, O)
2: Let F_k be the last function piece
3: **return** $(t + (B + O + F_k[h]) \cdot F_k[S] - F_k[v])/F_k[S]$
4: **end function**

In the case of Algorithm 3.13, the inverse procedure is done, i.e. local time is converted to parent time. This is achieved by also taking advantage of the last piece of the time function, but this time with the variant established in Equation 3.12. This equation is also used in Algorithm 3.14, but with a different strategy.

Algorithm 3.14: Computation of element active duration

1: **function** TIMEELEMENT.COMPUTEDURATION(D, E, b, B, O)
2: Let d be the computed duration
3: Let F_0 be the first function piece
4: Let F_K be the last function piece
5: Let $[I_b, I_e]$ be the internal timeline of the element
6: **if** E is constraint-based and unresolved **then**
7: $d \leftarrow \infty$
8: **else if** E is constraint-based and resolved **then**
9: $d \leftarrow t - b$, where t is the constraint solution
10: **else if** D is timeline-based **then**
11: $t_0 \leftarrow (I_b + (B + O + F_0[h]) \cdot F_0[S] - F_0[v])/F_0[v]$ ▷ Explicit filtered duration
12: $t_k \leftarrow ((I_b + D) + (B + O + F_k[h]) \cdot F_k[S] - F_k[v])/F_k[v]$
13: $d \leftarrow t_k - t_0$
14: **else if** D and E are undefined **then**
15: $t_0 \leftarrow (I_b + (B + O + F_0[h]) \cdot F_0[S] - F_0[v])/F_0[v]$ ▷ Implicit filtered duration
16: $t_k \leftarrow ((I_b + this.d_i) + (B + O + F_k[h]) \cdot F_k[S] - F_k[v])/F_k[v]$
17: $d \leftarrow t_k - t_0$
18: **end if**
19: **return** d
20: **end function**

Algorithm 3.14 adapts Algorithm 3.4 for duration computation to include the restriction of filtered duration. In particular, the algorithm is modified so that both the implicit and explicit duration are filtered. In order to do so, Equation 3.13 is applied taking as reference the explicit duration (lines 10-11) and the implicit duration (lines 15-16). In order to keep the scheduling procedure with these changes, the partial scheduling algorithm must be adapted to update the time function at every speed change. Algorithm 3.10 demonstrates these changes.

Algorithm 3.15: Incremental timegraph node scheduling with timing manipulations

1: **function** TIMEELEMENT.UPDATESCHEDULE( )

2: Let B, E, and D be the `begin`, `end` and `dur` attributes of $element$, respectively
3: Let $this.i_a$ be the element active interval
4: $this.d_i \leftarrow$COMPUTEPARTIALIMPLICITDURATION( )
5: $b \leftarrow$ COMPUTEBEGIN(B)
6: UPDATETIMEFUNCTION( ) ▷ Add a new piece if speed has changed
7: $d \leftarrow$ COMPUTEDURATION(D, E, b, B, O) ▷ Computes the filtered duration
8: Let $i_{new} \leftarrow [b, b + d)$
9: **if** $this.i_a \neq i_{new}$ **then**
10: $this.i_a \leftarrow i_{new}$
11: **if** $this.parent$ is not the root node **then**
12: $this.parent$.UPDATESCHEDULE( )
13: **end if**
14: **end if**
15: **end function**

The most important change in the algorithm is the inclusion of a call to update the time function at every schedule update. The function `updateTimeFunction` (line 6) will only update the time function if the speed has been changed since the last update. Otherwise, the time function will stay the same. Additionally, a call to the updated `computeDuration` function, which beyond all changes also requires additional parameters, is also included. with all these changes, the timegraph can properly respond to live modifications on the speed attribute.

3.5 Extended linking and media fragments

Linking is an key resource for providing interactivity in multimedia documents. By means of links a multimedia document can provide, for instance, alternative narrative routes and non-linear access. As previously illustrated in Table 3.1, the SMIL Timesheets recommendation, as most of the surveyed languages and engines, does not include the Linking modules. In consequence, for such languages and engines, linking is available only via the spatial document, i.e., HTML or XHTML. However, linking syntax in HTML does not encompass all the needed linking features for multimedia documents, such as anchoring temporal fragments. In order to tackle this limitation, the ActiveTimesheets engine includes part of the SMIL linking modules. Due to inherent characteristics of these modules, this integration is not seamless, thus requiring additional measures. In the remaining of this section, it is reported: *i)* how linking functionality was integrated in SMIL timesheets; and *ii)* how linking granularity was extended using media fragments addressing mechanisms.

3.5.1 Linking in ActiveTimesheets

The core linking features in ActiveTimesheets are based on a subset of the SMIL Linking modules. Such a sub-setting approach is necessary because the original modules assume that SMIL is used also as a spatial layout language, which is not the case for temporal-only languages derived from it. As a consequence, some adaptations are necessary to sort out conflicts between constructs provided by SMIL and similar constructs provided by HTML. The following discussion first exposes the most important characteristics of SMIL linking and then reports the adaptations provided by ActiveTimesheets.

The SMIL Linking modules abstracts hyperlinks as spatio-temporal associations, i.e. each link has a spatial and a temporal scope. Additionally, the linking model adopted by SMIL in many respects is similar to that found in hypertext languages such as HTML, in which a link is defined over a source anchor and uni-directionally points to a target. But differently from a-temporal languages, link anchoring in SMIL can occur not only on the spatial scope but also in the temporal scope. From a syntax perspective, *spatial anchoring* occurs by wrapping a document element in an anchor tag. Additionally, instead of a whole element, a spatial fragment of a visual element can be anchored, for instance a polygonal hotspot over an image. *Temporal anchoring*, in its turn, specify the lifetime of a link, i.e., it specifies the interval in which the link can be activated: outside the designated interval, any attempt to activate the link (e.g. via a mouse click) will not traverse it (i.e. the target will not be loaded). In addition, temporal fragments of media elements can be anchors: as a consequence, the link becomes active only during the traversal of the designated interval in the intrinsic timeline of the media element. Finally, link *targets* can either be external or internal. In the case of *external targets*, any addressable URL is allowed. In the case of *internal targets*, they can be spatial fragments (in the case of visual elements), temporal fragments (in the case of continuous elements) or named elements. In order to define links, SMIL defines the following elements:

- the `a` element, which has similar semantics as its homonymous HTML element: it is used to wrap a spatial element as a link anchor;
- the `area` element, on the other hand, has a spatial-only counterpart in HTML. In SMIL, the elaborate semantics of this element allows the definition of both source anchors and target anchors. When defining a source anchor, it is applied to a spatio-temporal fragment of a media element. Therefore, the element contains attributes that allow the definition of hotspots over visual elements (e.g. `shape` and `coords`) and to discriminate an interval in the timeline of continuous media elements (e.g. `begin`, `dur`). In this case, the spatial restriction determines the spatial region where a user can activate the link. The temporal restriction, on the other hand, determines the time interval during which the anchor can be activated (outside this interval, the anchor is not interactive). When defining a target anchor, the element discriminates a temporal interval of a continuous media object. In doing so,

other anchors can point to this area element as a target and, when activated, the underlying media element will have its timeline sought to the beginning of the discriminated interval.

In ActiveTimesheets, the `a` element is dropped in favor of the HTML counterpart, i.e., wrapping a spatial element as an anchor is delegated to the spatial document. The `area` tag, however, is fully included in the language, with some modifications:

- spatial fragmentation is dropped, provided that ActiveTimesheets is a temporal language. Consequently, if spatial fragmentation is needed, this can be achieved via constructs of the spatial document language;
- it can be applied to any timed element, instead of only to continuous media elements. The SMIL recommendation prescribes the use of this element only to media objects, e.g. discrete and continuous media elements. In ActiveTimesheets, the applicability of the area tag is extended to any element that has an internal timeline, which includes time containers and `item` elements.

The inclusion of linking functionality has several consequences to the scheduling and the sampling of a SMIL document. First, links have an internal timeline, and therefore should be scheduled like any other timed elements. Consequently, live editing of a link attribute might influence the schedule of its time container. Second, from a scheduling perspective, the activation of a link having as target an unresolved element must resolve the element in a compulsory manner. Third, once a link is activated, the presentation should seek to the internal timeline of the element, which involves a number of temporal conversions. In this case, the live editing model provided by ActiveTimesheets directly influence the linking behavior, because the extended time function derived from timing manipulations changes the way such conversions are made. Even though the link activation procedure is well documented in the SMIL recommendation, it is discussed here in order to demonstrate such influence.

Algorithm 3.16: Link activation procedure

```
1:  function TIMEELEMENT.LINKACTIVATE(Δ)
2:      Let N_parent ← this.parent
3:      Let N_cur ← this
4:      Let t_seek ← Δ
5:      if N_cur is unresolved then                ▷ Compulsory resolution of the node
6:          Let N_p ← N_cur.parent
7:          Let found ← false
8:          Let N_visited ← [N_cur]
9:          while N_p is defined and found is false do
10:             if N_p is resolved then
```

```
11:                 found ← true
12:             else
13:                 N_visited.APPEND(N_p)
14:                 Let N_p ← N_p.parent
15:             end if
16:         end while
17:         if found is false then                ▷ Even the root is unresolved, resolve it
18:             N_p.LOCALRESOLVE( )
19:         end if
20:         while N_visited is not empty do
21:             Let v ← N_visited.POP( )
22:             v.LOCALRESOLVE( )                 ▷ Resolve the active interval begin
23:             Let N_p ← v
24:         end while
25:         N_p.UPDATESCHEDULE( )                 ▷ Update the schedule with resolved interval
26:     end if
27:     while N_parent is defined do              ▷ Translate local time to root time
28:         Let t_seek ← N_cur.LOCALTOPARENT(t_seek)
29:         Let N_cur ← N_parent
30:         Let N_parent ← N_parent.parent
31:     end while
32:     N_cur.WRAPPER.SEEK(t_seek)                ▷ seek the root node to the converted time
33: end function
```

The link activation procedure does two important tasks: *i)* attempt to resolve the element in case it is not resolved; *ii)* activate the element to the designated offset. The resolution procedure for linking is different from the constraint-based resolution procedure presented in previous Sections. Given that any element in the timegraph can be activated by linking, it is likely to occur that a deep node in a whole unresolved branch of the timegraph be activated. In this case, the element must traverse its ancestors until a resolved one if found. If none of them is resolved, then the root node gets forcefully resolved. Resolution in this case, performed by `localResolve`, involves only creating an internal timeline for the element, i.e., this function is strictly local and do not propagate operations to other nodes. After this, the traversal is backtracked, resolving the active interval of every visited node. Whether or not the resolution procedure is executed, the next procedure involves seeking the timegraph timing to the local time of the element. This involves converting the local time of the element to the root time. In this case, the operation `localToParent`, defined in Algorithm 3.13 in the context of timing manipulations, is applied. After the seek time has been converted to the root time, then the timegraph is sought to the converted time. This is done by calling the `seek` function of the presentation wrapper, which is the global presentation controller. This component will adjust the

current time of the reference clock and command a sampling update in the timegraph. It is worth noting that the `linkActivate` function includes a parameter, Δ, to inform an offset for the link activation. This parameter is intended to support the Media Fragments extension that was developed in ActiveTimesheets, which is discussed in the next Section.

An important side effect of link activation is that the presentation can be sought to the past. Consequently, the time function needs to be updated in order to take into account this change. This is necessary because the time function tracks the history of the duration of the element and, in case of a backward seek, this history must be changed. This adjustment is performed by the seek algorithm, called for each node during a depth-first traversal of the timegraph. This is done by the presentation wrapper (refer to line 32 of Algorithm 3.16), which is the main controller of the timegraph. This operation is detailed Algorithm 3.17.

Algorithm 3.17: Seek adjustment procedure

```
1:  function TIMEELEMENT.SEEK( )
2:      Let T be the parent time
3:      Let i_found ← −1
4:      Let i ← 0
5:      Let F be the time function
6:      Let s be the current playback speed
7:      while i < F.length and i_found < 0 do          ▷ check if the seek is backward
8:          if F_i[T] ≥ T then
9:              i_found ← i
10:         end if
11:         i ← i + 1
12:     end while
13:     if i_found > 0 then          ▷ the seek is backward, adjust the time function
14:         Let q ← F.length − i_found          ▷ how many pieces to remove
15:         REMOVEPIECES(F, i_found, q)
16:         if this is not the root node then
17:             this.t ← this.PARENTTOLOCAL( )
18:         end if
19:         Let F_last be the last piece of the time function
20:         if F_last is undefined or F_last[S] ≠ s then          ▷ current speed differs from last piece
21:             APPEND(F, (T, t, s))
22:         end if
23:         this.LOCALSCHEDULEUPDATE( )
24:     else if this is not the root node then          ▷ the seek is forward, just sample current time
25:         this.t ← this.PARENTTOLOCAL( )
26:     end if
27: end function
```

The seek procedure in Algorithm 3.17 treats two basic cases: *i)* if the seek is forward, then no update is required in the time function, so just re-sample the current local time; *ii)* if the seek is backward, then the time function needs to be updated, i.e., all the pieces from the sought moment forward must be eliminated from the history. First, the time function, whose pieces are ordered by parent time, is traversed to find the cutting point. Second, the function is split in the cutting point and, after that, the local schedule of the element is updated. In this case, the schedule computation is local because the seek procedure itself is already occurring during a depth-first timegraph traversal. The seek procedure of time containers is not represented in the algorithm, but the general idea is that composite elements call the seek procedure of their children before calling their own seek procedure, i.e., a post-order traversal. Finally, a last measure must be taken. After removing the pieces in case of a backward seek, if the current presentation speed differs from the speed of the (new) last piece, then a new piece must be added so that the presentation can continue in the same pace after seeking. This is done via the `append` call in line 21.

3.5.2 Linking with media fragments

In order to refer to an internal or external element in a document, an addressing mechanism is required. For internal reference, i.e. reference inside a document, identification of media elements via the `id` attribute might suffice this need, as long as elements can be accessed unequivocally. However, for referring to elements in multiple documents, a more elaborate addressing mechanism is required. For this purpose, two mechanisms stand out: fragment identifiers and media fragments.

Fragment identifiers are a specific addressing syntax of the URI standard, and consists on appending the identifier of a document element to the URI of the document. Consequently, a fragment identifier is essentially a fine-grained retrieval mechanism. In SMIL documents, fragment identifiers are supported analogously to HTML: identifiers are used for activating any named element inside a document. But in the case of SMIL, such activation means the same behavior as if the element had been traversed if it was a target anchor: the presentation is sought to the internal timeline of the element. A limitation of fragment identifiers is that it only allows addressing whole named elements inside the document. However, in certain situations, it is also desirable to address the document in other scopes and granularity.

As an example, if a user wants to skim a presentation by interactively seeking to random moments, this would require the creation, in advance, of named elements or target anchors for every moment being sought. Regarding a continuous temporal domain, the quantity of such anchors could get easily unmanageable as far as the desired resolution for random access increases. However, if the presentation environment provides a time-oriented addressing mechanism, then the presentation can be sought to a moment defined as a numeric value, which could be easily associated to an interactive user interface control. Another important issue is the

occurrence of identifier conflict in multi-document presentations. This is particularly important, for instance, in ActiveTimesheets, since in this language, as in other Timesheet-based languages, a multimedia presentation is composed by two documents, a spatial one and a temporal one. In case both documents define the same identifier for two elements, one in each document, then an identifier conflict occurs. Naturally, such conflicts could be solved during authoring, but as long as document can be edited while it is active, it is interesting to count on ways to minimize the likelihood of their occurrence.

Media fragments [W3C, 2012b] is an addressing mechanism that has potential to tackle limitations of fragment identifiers in important ways. In its basic version, it defines a syntax for selecting temporal and spatial fragments directly in the fragment identifier component of a URI. In its advanced, non-normative version, it allows selections in other dimensions, such as named elements, tracks, and chapters. In ActiveTimesheets, media fragments are integrated with linking functionality in order to provide more flexibility on browsing a multimedia presentation. In particular, in ActiveTimesheets, the following fragment dimensions are defined:

`http://(...)/presentation.html`**`#video1`**	frag id	(1)
`http://(...)/presentation.html`**`#id=video1`**	named	(2)
`http://(...)/presentation.html`**`#t=10`**	temporal	(3)
`http://(...)/presentation.html`**`#video1&t=10`**	frag id + temporal	(4)
`http://(...)/presentation.html`**`#id=video1&t=10`**	named + temporal	(5)

Figure 3.16: Examples of fragment addresses in an ActiveTimesheets presentation

- **fragment identifier dimension**: when faced with a fragment identifier (syntax (1) in Figure 3.16), the default behavior of a user agent is to scroll the spatial document to make the element visible. ActiveTimesheets extends this behavior to encompass also the temporal scope of the document. Given that both the temporal document and the spatial document can have elements identified via the `id` attribute, this can occur in the following ways: *i)* if the addressed element is spatial and has been binded to the temporal document, then the behavior is to scroll the document and seek the presentation to the start of the element internal timeline; *ii)* if the addressed element is spatial and not binded to the temporal document, then only scroll occurs; *iii)* if the addressed element is temporal, only the seeking operation occurs;

- **named dimension**: this syntax (example (2) in Figure 3.16) allows to disambiguate between elements in temporal and spatial documents, in case name conflicts occur. Therefore, when this syntax is used, only the temporal element identified by the name is activated;

- **temporal dimension**: allows addressing a document using offsets in the element internal timeline (syntax (3) in Figure 3.16). The effect of this syntax is a seek to the specified moment in the internal timeline of the root node;

- **composite dimension**: this syntax allows to combine the temporal dimension with other dimensions. Examples (4) and (5), in Figure 3.16, illustrates this syntax. The effect of such operation is the activation of the designated element, but with an offset applied to its internal timeline, which is informed via the temporal dimension.

The ActiveTimesheets engine includes a URL resolution mechanism that parses target URLs and converts media fragment specifications to an appropriate response in the linking mechanism. The resolution procedure and its integration to the linking mechanism is demonstrated in Algorithm 3.18.

Algorithm 3.18: Media fragments resolution procedure

```
1: Let u the document URL
2: Let r be the root of the timegraph
3: function ENFORCETIMEDIMENSION(N)
4:     Let Δ ← u.GETTIME( )
5:     N.LINKACTIVATE(Δ)
6: end function
7: function ENFORCEMEDIAFRAGMENTS(u)
8:     if u has a frag id dimension then
9:         Let n ← u.GETFRAGID( )
10:        Let N ← r.FIND(n)
11:        if N is defined then
12:            if u has time dimension then
13:                ENFORCETIMEDIMENSION(N)
14:            else
15:                N.LINKACTIVATE( )
16:            end if
17:        end if
18:    else if u has a named dimension then
19:        Let n ← u.GETNAMED( )
20:        Let N ← r.FIND(n)
21:        if N is defined then
22:            if u has a time dimension then
23:                ENFORCETIMEDIMENSION(N)
24:            else
25:                N.LINKACTIVATE( )
26:            end if
27:        end if
28:    else if u has a time dimension then
```

```
29:         r.ENFORCETIMEDIMENSION( )
30:     end if
31: end function
```

Algorithm 3.18 demonstrates the resolution of media fragments and their integration to linking. Each media fragment rule is checked and the corresponding node is activated accordingly. In order to activate a node, the `linkActivate` function, defined in the previous Section, is employed. The function is called as method of the element that should be activated. In the case of the fragment identifier and named dimensions, the element must be parsed from the fragment specification and retrieved from the corresponding data structure (spatial DOM or timegraph): this is achieved by the function `parseNode`. When the time dimension is defined in a composite syntax, the link is traversed with an offset, Δ, which is passed as a parameter to the function. In case the time dimension is used in isolation, a link to the root node is traversed with the Δ parameter.

3.6 Reuse features

Reuse is an important feature in declarative languages because it allows that authoring constructs to be re-purposed in different contexts. SMIL originally supports only reuse of media object content, by declaring multiple media objects with the same source URL. SMIL Timesheets contributes with more reuse possibilities, by allowing reuse of whole documents (via the combination of nested `timesheet` elements and the `src` attribute) and spatial structure (via CSS selectors). ActiveTimesheets extends the reuse features in these languages by allowing reuse of document constructs in more granularities: *i)* named elements from external documents; *ii)* named internal elements; *iii)* temporal fragments (clipping) of named internal or external elements. Realizing these contributions requires the development of several extensions, encompassing language syntax and timegraph construction patterns. This Section reports these extensions. First, Section 3.6.1 demonstrates a syntax for reusing individual media elements, originated either internally or externally to a base document. Then, Section 3.6.2 demonstrated how such reuse can be further extended by including fragmentation of the underlying reused element as an additional restriction.

3.6.1 Element reuse

Reusing individual elements of an external document has several applications in authoring. In enrichment activities, for instance, an external fragment can be reused to be annotated in a separate document. This allows a granular versioning of a document using only references to the original content. In order to explore these opportunities, ActiveTimesheets extends the reuse

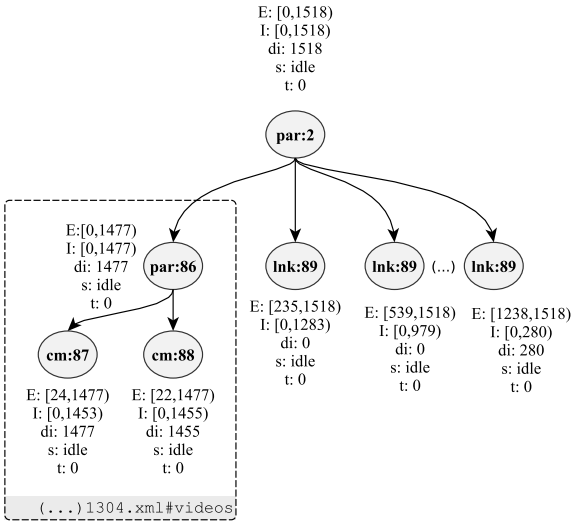
features present in SMIL and Timesheets to include reuse in finer granularity. In particular, any element of a document, provided that it is addressable, can be reused in another contexts. On defining such feature, specific syntax and timegraph construction patterns are required.

The `timesheet` element, in its original definition, allows the inclusion of a whole external document, via its `src` attribute. The expected behavior is that the temporal constructs in the external document be embedded in the local document. This is possible because the temporal semantics of the `timesheet` element is equivalent to that of `par` container, so that at the timegraph level they are almost indistinguishable. However, in certain situations, it is interesting to reuse, instead of whole document, only a fragment of this document. The expected behavior in this case, is that the temporal constructs of the fragment be seamless embedded in the local document.

The ActiveTimesheets engine extends the behavior of the timesheet element by allowing the reuse of fragments via its `src` attribute. This is achieved by allowing the use of fragment identifiers in this attribute, so that only the element identified by the fragment is reused. Figure 3.17a demonstrates an example reuse of elements from an external document.

```
1  <timesheet xmlns="http://www.w3.org/ns/SMIL">
2    <par id="bookmarks">
3     <timesheet
4      src="(...)/timesheet/1304.xml#videos" />
5     <area id="bk1" begin="235" />
6     <area id="bk2" begin="539" />
7     <area id="bk3" begin="861" />
8     (...)
9     <area id="bk3" begin="1238" />
10   </par>
11 </timesheet>
```

(a) Timesheet with a reused element.



(b) Resulting timegraph. Reused subgraph is emphasized.

Figure 3.17: Example of element reuse via the `src` attribute of the `timesheet` element

In Figure 3.17a, elements of the timesheet previously demonstrated in Listing 3.2 are reused. The general scenario modeled by this document is the bookmarking of external content by reference. But, instead of bookmarking the whole document content, only videos are included. Naturally, to take full advantage of this scenario, reuse of spatial elements should be provided, i.e. it would be interesting if the spatial fragment which contains the videos should also be (spatially) reused. This could be provided via some application-specific mechanism which is not discussed here, provided that the focus is to demonstrate temporal reuse.

First, the document reuses fragments of the external document, by addressing the document with a fragment identifier component. This is achieved by the `timesheet` element (lines 3-4). In particular, this document reuse a time container from the external document: the container identified as `#videos`. Additionally, the document includes a series of `area` elements representing bookmarks: these elements provide a "temporal index" to the specific points in the presentation, given that they can be activated via linking. The side effect of reusing an external element is, first, the retrieval and parsing of the element fragment and, only then, the inclusion of the corresponding subgraph in the presentation timegraph. Figure 3.17 demonstrates this effect.

In Figure 3.17b, the subgraphs which were included by reuse are highlighted. For all practical purposes, these subgraphs are considered as regular timegraph elements. Nodes corresponding to link anchors are prefixed with "*lnk*". In case some live editing operation occurs in the `src` attribute of any timesheet element, ActiveTimesheets will retrieve and parse the reused element again, replacing it in the timegraph.

The solution discussed here also allows reuse of internal elements. For that purpose, the only requirement is to use an internal fragment identifier, i.e. only the fragment identifier component without the prefix URL, in the `src` attribute of the `timesheet` element. In this case, the effect is the same as for external documents: the referenced element is parsed and added as a subgraph in the presentation timegraph.

In summary, reuse of external and internal elements allows some compelling use cases while enriching and extending content. Additionally, the possibility of reusing temporal structures potentially reduces duplication of declarative code, making the resulting document less verbose. One step further in reuse functionalities consists on reusing, instead of a whole element, only a fragment of this element. In the video bookmarking use case, for instance, this would allow importing only a reduced clip of a group of potentially long videos. The extensions that allow such improvement are discussed in the next Section.

3.6.2 Element fragment reuse

In certain situations it is interesting to reuse only a fragment of a element, for instance only a interval of a video. In SMIL, reuse of fragments of continuous media elements can be achieved via clipping attributes. Clipping consists on fragmenting a document element in the temporal scope. In SMIL, clipping is defined in the Media Object module and it is applicable only to continuous media elements. In practice, media object clipping is a kind of non-destructive, virtual, editing operation in the temporal scope: once the media content is edited, it can be reused in different situations. One possible use case of this functionality is, for example, the decomposition of a continuous media element into non-contiguous fragments, which can be used in a playlist. Another use case is stressing specific scenes in a long recording. All these cases can be achieved by authoring multiple media object declarations with the same source but different values for clipping attributes.

Therefore, clipping of a document element only makes sense as long as the element can be reused in another context. Given that SMIL originally only supports reuse of media content, clipping is available only for continuous media elements. However, in ActiveTimesheets any document element can be reused in other contexts and, as a consequence, clipping makes sense for any element that has an internal timeline, intrinsic or not. These include, besides continuous media elements, also temporal compositions, such as time containers and `item` elements.

A compelling advantage of an extended clipping model is the application to time containers, as it allows reuse of composite abstractions. For instance, if a time container aggregates a video track, an audio track, a subtitle and associated discrete annotations, clipping over this container would allow reuse of a fragment of the whole composition, framed to the specified clip. Clipping of time containers has been advocated before [W3C, 2003], but aside from statement of requirements, no solution has been presented for this problem as far as the literature is concerned.

In the original model, clipping is achieved via the attributes `clipBegin` and `clipEnd` which, combined, form the *clipped interval* of the element. The clipped interval, for all purposes, virtually replaces the internal timeline of the clipped element. As a consequence, the implicit duration of the element is affected by clipping. The clipping attributes can be used independently of each other, which creates several clipping layouts. Let $I = [I_e, I_b)$ be an element internal timeline, and c_b and c_e, the values for the `clipBegin` and `clipEnd` attributes, respectively. When clipping is applied to an element, the following properties apply:

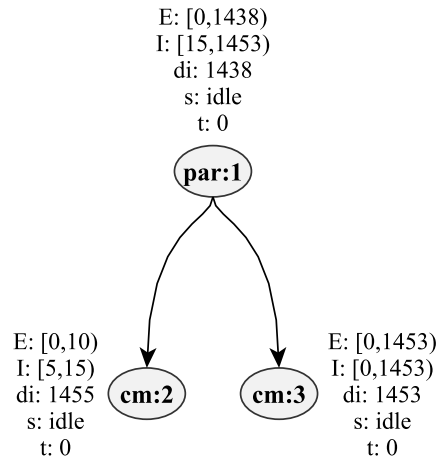
$$\begin{aligned} & I_b \leq c_b \leq c_e \leq I_e \\ & exists(c_b) \wedge \neg exists(c_e) \rightarrow I = [c_b, I_e) \\ & \neg exists(c_b) \wedge exists(c_e) \rightarrow I = [I_b, c_e) \\ & exists(c_b) \wedge exists(c_e) \rightarrow I = [c_b, c_e) \end{aligned} \tag{3.17}$$

Figure 3.18a presents a timesheet in which clipping attributes have been applied to `item` elements and to their time container as well. The item element `mevideo_82` has been clipped to the interval $[5, 15)$. Recalling from previous examples, the binding expression for this element selects a single video, which in Figure 3.18b is represented by `cm:2` in the timegraph. Notice that the element internal timeline, or clipped interval, is scaled to $[5, 15)$, thus the element implicit duration becomes $10s$. Notice that changing the *implicit* duration is a particular characteristic of the clipping attributes, given that `dur` and `end` attributes can change only explicit duration. As a consequence, based only on this information, the element external timeline, or active interval, is scheduled to the interval $[0, 10)$. The main effect of clipping is that the element, when activated, instead of starting the continuous media element at $0s$, will seek the element to $10s$ and when the continuous media element reaches $15s$ it is deactivated. This is also a distinguishing characteristic of the `clip-begin` attribute, given that no other element attribute allows to apply an activation offset in the intrinsic timeline of a continuous media element[12].

[12] An activation offset could also be achieved via linking offsets (refer to section 3.5). But in this case, the effect is not being enforced by an attribute of the activated element but via an external event.

```
1 <par clipBegin="15" id="videos">
2  <item clipBegin="5" clipBegin="15" id="mevideo_82" selec
3  <item begin="22" end="1477" id="mevideo_83"
4   select="#video_83"/>
5 </par>
```

(a) Timesheet



(b) Timegraph

Figure 3.18: Example of combined clipping in `item` and time container

Another case for clipped interval occurs in the container `videos`, represented by node `par:1` in the timegraph. This element is left-clipped at 15, consequently its internal timeline, which in its unclipped form would be $[0, 1453)$, becomes $[15, 1438)$. As the element is a composite structure, clipping this element means that all its temporal relationships are affected by the clipping. This means that the time container, when activated, will start with its internal timeline at $15s$. A consequence of this is that the element `cm:2` will never be activated, because the beginning of its active interval occurs in the past.

In summary, clipping affects the scheduling of an element in important ways, given that it manipulates the element internal timeline. As it does not bring any new structure to the timegraph node, but only alters the node internal timeline, scheduling algorithms as presented in the previous Sections already support these attributes. As demonstrated in Table 3.2, live editing of clipping attributes affects the element schedule and, therefore, should trigger schedule updates. This can also be achieved by the incremental update procedure demonstrated in Algorithm 3.15. As stated earlier, the usefulness of clipping temporal compositions is more appealing when an element is being reused, otherwise the same clipping result could be achieved by proper timing in the children elements.

3.7 The ActiveTimesheets engine

The ActiveTimesheets language has been implemented in a engine to be used in HTML5-compliant web browsers. The engine has been implemented in Javascript and includes all the language extensions discussed in this Chapter. Figure 3.19 presents an overview of the most important components of this implementation.

The most important component is `PresentationWrapper`, which controls the whole

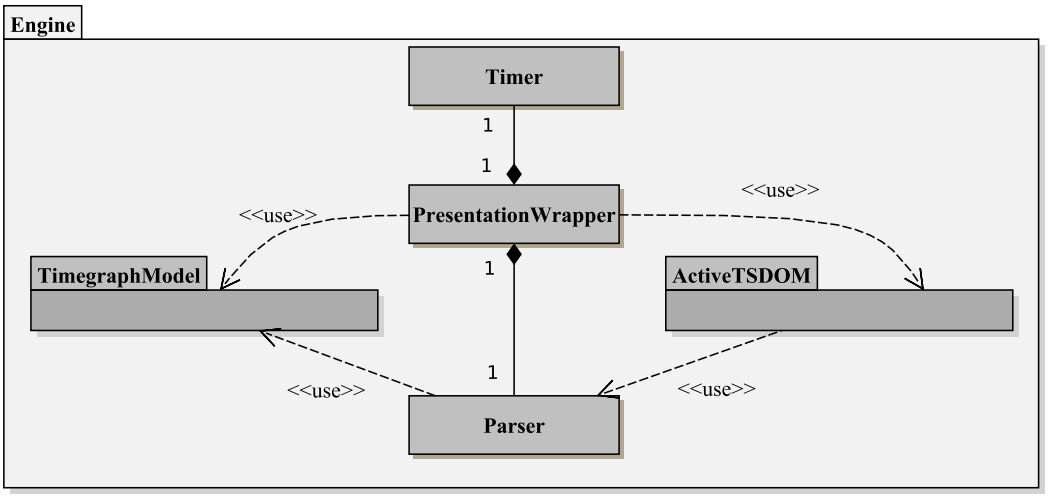


Figure 3.19: ActiveTimesheets engine simplified package diagram

lifecycle of the presentation, i.e., it manages the engine state machine discussed in Figure 3.8. This component is the main entry point of the engine, providing methods for most of the exposed functionalities. In particular, the wrapper tackles the following issues:

- **presentation setup**: the wrapper, once started, commands the parsing of the DOM (using the *Parser* component), building the timegraph as a result. For this purpose it takes advantage of the the *TimegraphModel* and *ActiveTSDOM* packages, which implement the ActiveTimesheets timegraph model and DOM model, respectively. The general structure of these packages has been presented previously in this Chapter;
- **scheduling and sampling**: it triggers the initial scheduling of the timegraph. In addition, it controls the reference timer and responds to every clock tick by sampling the timegraph. For that purpose, the wrapper includes a reference to the root node of the global presentation timegraph and aggregates the reference timer;
- **playback control**: it manages the playback state machine of the presentation, providing methods, for instance, to play, stop and pause the presentation. Additionally, it manages buffering conditions, by making sure the presentation is halted in case the playback of some media element is stalled;
- **DOM access**: it provides methods for accessing the ActiveTimesheets DOM in order to perform live editing operations.

Listing 3.3: Example startup and live editing code for the ActiveTimesheets engine

```
1  <html>
2  (...)
3  <head>
4   (...)
5   <script type="text/javascript" src="(...)activetimesheets.js"></script>
6   <link href="(...)/timesheet/1304.xml" rel="timesheet" type="application/smil+xml">
7   (...)
8   <script type="text/javascript">
9    function bookmarkHandler(presentation) {
10    var dom = presentation.getExternalDOM();
11    var ts = dom.getTimesheetElement();
12    var root = ts.find('#presentation')[0];
13    var v_node = dom.createElement('area', {
14     begin: root.getTime()
15    });
16
17    root.append(v_node);
18   };
19
20   $(document).ready(function () {
21    $(ACTIVETIMESHEETS.engine).on('presentation_ready', function (ev) {
22     var presentation = ACTIVETIMESHEETS.engine.getPresentation();
23
24     presentation.play();
25     $('#bookmark').click(function () {
26      bookmarkHandler(presentation)
27     });
28    });
29    ACTIVETIMESHEETS.engine.start();
30   });
31  </script>
32 </head>
33 (...)
34 </body>
35 </html>
```

Listing 3.3 demonstrates the use of the ActiveTimesheets engine in an example document. This example is an excerpt of a simple application to bookmark moments of a presentation while it is being watched. In the HTML document of the example, the ActiveTimesheets engine is included in the `head` block (line 5), alongside the temporal document used in the application (line 6). In this case, the temporal and spatial documents are assumed to be the ones used in sample recorded talk presented in Listing 3.1. The engine should be started via scripting. For that purpose, the Javascript code in lines 20-30 is triggered once the spatial document is completely loaded. The engine is started via its method `start()` (line 29) and, once the presentation has been built, i.e. parsing the document has finished and the timegraph is ready, the engine triggers an event `presentation_ready`, which is captured by the handler code in lines 21-28. This code will retrieve the presentation wrapper and play it. Additionally, it will bind an interactive "click" event to the element `#bookmark` of the spatial document (lines 25-27): this element can be a button, for example, that will be clicked to bookmark the current playback time of the presentation.

Live editing of the temporal document occurs every time the `#bookmark` button in clicked.

When this event happens, the function `bookmarkHandler` (lines 9-15) edits the temporal DOM. This is achieved by first retrieving the DOM and finding the node where the bookmark node will be inserted. Then a new `area` element is created, using the proper method of the ActiveTimesheets extended DOM. The new node take the current presentation time as its `begin` attribute. After that, the element is appended to the selected parent node. As a result of this operation, the timegraph will have its schedule updated. Additionally, the newly created link can be used right away to seek to the bookmarked moment of the session.

3.8 Final remarks

This Chapter presented the ActiveTimesheets language for Web-based active multimedia documents. Based on the SMIL language, ActiveTimesheets provides a number of important extensions, such as live editing, extended linking, media fragments support and reuse of elements and fragments. These extensions have been demonstrated taking advantage of an improved formalization of the timegraph model. Finally, practical use of the language has been realized in a Javascript-based presentation engine. All these contributions have several implications to the current state of the art.

SMIL has been advocated for several years, but its adoption has not gained much traction. Multimedia on the Web, until not so long ago, has been characterized by integration of self-contained proprietary technologies embedded in user agents via encapsulated execution environments (e.g. Flash). With the advent of native execution of multimedia streams in user agents, provided by HTML5, this situation has been changing, with more multimedia-oriented applications revealing requirements for advancing these technologies. Synchronized multimedia presentations are still a gap as far as native Web technologies are concerned, which has lead to developments of incremental solutions (e.g. Mozilla Popcorn and static Javascript-based engines) in attempts to bridge this gap. The extensions reported in this Chapter have great potential to make SMIL gain traction in Web development, as long as it exposes the SMIL model combined with effective adaptive behaviors, which is a constant pattern nowadays in the context of rich Internet applications.

In document languages that support scripting, it is expected that scripts can modify the structure of the document. This is true for most languages on the Web technology stack, such as HTML and SVG, just to name a few. Thus, realizing live editing in SMIL documents is an important contribution to subsidize scripting in this language. The SMIL recommendation does not prescribe any scripting functionality; even the SMIL Timesheets language, that is supposed to be used in a heavily scriptable host language such as HTML, does not provide means for scripting a temporal document. This contributes to make SMIL lag behind other multimedia document languages, such as NCL, that already support scripting.

The ActiveTimesheets language takes only a subset of the SMIL language in order to demonstrate live editing. Even though it does not encompass the complete language, it establishes

the concrete basis of a live editing model that can be further extended: a better formalization of the timegraph model, an extended time function for timing manipulations, and a detailed account of the most important timegraph algorithms, such as scheduling, sampling and resolution. It is advocated here that these models and algorithms, as presented in this dissertation, can be extended to encompass other SMIL features in a modular approach. Further, a modular approach is what the SMIL recommendation prescribes by means of profiles, so the contributions of this Chapter are aligned with this point of view.

From the standpoint of reuse features, ActiveTimesheets also brings important extensions to SMIL. Reusing arbitrary elements of document, via internal or external reference, has several applications in authoring. On one hand, reuse enables unobtrusive extension of external content, as long as enrichments are applied in a decoupled manner in a document fragment. Additionally, reusing elements in fine granularity potentially reduces the verbosity of a document, as long as it inhibits the creation of verbose, duplicate, declarative code. Another important extension, this time compared with similar engines, is the inclusion of linking functionality, which takes full advantage of SMIL as a hypermedia language. The ActiveTimesheets model contributes also by making a smooth integration between spatial linking in HTML and temporal linking in SMIL.

Differently from other multimedia languages, which presume a "push" (server-initiated requests) transmission model, in this Chapter no solution has been proposed for server-side editing. This is mostly because the solutions proposed in this Chapter have been mostly applied over a Web-based architecture which, even if it follows essentially a "pull" (client-initiated requests) transmission model, it also counts on various methods to provide "push" behavior whenever it is needed. For instance, a common model for enabling server commands is via polling, in which a script periodically queries a server which, depending on the conditions, may order an editing operation on the client. Besides this script-oriented pattern, it is common to take advantage of persistent HTTP connections combined with a publish-subscribe model: consequently, the server can asynchronously send update instructions to the client. In summary, current Web design patterns already provide proper methods for server-side commands, being up to specific applications to be instrumented with these patterns in case they are deemed necessary.

Another important issue that is not tackled by ActiveTimesheets is that of spatio-temporal consistency. This problem consists on keeping the temporal layout updated if a change in the spatial document occurs. This is particularly important because timesheets are binded to the spatial document via selection queries: if the spatial document is modified (nodes are added or removed) in a way that affects the results of these queries, then the temporal layout must be updated as well. A solution for this problem would require some change detection mechanism and an efficient result sampling procedure to keep the bindings updated. This also requires deeper investigation regarding the best alternatives and their trade-offs.

The expression power of the SMIL language, as in other similar languages, can lead, in certain situations, to the specification of inconsistent presentations. Such inconsistencies generally occur due to use of conflicting synchronization patterns, for instance constraint-based loops that might

lead the document to a never-ending execution. These problems are particularly important in the context of live editing, in which the document can immediately enter in an inconsistent state. Many solutions have been proposed to verification of SMIL documents, most of them adopting formalisms different from the timegraph model. Besides the formalism mismatch when compared to ActiveTimesheets, most of these solutions incur a high computational cost, which makes them more suitable to offline verification, i.e., verification before the document is active. In summary, given that verification of the document consistency is a desirable feature during live editing, a fruitful theme for future research consists on the investigation of timegraph-based methods for efficient online verification.

An issue that has been discussed before in multimedia synchronization languages is that of elastic time computation. This technique consists on slowing down or speeding up the streaming or playback rates of media elements to compensate for eventual synchronization slips caused by environmental conditions (e.g. network and processing load). Currently, the ActiveTimesheets engine does not support such a synchronization control. The solution currently adopted consists on "suspending" the presentation until all media buffers are sufficiently full, a situation which can impair the quality of experience. It is advocated here that the dynamic piecewise-defined time function proposed in this Chapter can be a first direction towards a solution to this problem, given that it allows to adapt the timing of the presentation due to speed changes. However, deeper investigation is still necessary in order to determine all the requirements and trade-offs of such a solution.

Chapter 4

IAMmDocT: Interactors Algebra for Multimedia Document Transformations

Multimedia production is an elaborate activity composed of multiple information management and transformation tasks that support an underlying creative goal. Examples of these activities are structuring, organization, modification and versioning of media elements, all of which depend on the maintenance of supporting documentation and metadata. In professional productions, which can count on proper human and material resources, such documentation is maintained by the production crew, being important to secure a high quality in the final content. In less resourceful configurations, such as amateur-oriented productions, at least reasonable quality standards are desirable in most cases, however the perceived difficulty in managing and transforming content can inhibit amateurs on producing content with acceptable quality. Therefore, a feasible solution for this problem should tackle the following requirements:

1) how to generate documentation of the multimedia production process with low user effort;

2) how to represent and manage such documentation along the process;

3) how to take advantage of such documentation to author multimedia content.

Naturally, in requirement *(1)* the term "documentation" entails an ample group of resources, from high level information, such as scripts, storyboards, etc., to low level information, such as scene discrimination and production logging metadata. There is no universal solution to completely automatize the generation of such information, so as high quality, complete documentation will require significant human effort to be created. However, an approximate, manageable solution for this problem consists on identifying a reduced group of information that can be captured with low effort whereas still being useful, if not for all authoring purposes, at least for a controlled set of content manipulation activities.

Previous related research, e.g. Interactors and Watch and Comment (WaC) — both discussed in more details in Section 2.7 — has tackled requirement *(1)* by identifying opportunities to collect

annotations with low user effort along the production in specific domains. Interactors has focused on documenting interactions that occur in recordings of small group activities, for instance slide changes, turn taking, live annotations, chat messages, ink interactions and so on. The utility of these annotations, i.e., how requirement *(3)* has been tackled, have been demonstrated for automatic authoring (e.g. generating an interactive session browser) [Vega-Oliveros et al., 2010a], manual authoring (e.g. filtering and expanding the granularity of frame-based slide presentations) [Cattelan et al., 2008b] and for browsing (e.g. non-linear navigation in a session indexed by the annotations) [Vega-Oliveros et al., 2011b]. WaC has tackled requirement *(1)* by documenting user commenting behaviors via tracking of explicit annotations. Regarding requirement *(3)*, the utility of these annotations has been demonstrated in the implicit authoring of interactive multimedia documents as a consequence of the annotation process [Cattelan et al., 2008a]. What all these previous research examples demonstrate is that through the various stages of a multimedia production process (from capture to distribution) there are opportunities to collect, with low user effort, information that can document multimedia content and that can be used for authoring purposes. Much of this information has a low-level semantics, but their ease of capture and perceived utility for certain situations can compensate the general difficulty of generating, for instance, automatic annotations with high-level semantics.

Regarding requirement *(2)*, Interactors research has concentrated on the definition of a structured XML-based vocabulary which documented interaction events by classifying them in specific groups and identifying their temporal scope [Vega-Oliveros et al., 2010a]. More elaborate models have been designed in WaC research, tackling various scenarios, for instance, documenting TV interactions [Teixeira et al., 2010a], or documenting video annotation interactions [Fagá et al., 2009]. A commonality of WaC interaction representation models is their reliance on the 5W1H (What, Where, When, Who, Why and How) abstraction framework, recurrently used to model context information [Dey, 2001]. Based on this framework, such models encompassed the multiple facets of an interaction event, such as details about the user, time, location and interaction devices. The main application of instances of the models consisted on encoding annotations directly in the documents.

The contributions reported in this Chapter tackle new requirements derived from previous research, in order to bring more flexibility and expression power to conceptual models and authoring approaches. Regarding requirement *(1)* the basic assumptions are the same, i.e., to take advantage of logged interactions in small group activities and user behavior while accessing multimedia content. Improvements are focused on requirements *(2)* and *(3)*, tackling the following issues: *i)* a combination of manual and automatic authoring approaches based on formally defined operators; *ii)* composition operators which are applicable to both browsing and authoring/transformation; *iii)* authoring flexibility, by allowing fine-grained inclusion of which annotations are used to compose document versions. In order to realize these improvements, contributions of this Chapter are distributed in the following themes:

- **Documentation.** Small group activities such as meetings, lectures, webconferences

and interviews are commonly recorded so that the activity can be revisited later by the participants or shared with someone who was not present in the live experience. In the case of multimedia recordings of these activities, it is recurrent the capture of a session composed of temporally synchronized media streams (e.g. audio, video, slides, screen sharing, etc.), aggregated in a linear video or in a declarative multimedia presentation. In the context of this dissertation, multimedia documents generated from capture and access sessions are the focus for enrichment and editing operations, i.e., they serve as a basis for generating new, improved versions. Therefore, an important issue is how to enrich these documents with useful annotations to subsidize authoring activities.

The I+WaC-IE (Interactors+WaC Interaction Events) conceptual model fulfill this goal, by integrating the interaction abstractions relevant to both Interactors and WaC approaches. This is achieved by building upon a modeling approach based on multimedia events, specializing it to model interaction events. The model adopts a multi-dimensional approach to characterize interaction events and annotations in an integrated manner, encompassing interaction details, temporal scope, spatial scope, and document scope. Beyond integration of both authoring approaches, I+WaC-IE also includes more accurate and fine-grained characterizations of the spatial and temporal dimensions, which are particularly useful not only for transforming multimedia documents, but also for tracking the provenance of annotations and document versions. Besides documentation of the production process, the main purpose of I+WaC-IE is to serve as a schema for the transformation operators.

- **Browsing.** As part of a production process, a user revisits the recording in order to identify and select which fragments will enter the final piece. This can be a daunting task that demands considerable time, in some cases several times the duration of the recording. For that reason, in some situations a user may not be interested in watching all the content linearly, but only on reviewing or skimming fragments of interest. Additionally, reviewing low level interaction events, such as slide changes and occurrence of voice utterances might not suffice to satisfy the user information needs. In order to improve the level of abstraction of the events, the literature suggests the opportunity of combining low level interaction events to generate higher level interaction events by means of composition relationships. For instance, hypotheses regarding the occurrence of these patterns have been investigated in the analysis of voice utterances to derive dialogue acts [Yu et al., 2012] and the discrimination among brainstorming and decision-making discussions [Jayagopi et al., 2012]. The analysis of composite interaction patterns have also been investigated in the context of recordings from doctor-patient interviews [Koyama et al., 2010]. These research results suggest the feasibility of combining low-level interaction cues in order to make sense of specific events in small group activities, opening up possibilities for better session browsing mechanisms.

  For that purpose, part of the IAMmDocT (Interactors Algebra for Multimedia Docu-

ment Transformations) operators have been defined to compute temporal composition relationships between groups of annotations. In this case, the objective here is not to propose automatic pattern derivation algorithms, but on providing operations to combine annotations, a feature that can be used, for instance, in dynamic visualization of composition patterns. In this Chapter the foundation for temporal compositions are defined and then, in Chapter 5, a dynamic visualization scheme is realized in the I+WaC Editor as an application of these definitions.

- **Authoring.** Once a recording has been properly analyzed via its associated annotations, activities that remain to be done is to make editing decisions and to apply them over the content. This could be done, for instance, in any non-linear multimedia editing system involving elaborate authoring metaphors, such as timelines, cards, or other techniques discussed in Section 2.6. However, a compelling alternative investigated here is to support amateurs in performing focused, small, editing and enrichment tasks, such as highlighting moments, overlaying additional content, applying comments and extracting fragments, just to name a few. Such operations have potential to fulfill, if not all, at least a considerable range of use cases toward improving captured content.

  The IAMmDocT operators, which are formally defined to subsidize authoring operations, directly contribute with this issue. The main purpose of the operators is to generate new versions of a multimedia document via manipulation of annotations, which are represented as instances of I+WaC-IE model. Consequently, the operators apply transformations in a temporal document taking as a reference the temporal scope of annotations. The algebra includes operators for joining fragments in the temporal scope, constraining the media elements that will be included in a new version, and extracting fragments of interest. This Chapter defines the formal semantics and algorithms of these operations, whereas their mapping to concrete interaction techniques in an authoring tool is discussed in Chapter 5.

In summary, the contributions of this Chapter extend Interactors and WaC in important and novel ways, by providing integration, flexibility and expression power to authoring tasks. The remaining of this Chapter is organized as follows: Section 4.1 discusses related work. Section 4.2 details the I+WaC-IE conceptual model. Section 4.3 provides an overview of the IAMmDocT semantics. Section 4.4 presents the Interactors operators. Section 4.5 discuss the temporal join operators. Section 4.6 discuss the transformation operators, including the effect operators (Section 4.6.1), projection operator (Section 4.6.2) and slicing operator (Section 4.6.3). Finally, Section 4.7 discuss the contributions and implications of the research, concluding this Chapter.

4.1 Related work

4.1.1 Documentation of interaction events

Events are defined as "real-world occurrences that unfold over space and time" [Xie et al., 2008]. In the multimedia representation literature, the concept of "event" has been recurrently advocated as a powerful abstraction to represent semantics in multimedia content. For that purpose, much research has been concentrated on modeling, detecting and querying multimedia information based on event-centric annotations. An event model, in particular, encompasses: *i)* the conceptual schema over which detected events will be stored; and *ii)* the common base over which the events will be queried. Respected the importance of event models for multimedia abstraction, in general, and for the contributions of this dissertation, in particular, the following discussion focus on related work that deal with the problem of modeling multimedia events.

The event model proposed by Westermann and Jain [Westermann and Jain, 2007] presents many aspects for characterizing an event: i) a *spatial* aspect and a *temporal* aspect, which document how the event occurs in the physical space; *ii)* a *structural* aspect, which document the composition of multiple events; *iii)* a *causal* aspect, which denotes the chain of relationships between events; *iv)* a *informational* aspect, which specifies entities comprised in a event, such as participants, roles and the types of interaction among them; and *iv)* a *experiential* aspect, which refers to related media that enrich and document the event, such as manually or automatically assigned (multimodal) annotations. Scherp et al. [2010] build upon Westermann and Jain's model while extending it with the *interpretation* aspect, which is intended for the association of contextual information related to the event.

The model proposed by Xie et al. [2008] shares many similarities with conceptual models for context-aware computing [Dey, 2001]. The model defines the 5W1H scheme that is structured on six facets: *i) when* denotes temporal relationships among events; *ii) where* denotes spatial relationships between events; *iii) who* refers to the identity of participants of the event; *iv) what* refers to the activities that take place in the event; *v) why* characterizes the context of the events; and *iv) how* expresses the (causal) dynamics of events. By adopting a more simplistic approach, van Hage et al. [2011] characterize events in three main classes: actor, place and time, which refers to participants, spatial and temporal relationships, respectively. Nagargadde et al. [2007] formally define a framework for modeling real-world events whose conceptual model encompasses time, space and label — where a label is understood as a semantic representation of an event.

The strategy adopted in these models, i.e. to abstract media-reported occurrences in multiple dimensions, facets or aspects, is very useful to document occurrences in a production process. For instance, the types of interaction events that occur in small group activities can be categorized in multiple dimensions as well. However, directly applying these models for documenting interaction events present some limitations. This is because, apart from the pervasive spatio-temporal dimensions, the models can be distinguished based mainly on (refer to Table 4.1 for

Table 4.1: Comparison of various dimensions across surveyed multimedia event models.

Model	**Composition**	**Spatial-temporality**	**Human experience**
Westermann and Jain [2007]	structural causal	temporal spatial	informational experiential
Scherp et al. [2010]	structural	temporal spatial	experiential participation interpretation
van Hage et al. [2011]	-	place time	actor
Xie et al. [2008]	-	when where	who what why how
Nagargadde et al. [2007]	-	space time	label

a comparison): *i)* the coverage of event compositions (in the structural or causal dimensions) and; *ii)* the entities that report the human experience in the event. About human experience, the great variability of modeling primitives across the approaches suggests a low level of agreement on the kinds of information that are relevant to characterize it. This is apparently because there is great heterogeneity in the semantics of activities across different domains. Therefore, these models might prove exceedingly generic to model interaction events with sufficient accuracy. About composition, it is described mostly on structure and causality between events, provided that, in the domain being analyzed, events can be organized and ordered according to these compositions. However, once again these models might prove to be too generic, as long as none of them provide constructs for temporal and spatial composition of events, for instance.

In this dissertation it is advocated that the several types of interactions that can be captured in multimedia productions constitute a valuable dimension for documenting multimedia information. Based on that assumption, the I+WaC-IE conceptual model, defined in this Chapter, is largely based on the multidimensional approach that is recurrent in multimedia event models whereas it provides several adaptations: *i)* the model is specialized to interaction events, encompassing dimensions to characterize the documentation of an interaction event (e.g. space, time, document and interaction); and *ii)* the space and time dimensions are extended to separate its different scopes (e.g. valid space/time vs. presentation space/time).

4.1.2 Multimedia operators

The problems of modeling and manipulation of multimedia information have been investigated before under two core topics: presentation specification and query result generation. In presentation specification approaches, operators are generally used as a formalism to represent and validate the spatio-temporal layout of a document [Presti et al., 2002]. Query result gen-

eration approaches, on the other hand, generally attempt to model the content of multimedia databases, in most cases composed of videos, and define operators to query these databases, generating results in the form of videos or presentations.

Specification-based approaches generally do not deal with the problem of transforming a multimedia presentation, instead they focus on defining a low-level formal language for authoring multimedia presentations. A classical example is the TAO (Temporal Algebraic operators) [Presti et al., 2002] approach, which consists on a group of decomposable operators that, via algebraic expressions, represent the temporal layout of a document. Another recurrent example is consistency checking in multimedia documents, i.e. verifying if the spatio-temporal layout of a document is consistent. Such verification can be online, i.e. incrementally, while the document is being authored (e.g. [Bouyakoub and Belkhir, 2011]), or offline, i.e. after the document has been authored (e.g. [Gaggi and Bossi, 2011]). Operator-based models have been used in this problem, for instance, by Elias et al. [2006]. In summary, an operator-based specification would have the same function as other formal models surveyed before in this dissertation (refer to Section 2.6), such as graphs, petri nets, objects, and so on. Consequently, they provide only low level formalisms to be used for any document representation purposes, and do not directly tackle the problem of transforming multimedia documents.

Proposition of operators for generating visual results from multimedia repositories is also a recurrent issue. One of the seminal approaches, the Video Algebra [Weiss et al., 1995] system, defined operators to generate presentations from video repositories. Assuming that videos in a repository have intervals annotated with metadata, the operators are used to retrieve fragments and merge them in a composition. Consequently, the result of a query is a presentation, specified with simple temporal composition operators. A related contribution is given by the operators defined by Pissinou et al. [2001], whose query results generate VRML documents out of annotated video fragments. The model adopted by these systems is characterized by the query-based composition of presentations from simple media elements, so as they do not support transformation in multimedia documents, i.e. having a multimedia document as both input and output to an algebraic expression.

Transformation of multimedia documents has been tackled by Celentano et al. [2004], which defined a group of operators for retrieval of information from document repositories. The operators allow the extraction of fragments from a multimedia presentation in a way that their spatio-temporal context is preserved to a certain degree. Beyond extracting fragments, however, no other editing or enrichment operation is supported. Another relevant contribution is the algebra proposed by Pattanasri et al. [2013]: by adopting a simplified multimedia document model, composed of a single video and a set of slides, the operators allow to fragment multimedia documents and assemble the fragments into new documents. However, besides fragmentation and composition, no other editing or enrichment operation is defined. A recurrent characteristic of these approaches is the lack of enrichment operations: this is because they have been defined for retrieval purposes, being transformation in these cases just a means to retrieve information

granules from a document.

IAMMDocT, the algebra proposed in this Chapter, presents several advantages when compared to related work. First, it has been defined with a focus on authoring issues, consequently the operators allow both editing and enrichment via transformations, instead only transformations (such as extraction of fragments). Second, the enrichment operations take advantage of live editing features of multimedia documents, consequently, instead of generating whole new versions upon every operation, incremental versions of the document are created. This has several consequences for interactive authoring applications.

4.2 I+WaC-IE conceptual model

I+WaC-IE (Interactors+WaC Interaction Events[1]) is a data model that is used as a scheme for all the operators defined in this Chapter. This conceptual model abstracts annotations created in a production process as *interaction events* that are collected either automatically or manually through the stages of multimedia production. Being annotations of occurrences, interaction events can be classified by most of the categories discussed in Section 2.3. Therefore, they can be implicit or explicit, be abstracted as content or as metadata, be unstructured or structured, and so on. Detection of interaction events can occur via virtual or physical sensors (e.g. by instrumenting a room or an application) or via content analysis (e.g. by analyzing audio and video signals). Table 4.2 lists some example interaction events whose modeling have been experimented in I+WaC-IE.

As Table 4.2 demonstrates, most of those interaction events are abstracted from application logs, which must be instrumented to report interactions. This is the case of capture-time applications, such as a whiteboard component or a chat component of a webconferencing environment, or individual notes taken by participants using distributed devices (e.g. tablets or smartphones). Additionally, interaction with external physical devices can also be documented, such as interaction with electronic pens over paper documents and toggling audio capture in a microphone. Moreover, access-time annotations, such as comments, bookmarks, ratings, and playback behavior, can be also be documented, provided that the base applications are instrumented to do so. Signal analysis of the content streams can also lead to innumerous types of events, generally generated at post-production. In the Table these are represented, for instance, by the detection of moments of voice activity or the absence of it.

A common characteristic of interaction events is that they are represented by groups of attributes that describe them. For instance, all interaction events will have a temporal attribute to contextualize them in time. Additionally, some attributes may have spatial attributes, for instance ink-based operators can be contextualized in a board region but also, as being an annotation, in

[1] In the context of the I+WaC-IE model, "event" should not be confused with the notion of events in multimedia documents, i.e., with a constraint-based synchronization concept. Here, "event" denotes an occurrence in the real world, whereas "interaction event" denotes an interaction occurrence between multiple agents (e.g. people, objects, etc.) in the real world.

Table 4.2: Examples of interaction events captured during multimedia production

Event	Origin	Description
strokeDrawn	whiteboard log	a stroke has been drawn in a whiteboard
slideChanged	whiteboard log	a switch of slides occurred in a whiteboard
slideAdded	whiteboard log	a slide has been included in a whiteboard
slideRemoved	whiteboard log	a slide has been removed in a whiteboard
slideHighlighted	whiteboard log	a fragment of a slide has been emphasized (e.g. via a virtual laser pointer)
boardChanged	whiteboard log	an additional board has been activated
textEntered	chat log	a text comment has been entered in a whiteboard
attributeChanged	chat log	some stroke, slide or comment had an attribute modified
messageSent	chat log	a text message has been transmitted
commentCreated	annotation log	a comment over a slide or a media element has been created
commentEdited	annotation log	an existing comment has been modified
bookmarkCreated	annotation log	a fragment of a media element has been spatio-temporally discriminated
ratingEntered	annotation log	a qualified spatio-temporal discrimination has been associated to the presentation
presentationPaused	player log	playback suspended
presentationPlaying	player log	playback resumed
presentationScrubbed	player log	a non-linear access in the presentation has been triggered
micMuted	capture log	a microphone has been explicitly muted
micUnmuted	capture log	a microphone has been explicitly unmuted
silenceBegin	audio analysis	a period of no sound has started
conversation	audio analysis	several participants are talking at the same time
participantSpoke	audio analysis	voice activity of a participant detected
voiceIncreased	audio analysis	above average voice volume increase detected

the geographical location where it was captured. Still in the case of ink activity, style attributes might be relevant to document as well, as long as such attributes can carry semantic connotation (for instance, using a bigger font case for emphasis).

The I+WaC-IE conceptual model, depicted in Figure 4.1, abstracts such interaction events and their attributes. This model characterizes an interaction event as a multidimensional concept, each dimension encompassing an aspect of the attributes of an event. Therefore, an interaction event is a central concept of the model, around which all the other concepts are structured. In addition to dimensions, the model also include concepts for managing interaction events as annotations along the production process. The central concept in this context is *annotation*, which associates an interaction event to a document, in coarse granularity, or to a media element of a document, in fine granularity. Such association is performed by a special type of operation, defined later in this Chapter, that binds groups of interaction events to fragments of documents. An annotation is associated to a document as a media element: as such, it participates in the

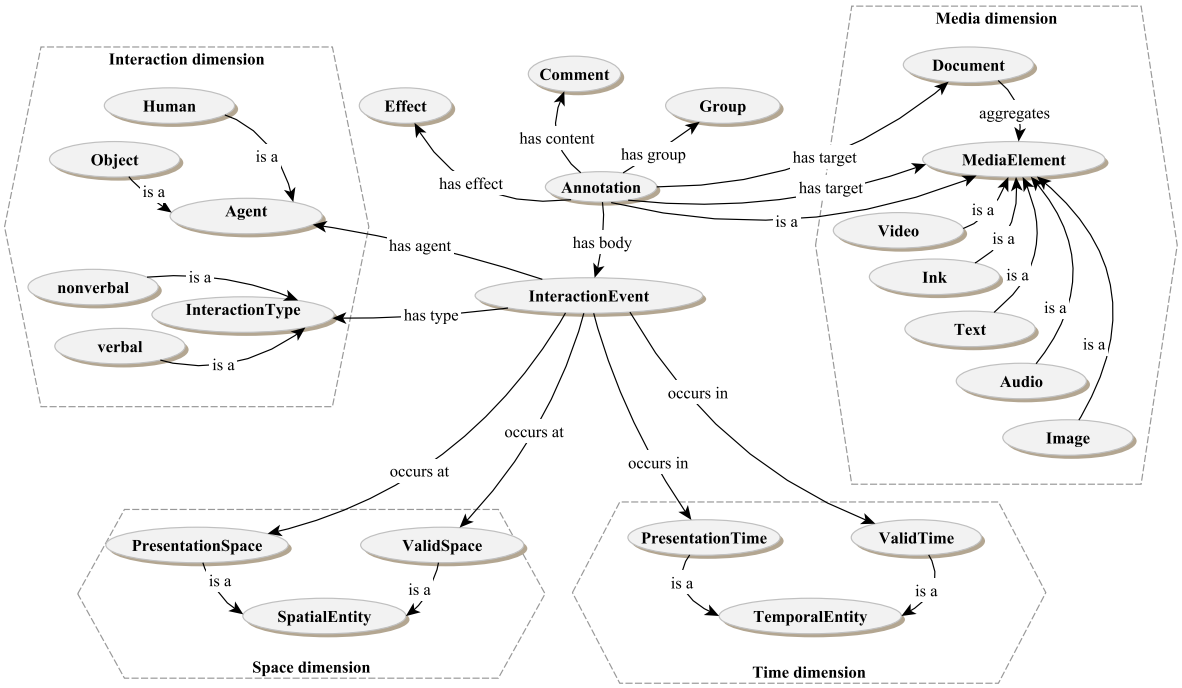


Figure 4.1: Overview of the I+WaC conceptual model

presentation and can be annotated as well (i.e. the model allows “annotations of annotations”), like with a regular media element. Besides associating documents and interaction events, an annotation also comprises the following concepts:

- the *group* concept, which classifies the annotation event according to a common set of attributes shared by other events. Values for this concept are, for instance, `changeOnAuthor`, `filterByAttribute`, and so on. In fact, the group concept is applied by an annotation operator, a mechanism that filters groups of events based on their attributes. The details of this mechanism will be discussed in Section 4.4;

- the *comment* concept, which associates additional information to the annotation. Annotations can act as metadata or as content. Metadata-oriented annotations are those that just spatio-temporally record an interaction event to the associated document, consequently they do not have any content associated with them. Content-oriented annotations, on the other hand, are those that, besides recording occurrences, complement such records using structured or unstructured content. Examples of content-oriented annotations are intentional annotations (e.g. comments), which by themselves are interaction events (e.g. user-media interactions), but that generate, as a result of the interaction, a content (e.g. a text fragment, an audio stream, a ink stroke, and so on). The comment concept references an arbitrary media resource, such as audio, video, text, ink and so on (i.e. multimodal comments). As an annotation can participate as a media element in the presentation, the resource identified by this media element represent the visual or aural embodiment of the annotation;

- the *effect* concept, which prescribe an interactive temporal effect of an annotation over the presentation. Being annotations collected here with authoring purposes, a direct way to do so is to take advantage of the spatio-temporal grounding of the annotation and their content in the document in order to perform editing operations. These editing operations are virtual, i.e., they do not actually modify the document, but only are realized via document patterns, i.e., they can be regarded as "virtual", non-destructive operations. The default effect is no effect, i.e., an annotation participates in the presentation as a regular media element, in a passive manner. However, if other effect values are assigned, the temporal flow of the document can be manipulated. Possible values are pause (when activated, the annotation will pause the presentation), loop (when activated, the temporal scope of the annotation is indefinitely repeated in the timeline of the presentation) and skip (when activated, the temporal scope of the annotation is ignored, causing a jump in the presentation timeline). The effect operators, and their realization in concrete multimedia documents, are discussed in Section 4.6.1.

The remaining concepts of the model, i.e., those related to the space, time, media and interaction dimensions deserve separate Sections for in-depth discussion.

Media dimension

The media dimension encompasses the several media aggregated in a document, the document itself and the spatial and temporal relationships between them. The I+WaC-IE model does not constrain a document format, delegating this issue as a design decision to be taken when instantiating this model. However, the document model should provide some means to address the document as a timeline, given that the transformation operations defined hereafter are based on manipulation of timelines. If a document format can not provide a timeline representation, then the transformation operations will be defined only for its continuous media objects. Additionally, the underlying document format should allow addressing fragments of the document and its media elements, given that interaction events are annotated in these elements.

Definition 4.2.1 (Document)**.** A document is a structure (M, R_t, R_s, R_i), where: M is a set of media elements, R_t is a set of temporal relationships between media elements, R_s is a set of spatial relationships and R_i is a set of interactivity resources.

Temporal dimension

The temporal dimension is built upon the distinction between original time and derived time. The discrimination is required because the mapping among these two timelines is not always direct. In general the distinction between these concepts must be made regarding two types of transitions:

- **from capture to media**: in this case, the original time corresponds to the occurrence of the captured media in the real world, e.g. in wall clock time, whereas the derived time corresponds to the temporal scope of the resulting media. In the cases in which a media element has been continuously captured (e.g. in some configurations of meetings and lectures) without interruption, both the original time and the derived time will be represented by a single interval. On the other hand, in the cases where the document has not been continuously captured, the original time will be composed by multiple, non-overlapping intervals whereas the derived time will be composed by a single interval;
- **from a base version to a derived version**: when a document is versioned by editing operations, the temporal correspondences between the underlying timelines is also altered. In this context, the original time corresponds to the time in the original document whereas derived time corresponds to the time in the new version. In unedited recordings, i.e. those that receive only enrichments, there is a direct correspondence between the original time and the derived time, provided that no temporal transformation has been applied. In edited recordings, on the other hand, in case temporal transformations are applied, the original time and the derived time may differ.

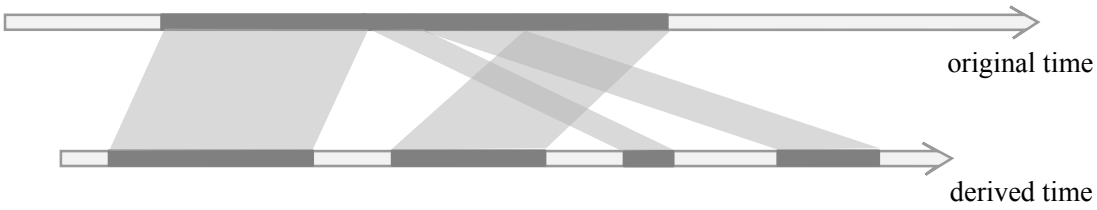


Figure 4.2: An example of mapping between original time and derived time.

An example of lack of correspondence between original time and derived time is demonstrated in Figure 4.2, which illustrates the temporal modifications present in an edited document. In this example, a continuous media element has been recorded in a uninterrupted manner. In an edited document, on the other hand, the media element has been fragmented and those fragments reordered in a way that does not correspond to the original timeframe. Additionally, fragments may be interspersed with other fragments from other capture sessions. Consequently, the resulting document timeline significantly differs from the original media element timeline. Tracking both types of time is particularly important if there is an interest in documenting the original, real-world occurrence, that the media is reporting, as is the case for interaction events in the I+WaC-IE model. Additionally, distinguishing and recording original time and derived time provides important resources to track the temporal provenance of document fragments.

The concept of derived time is also important in the case of interactive multimedia documents. A consequence of associating interactivity constraints to a document is that its temporal schedule becomes undeterministic, so it might not be possible to accurately predict the duration, and

consequently the timeline, of the document presentation before its execution. In fact, upon the occurrence of every interaction, a new timeline is predicted to represent the document duration and, depending on how interactivity is specified in the document, the set of possible timelines generated during the presentation could contain infinite elements. More importantly, the ocurrence of interaction events, on changing the schedule of the presentation, can create a lack of correspondence between the original time and the derived time.

Another issue in which the distinction of original time and derived time is useful refers to the scope of the navigation in the information dataset. In the case of intra-document navigation, derived time is more useful, since the interaction events can be temporally indexed to the beginning of the document, in order to render decorated timelines, for instance. On its turn, original time is particularly useful in inter-document navigation, such as browsing a particular subset of interaction events that spawn through several different documents that were historically captured in distinct periods of time.

These issues must be properly encoded in the interaction events model and, consequently, in the operators. For that purpose, first a set of temporal primitives is defined and then, based on these primitives, original time and derived time are defined.

Definition 4.2.2 (Temporal primitives)**.** The following temporal primitives are important for the definitions in this chapter:

- A *temporal domain* $\{t_1, t_2, t_3, ...\}$ where $t_k \in \mathbb{N}$;
- The *duration* Δ of a time interval $I = [t_a, t_b]$ is $\Delta_I = t_b - t_a$;

Definition 4.2.3 (Original and derived timeline)**.** Let $S = (I_1, I_2, ..., I_n)$ be a sequence of disjoint time intervals, with durations given by the sequence $(\Delta_1, \Delta_2, ..., \Delta_n)$. A timeline is a structure $T = (C_S, I)$ where C_S is the concatenation of every interval of S, in the given order, and $I = [0, \sum_{i=1}^{n} \Delta_i]$ is an interval representing the temporal scope of the timeline. In this context, C_S is the original timeline and I is the derived timeline.

Spatial dimension

Similarly to the time dimension, the spatial dimension is characterized by the distinction of original space and derived space. The distinction among original space and derived space is justified by the difficulty in mapping the spatial domain of interactions in the media scope and the spatial domain of the real world. In the case of unedited recordings of co-located meetings, for instance, some research results have demonstrated the utility of mapping the physical spatial relationships among participants around a table to equivalent spatial relationships in a multimedia presentation [Junuzovic et al., 2008]. Although useful, the mapping of these relationships can not be naturally obtained without additional processing or external knowledge.

In the original space the entities can be modeled in several absolute domains, such as locations (defined for instance via global positioning coordinates) or regions and volumes (defined by

grouping several global coordinates). Moreover, the original space scope can also be mapped to a relative domain, as in the case of in-door positioning systems that frame all locations to the geographical boundary of a building.

Particularly in the case of interaction events, in some situations it is relevant to spatially reference the events considering a composite spatial scope. As an example, when modeling human-object haptic interactions it is necessary to document the spatial boundaries in the object spatial scope where the haptic interaction occurred. This requirement can be observed in interaction events that occur over a board in order to draw strokes or to highlight regions using a laser pointer: in this case it is necessary to document the spatial scope of the object and the region inside this scope where the event occurred. Additionally, given that an annotation is also a media element, document modifications can also transform the spatial constraints of these annotations in a new version.

IAMmDocT does not define operators for spatial transformations, however documenting these kinds of operations has several applications. Spatial attributes can be used, for instance, for selection purposes. This is the case of some types of Inkteractors, such as *boundariesDistance* and *changeOnArea* (refer to Table 2.2), which allows selecting fragments of media elements based on occurrences of spatial attributes in the original space scope. Consequently, even though spatial representations are not used here for transformation purposes, spatial attributes are still useful as a building block for selecting information.

Definition 4.2.4 (Spatial dimension)**.** The spatial dimension is composed of original space and derived space. Regarding capture-to-media transition, original space refers to the spatial framing of an interaction event in the real world, whereas derived space refers to the spatial representation or rendering of the interaction event in the visual media elements comprised in a session. Regarding versioning transition, original space refers to the temporal constraints of the interaction event in the original document whereas derived space refers to the spatial constraints of the interaction event in the derived document.

Interaction dimension

Definition 4.2.5 (Interaction dimension)**.** The interaction dimension is composed of the types of agents that participate in a interaction event and of the types of interactions in which these agents engage.

In the context of an interaction event, an agent can be a human or an object. A human is any participant of the experience whereas an object is any entity relevant for the experience; examples of objects that are recurrent in meetings and lectures are boards, pens, laser pointers, software tools, documents, microphones, mobile devices, etc. In addition, in the case of human-media interactions, media elements are also classified as (intangible) objects. The interactions among agents can adopt several configurations, namely human-human interactions (e.g. dyadic

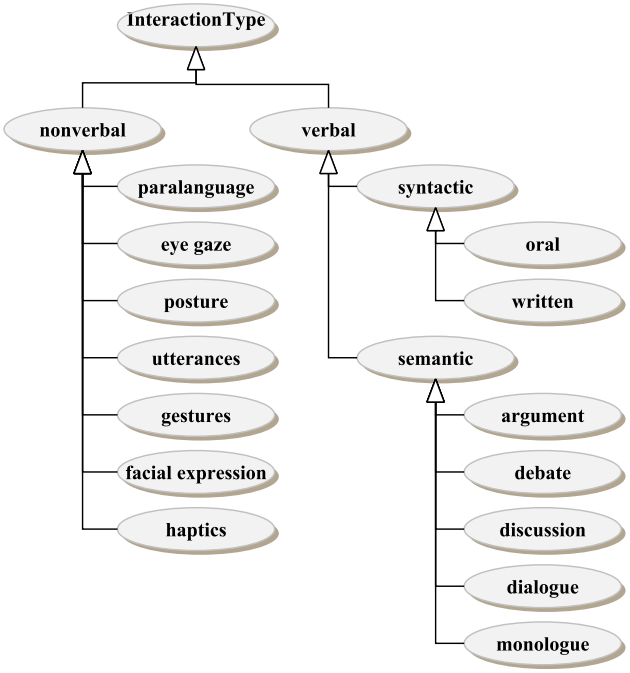


Figure 4.3: Overview of the interaction hierarchy in the I+WaC-IE model

dialogues), human-object interactions (e.g. writing over a board), human-media interactions (e.g. scrubbing a video) and group interactions (e.g. multiparty discussions).

Figure 4.3 depicts the hierarchy of interaction types that is part of the I+WaC-IE model. According to the model, interactions at a higher level can be classified as nonverbal or verbal. The interaction types can be more specialized, for instance verbal communications are classified in syntactic and semantic to denote the level of abstraction of the interactions (e.g. primitive actions or high-level behaviors). In this context we focus on inter-personal interactions occurring in small groups, in opposition to intra-personal, public and mass interactions. This classification is not intended to be exhaustive, but just a initial group of upper entities that can be specialized to particular applications: especially those related to domains involved computer-supported small group activities, such as meetings, lectures and conferences. Consequently, in practical applications in these domains, it is expected that the instances of the interaction dimension will belong to more focused, application-specific concepts.

In summary, the I+WaC-IE model represents interaction events of the production process whereas associating them to a reference document as annotations. I+WaC-IE is a graph-based conceptual model whose instances are direct acyclic graphs (DAGs). As such, any language whose data model represents DAGs can be used to encode the conceptual model. In the remaining of this Chapter, examples of instances of the I+WaC-IE model are given in RDF (Resource Description Framework), a graph-based language that, beyond being largely used for metadata description on the Web, can be naturally be integrated into SMIL documents.

4.3 Overview of IAMMDocT

The Interactors Algebra for Document Transformations is composed of operands and operators that are composed in expressions. The result of an expression is a transformation in the document it is applied to. IAMMDocT is composed by *operands* and *operators*. Operands can be simple, in the case of an annotation, or composite, in the case of interactor timelines:

- an *annotation operand* is a simple operand which represents a single instance of the annotation concept in the I+WaC-IE model;
- an *interactor timeline* is a group of annotations organized in the temporal scope so as their intervals do not overlap. For operation purposes, the relevant information of an interactor timeline is the identification and temporal scope (interval) of each annotation.

Similar to operands, operators are also simple or composite. Simple operators can be applied only to annotations. The only simple operator defined in IAMMDocT is the effect operator. Composite operators can be applied only to timelines and are classified in temporal selection, slicing and projection. Simple operators are applied to simple operands whereas composite operators are applied to composite operands. A simple operator is always unary, whereas composite operators can be unary or binary. Additionally, some operators are used only as helpers (as in the case of temporal join operators and the interactor selection operator), consequently they do not generate document transformations: these operators only assist the transformation operators on retrieving and filtering annotations. In summary, the operators are the following:

- the *interactor*, which is composite and unary, in its selection component, takes as input label of a group of media elements, and retrieves a timeline of annotations;
- the *effect* operator, which is simple and unary. When applied to an annotation, it causes a manipulation in the temporal flow of the document when the annotation is activated;
- the *temporal join*, which encompasses a group of operators, all of them composite. Binary operators include the union join, intersection join and difference join. The only unary operator is the complement join. These operators take as an input a set of interactor timelines and outputs new (joined) interactor timeline;
- the *projection* operator, which extracts individual elements from a document. It generates a new document with only the informed elements;
- the *slicing* operator, which takes as input a document and a timeline and slices the documents to the intervals in the timeline. Consequently, the output is a new document.

The operators can be combined in expressions. However, as the domains and co-domains of the operators vary, expressions must be formed in a way that compatibility between operators is respected. Operators which output a document can only be chained to operators that take a document as input (this is the case of the transformation operators — i.e., projection and slicing). Operators that output timelines can only be chained to operators that take timelines as input (this is the case of interactor selection and temporal join operators). Additionally, operators that generate timelines can be used as parameters to operators that generate documents, but the inverse is not true. In expressions that combine both types of operators (timeline-based and document-based), the timeline-based operations have precedence over the document-based operations.

In the following Section, each of the aforementioned operators are discussed. First, non-transformation operators are discussed: Section 4.4 discuss the Interactors operators and Section 4.5 discuss the temporal join operators and their associated algorithms. Section 4.6 discusses the transformation operators, encompassing: the effect operators (Section 4.6.1), the projection operator (Section 4.6.2) and the slicing operator (Section 4.6.3).

4.4 Interactors: annotation operators

Interaction events, as discussed previously, are captured from heterogeneous sources (e.g. virtual and physical sensors, content analysis, and so son) each employing different representation formats. In order to integrate information from various sources, some mechanism must be provided to transform events from their particular capture format to a common representation format that can be used for authoring purposes. This common format is materialized in the I+WaC-IE model, which serves as a schema to represent interaction events from all sources. For tackling conversion issues, an *Interactor* is defined as an *annotation operator* representing a mechanism to: *i)* transform raw interaction events to the I+WaC-IE format; *ii)* associate an interaction event to a session fragment; *iii)* generate a timeline for a group of interaction events to act as composite operand in IAMmDocT expressions.

An interaction event is an action that occurs when two or more agents have an effect upon each other. Interaction events physically occur in the original time and original space scopes and are documented in the derived time and derived space scopes. Moreover, events are documented by one or more media elements aggregated in a session. Consequently, upon associating an interaction event with a document, some mechanism must be provided to convert the original spatio-temporal grounding to the derived spatio-temporal grounding of the event. In order to define this behavior, first an interaction event is defined (Definition 4.4.1) and then an Interactor is defined (Definition 4.4.2).

Definition 4.4.1 (Interaction event)**.** The *event space* is a 5-ary relation $E = (A, I, T_v, S_v, M)$ that represents all interaction events captured in a production, where:

- A is a set of agents, i.e. humans or objects;
- I is a set of interaction instances;
- T_v is a set of original time intervals;
- S_v is a set of original space regions;
- M is a set of media elements in a production;

An *interaction event* is a tuple $(P, i, t, s, N) \in E$ where:

- $P \subseteq A$ is a set of participants in the event[2];
- $i \in I$ is an interaction instance;
- $t \in T_v$ is the original time interval when the event occurred;
- $s \in S_v$ is the original space region where the event occurred;
- $N \subseteq M$ is a set of media elements that report the event.

Given that the event space contains heterogeneous representations of events, it is necessary some means to associate a group of events which share common attributes to fragments of a document. For this purpose we define an *Interactor* as an annotation operator that associates interaction events to document fragments.

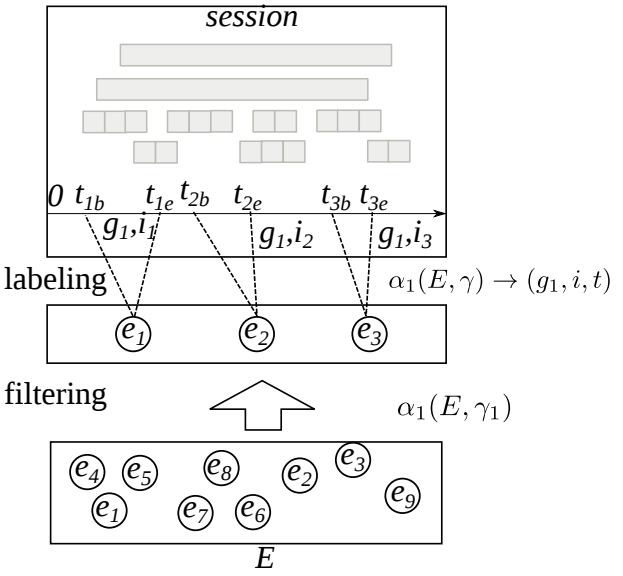


Figure 4.4: Illustration of the execution of an Interactor operator.

[2]The method for identification of participants is dependent on the particular application in which the model is being instantiated. Such identification can be be explicit (e.g. via RFID tags in meeting rooms or via user names in webconferences) or implicit (e.g. via diarization of voices in aural recordings or object recognition in visual recordings).

Definition 4.4.2 (Interactor)**.** An interactor is an annotation operator $O = (\alpha, X)$ where α is a filtering operation and X is a labeling function, both defined as follows:

- $\alpha(E, \gamma) = \{e : e \in E, e \models \gamma\}$ is a filtering operation over the event space E, where γ is a composite restriction over the domains of the event space. The composite restriction γ is a propositional formula over the domains of the dimensions involved in an event space. For instance, the composite restriction $\gamma_1 : a_1 \wedge a_2 \wedge i_1 \wedge m_1$, where $a_1 = professor$, $a_2 = pen$, $i_1 = haptics$ and $m_1 = ink$ restricts the event space to the events on which the following restrictions apply: an agent in the professor role interacted with a pen; the interaction is classified as an haptic interaction; this interaction was documented in a media element of type ink stroke.

- $X : \alpha(E, \gamma) \rightarrow (g, i, t)$ is a function that associates each element e in the filtered event space to a group label g, a production-level annotation identifier i and a interval t corresponding to document fragment.

If an Interactor can associate at least one label to a document, then the Interactor applies to the document.

Figure 4.4 illustrates the behavior of an Interactor. Suppose that a session defines an event space $E = \{e_1, e_2, e_3, ..., e_9\}$ and Interactor $O_1 = (\alpha_1(E, \gamma_1), \alpha_1(E, \gamma_1)) \rightarrow (g_1, i, t)$ has been defined. If O_1 applies to the session, then at least one fragment of the session has been associated to a group label g_1. The operator, at first, filters the interaction events which satisfy the composite restriction γ_1, so that $\alpha(E, \gamma_1) = \{e_1, e_2, e_3\}$. Then for each filtered event the labeling function assigns a label g and an annotation id to an interval of the session so that $X(\{e1, e2, e3\}) \rightarrow \{(g_1, i_1, [t_{1b}, t_{1e}]), (g_1, i_2, [t_{2b}, t_{2e}]), (g_1, i_3, [t_{3b}, t_{3e}])\}$.

A family of Interactors have been defined in previous work (these have been documented in Table 2.2), each defining a distinct group label. For instance, Interactors that are abstracted from audio signals are called *AudioInteractors* whereas Interactors abstracted from electronic ink annotations are called *Inkteractors*. Other types of Interactors include *BoardInteractors* (related to interaction with boards), *VideoInteractors* (which encompass video-related events) and *TextInteractors* (related to interactions via text messages). Each one of these groups define a set of operators, whose names are used as values to the label g. Upon annotating a group of interaction events with a label g, an Interactor forms an *Interactor timeline*, formally defined in the next Section (Definition 4.5.1) due to dependencies on notation.

Listing 4.1: Sample encoding of annotation with no content (metadata-oriented)

```
1 <timesheet xmlns="http://www.w3.org/ns/SMIL">
2   (...)
3   <item id="video84" select="#video84">
```

```
4       <area id="ie431" begin="15" end="28">
5        <metadata>
6         @prefix : <http://(...)/iwac> .
7         :note431 :hasBody :ie431;
8          :hasGroup "participantSpoke".
9        </metadata>
10      </area>
11    </item>
12    (...)
13  </timesheet>
```

An important issue to be discussed is how annotations are encoded in a document. Listing 4.1 illustrates an example of such encoding in a SMIL document. In the example, an `item` element, binded to a video (`#video84`), is being annotated. The annotation is represented by an `area` element, whose `begin` and `end` attributes maps the interval of the annotation. The `metadata` element inside the `area` element encode additional instances of the I+WaC-IE model, using an RDF graph. In this example, for legibility purposes, the more readable Turtle [W3C, 2011c] syntax has been favored in detriment of the usual, but more verbose, RDF/XML syntax [W3C, 2004b]. The graph in the annotation describes the body of the annotation, which is an interaction event (referenced only as an instance identifier) and an instance of the *group* concept, with the value *"participantSpoke"* denoting an audio-based interactor.

Listing 4.2: Sample encoding of annotation with content (content-oriented)

```
1   <timesheet xmlns="http://www.w3.org/ns/SMIL">
2     (...)
3     <item id="video84" select="#video84">
4      <area id="ie395" begin="34" end="122">
5       <metadata>
6        @prefix : <http://(...)/iwac> .
7        :note395 :hasBody :ie395;
8         :hasGroup "audioWaC";
9         :hasContent <http://(...)/audiocomm427.mp3> ;
10        :hasEffect "play".
11      </metadata>
12     </area>
13    </item>
14    (...)
15    <item id="audiocomm427" select="#audiocomm427" />
16    (...)
17  </timesheet>
```

Another example of annotation encoding is given in Listing 4.2. In this case, the annotation is content-oriented, i.e., it has additional content associated with it. The annotation group is ``audioWac'', meaning that it is a WaC comment made with audio. The content of the annotation is referenced by its URL. In this example the *effect* concept is also instantiated, with a value of ``play''. This implies that, when the `area` element is activated, the associated media element will be automatically played. Notice also that, in line 15, the media element

corresponding to the comment is also included in the document. The inclusion of the effect instance in the RDF graph is not enough to realize the effect, given that it demands a document transformation (as will be discussed in Section 4.6.1). In the example, this instance is represented only for documentation purposes.

In summary, an Interactor is a composite operation that defines single operations for filtering events of interest, labeling these events in a session, and generating a timeline out of a group of events. For purposes of use in IAMMDocT expressions, the most relevant output of this operator is the generated Interactor timeline. Consequently, the other operations, event filtering and labeling, can be done offline, prior to execution of an expression. When an expression is executed, all Interactors timelines are already available: consequently, no computation is expected from this operator, but only retrieval of interactor timelines. Therefore, as a shorthand notation, the operation interactor timeline accomplishes the retrieval of the interactor timeline corresponding to a label, where:

$$\Theta(g) \rightarrow ((i_1, [t_{1b}, t_{1e}]), (i_2, [t_{2b}, t_{2e}]), ..., (i_3, [t_{3b}, t_{3e}])) \tag{4.1}$$

- g is an interactor label;
- $((i_1, [t_{1b}, t_{1e}]), (i_2, [t_{2b}, t_{2e}]), ..., (i_3, [t_{3b}, t_{3e}]))$ is the interactor timeline corresponding the label, where each $(i_i, [t_{ib}, t_{ie}])$ is a pair representing an annotation identifier and interval, respectively.

4.5 Temporal join operators

The temporal join operators enable the composition of Interactor timelines using set-theoretic semantics and interval logics. These operators do not lead to document transformations, instead their role is restricted to manipulation of Interactor timelines, a task that occurs prior to a document transformation. As Interactor timelines are operands to transformation operators, the temporal join operators allow to customize Interactor timelines before they are used in an effective document transformation.

Another situation on which such operators can be useful include dynamic visualizations of the annotations, fact-finding tasks over a document, and summarization of a document based on temporal composition between interaction events. This is because they allow to explore temporal composition relationships between interaction events. Consequently, these compositions can be mapped, for instance, to visual representations and used for analysis purposes. Another use case is merging groups of annotations in the temporal domain, and generating a summarized document containing only those merged fragments. The basic idea of a temporal join operator is to perform set-theoretic merging of intervals from Interactor timelines, being each timeline a sequence of non-overlapping time intervals. As a result, a new Interactor timeline is derived. Bearing on these principles, this Section first review preliminary concepts on interval logics and set-theoretic

interval arithmetic (Section 4.5.1), providing the notation for the discussion. Then the temporal join operators are defined and their respective algorithms demonstrated (Section 4.5.2).

4.5.1 Preliminary definitions

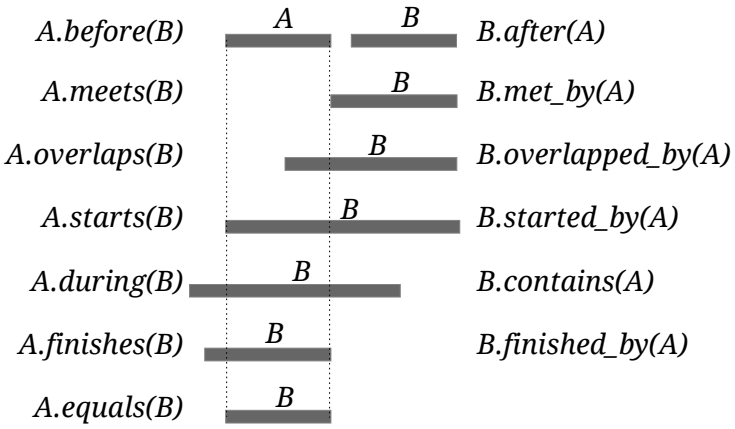


Figure 4.5: Allen's 13 interval relations [Allen, 1983].

As preliminary concepts for the temporal join operators, we take as reference the 13 temporal relationships from Allen's interval logics [Allen, 1983][3]. Figure 4.5 provides a graphical overview of these operators, considering that A and B are intervals. As an additional reference, Table 4.3 present formulations for all relationships. The set of 13 relationships can be reduced to a smaller set of seven relationships if we consider six of them as the inverse of other relationships; the seventh relationship in this set is the *equals* relationship whose inverse does not generate a new relationship. Let $A = [a_1, a_2]$ and $B = [b_1, b_2]$ be time intervals. Table 4.3 lists the relationships and the conditions that must hold for A and B.

Additionally to the relationships detailed in Figure 4.5 and Table 4.3, we define six composite relationships as a shorthand notation to common operations. The composite relationships *left overlaps*, *right overlaps* and *intersects*, among two intervals $A = [a_1, a_2]$ and $B = [b_1, b_2]$, are defined respectively as:

$$\begin{aligned} &A.l\_overlaps(B) \rightarrow a_1 < b_1 \wedge a_2 \geq b_1 \wedge a_2 \leq b2 \\ &A.r\_overlaps(B) \rightarrow a_1 \geq b_1 \wedge a_1 \leq b_2 \wedge a_2 > b_2 \\ &A.intersects(B) \rightarrow a_2 \geq b_1 \wedge a_1 \leq b_2 \end{aligned} \tag{4.2}$$

[3]The 13 Allen's relationships present well known limitations for specifying temporal constraints in multimedia documents. Duda and Keramane [1995] mentions as the most important problems: i) the relations do not represent causal dependency between intervals; ii) the relations depend on interval duration and, as such, demand recomputation of the schedule in case durations change; and iii) the relations may lead to inconsistent schedule, due to ambiguity in the relationships. In this chapter, on the other hand, such limitations do not apply, given that Allen relations are not being used for scheduling purposes, instead they are being used as a tool to define algorithms for joining sequences of intervals.

Table 4.3: Formulations of Allen's interval relationships [Allen, 1983]

base relationship	**condition**	**inverse relationship**	**condition**
$A.before(B)$	$a_2 < b_1$	$A.after(B)$	$b_2 < a_1$
$A.meets(B)$	$a_2 = b_1$	$A.met\_by(B)$	$b_2 = a_1$
$A.overlaps(B)$	$a_1 < b_1 \wedge a_2 > b_1 \wedge a_2 < b_2$	$A.overlapped\_by(B)$	$b_1 < a_1 \wedge b_2 > a_1 \wedge b_2 < a_2$
$A.finishes(B)$	$a_2 = b_2 \wedge a_1 > b_1$	$A.finished\_by(B)$	$a_2 = b_2 \wedge a_1 < b_1$
$A.starts(B)$	$a_1 = b_1 \wedge a_2 < b_2$	$A.started\_by(B)$	$a_1 = b_1 \wedge a_2 > b_2$
$A.during(B)$	$a_1 > b_1 \wedge a_2 < b_2$	$A.contains(B)$	$a_1 < b_1 \wedge a_2 > b_2$
$A.equals(B)$	$a_1 = b_1 \wedge a_2 = b_2$	$A.equals(B)$	$a_1 = b_1 \wedge a_2 = b_2$

The three relationships are interpreted as composite relationships given that they can be represented by means of atomic relationships, namely:

$$
\begin{aligned}
&A.l\_overlaps(B) \rightarrow A.overlaps(B) \vee A.meets(B) \vee A.finished\_by(B) \\
&A.r\_overlaps(B) \rightarrow A.started\_by(B) \vee A.met\_by(B) \vee A.overlapped\_by(B) \\
&A.intersects(B) \rightarrow \neg(A.before(B) \vee A.after(B))
\end{aligned}
\tag{4.3}
$$

In addition to the three composite relationships defined in Equations 4.2 and 4.3, we define three new primitive interval relationships to model the adjacency property of intervals. Since in some circumstances events can be indexed only in a discrete temporal domain, two adjacent intervals can be considered as a single interval. As illustration, consider two intervals $i_1 = [t_0, t_n]$ and $i_2 = [t_{n+1}, t_m]$. These intervals are adjacent because, in the discrete temporal domain, t_n and t_{n+1} are consecutive points. If the points enclosed in the two intervals are enumerated and combined into a sequence of points it yields the sequence $i_r = (t_0, ..., t_n, t_{n+1}, ..., t_m) = [t_0, t_m]$, i.e. the two consecutive intervals i_1 and i_2 enclose the same points as the single interval i_r. In order to take advantage of the adjacency property and its implications, we define the relationships $almost\_meets$ and $almost\_met\_by$, to model left- and right-adjacency respectively[4]. Additionally, we define the shorthand composite relationship $mutually\_continues$ to denote that two intervals are adjacent or intersecting regardless of the reference direction.

The *almost_meets*, *almost_met_by* and *mutually_continues* relationships, among two intervals $A = [a_1, a_2]$ and $B = [b_1, b_2]$, are defined respectively as[5]:

[4]The original Allen's $meets$ relationship does not model an adjacency property. For two intervals $I = [i_b, i_e]$ and $J = [j_b, j_e]$, $I.meets(J) \iff i_e = j_b$, i.e., according to this relationship I and J are not adjacent but overlapping in a single point.

[5]As defined in the temporal primitives (Definition 4.2.2), the temporal domain regarded in IAMMDocT is discrete. Consequently, the definitions of the *almost_meets*, *almost_met_by* and *mutually_continues* relationships are based on these assumptions.

$$
\begin{aligned}
&A.almost\_meets(B) \rightarrow b_1 - a_2 = 1 \\
&A.almost\_met\_by(B) \rightarrow a_1 - b_2 = 1 \\
&A.mutually\_continues(B) \rightarrow \begin{array}{l} A.intersects(B) \vee A.almost\_meets(B) \\ \vee\ A.almost\_met\_by(B) \end{array}
\end{aligned} \tag{4.4}
$$

This extended set of temporal relationships is used to define set-theoretic operations over individual intervals. The *union*, *intersection* and *difference* of two intervals $A = [a_1, a_2]$ and $B = [b_1, b_2]$ are defined respectively as:

$$
A \cup B = \begin{cases} [min(a_1, b_1), max(a_2, b_2)] & \text{if } A.mutually\_continues(B) \\ \{[a_1, a_2], [b_1, b_2]\} & \text{otherwise} \end{cases}
$$

$$
A \cap B = \begin{cases} \{[max(a_1, b_1), min(a_2, b_2)]\} & \text{if } A.intersects(B) \\ \varnothing & \text{otherwise} \end{cases} \tag{4.5}
$$

$$
A - B = \begin{cases} A & \text{if } \neg(A.intersects(B)) \\ [a_1, b_1 - 1] & \text{if } A.l\_overlaps(B) \\ [b_2 + 1, a_2] & \text{if } A.r\_overlaps(B) \\ \{[a_1, b_1 - 1], [b_2 + 1, a_2]\} & \text{if } A.contains(b) \\ \varnothing & otherwise \end{cases}
$$

4.5.2 Operators and their algorithms

All temporal join operators adopt an Interactor timeline as operand, which is formally defined in Definition 4.5.1.

Definition 4.5.1 (Interactor timeline)**.** A sequence of intervals $S = (I_1, I_2, I_3, ..., I_n)$ is a non-overlapping interval sequence if $I_i.before(I_{i+1})$ for every $I_i \in S$. An interactor O_i defines its own *annotation timeline* as a finite non-overlapping interval sequence $T_i = (t_{i1}, t_{i2}, ..., t_{ik})$ where $t_{i1}, t_{i2}, ..., t_{ik}$ are intervals of the document derived timeline in which the same label g_i has been applied by O_i.

Building upon the definition of an Interactor timeline and the set-theoretic interval operations, we define set-theoretic joining of the interval sequences. Figure 4.6 overviews the results of the operators applied to two timelines T_A and T_B.

Definition 4.5.2 (Temporal join operators)**.** A *timeline join operator* is defined as $T_A \bowtie_\varphi T_B = (I_{c1}, I_{c2}, ..., I_{cn})$ where:

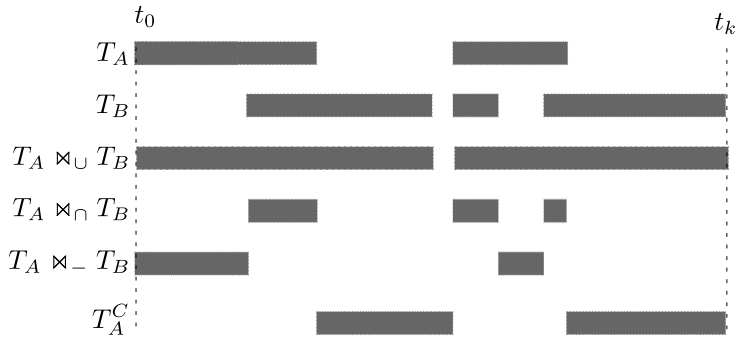


Figure 4.6: Overview of sample results for the composition operators

1. T_A and T_B are Interactor timelines;
2. $T_C = (I_{c1}, ..., I_{cn})$ is a non-overlapping interval sequence;
3. $\varphi \in \{\cap, \cup, -\}$.

According to the variation of φ, different join operations are defined:

- *Intersection join* ($T_A \bowtie_\cap T_B$): $\varphi = \cap$, $I_{ci} = I_{aj} \cap I_{bk}$ where $I_{ci} \in T_C$, $I_{aj} \in T_A$ and $I_{bk} \in T_B$;
- *Union join* ($T_A \bowtie_\cup T_B$): $\varphi = \cup$, $I_{ci} = I_{aj} \cup ... \cup I_{bk}$ where $I_{ci} \in T_C$, $I_{aj} \in T_A$ and $I_{bk} \in T_B$;
- *Difference join* ($T_A \bowtie_- T_B$): $\varphi = -$, $I_{ci} = I_{aj} - I_{bk}$ where $I_{ci} \in T_C$, $I_{aj} \in T_A$ and $I_{bk} \in T_B$.

Based on the timeline difference join we define a *complement operation* of the timeline T_A as non-overlapping interval sequence given as

$$T_A^C = S \bowtie_- T_A \quad \text{where } S = [t_0, t_n] \text{ is the session timeline.}$$

Based on these definitions, in the following topics a group of linear-time algorithms for these operations are discussed. For all the algorithms, let $s_a = (a_1, a_2, ..., a_n)$ and $s_b = (b_1, b_2, ..., b_m)$ be two input sequences, where $a_1, ..., a_n, b_1, ..., b_m$ are closed intervals, $pres$ is a closed interval representing preliminary results, and res is a sequence resulting from a join operation.

Union join

The union join algorithm takes as input two sequences of non-overlapping closed intervals and merges them using the union semantics. The general strategy of the algorithm consists on traversing the two sequences s_a and s_b in parallel, selecting pairs of intervals, one from each sequence, and deciding what to do with each pair. The algorithm combines each pair of intervals $(a_i, b_j), a_i \in s_a \wedge b_j \in s_b$ according to the following cases:

- **case 1**: if a_i before b_j, then append a_i to res, schedule next pair to analyze as (a_{i+i}, b_j);
- **case 2**: if a_i after b_j, then append b_j to res, schedule next pair to analyze as (a_i, b_{j+1});
- **case 3 (merging case)**: if a_i intersects or is adjacent to b_j, or vice versa (refer to the definition of the `mutually_continues` relationship), then append $a_i \cup b_j$ to $pres$ or res.

  In the merging case, the decision about whether to append to $pres$ or res involves the following cases:

  - **case 3.1**: if $pres = \varnothing$, then $pres \leftarrow a_i \cup b_j$;
  - **case 3.2**: if $pres \neq \varnothing \wedge \neg pres.mutually\_continues(a_i \cup b_j)$, then $res \leftarrow pres$ and $pres \leftarrow a_i \cup b_j$;
  - **case 3.3**: if $pres \neq \varnothing \wedge pres.mutually\_continues(a_i \cup b_j)$, then $pres \leftarrow pres \cup a_i \cup b_j$.

Based on this general strategy, a Python implementation is presented in Algorithm 4.3:

Listing 4.3: Python implementation of the union join algorithm

```
def union_join(sa, sb):
 ai = 0
 bi = 0
 res = []
 c = None
 presult = None

 while ai < len(sa) and bi < len(sb):
  a = sa[ai]
  b = sb[bi]

  # check merge among the two intervals
  # if a.intersects(b) or a.adjacent(b):
  if a.mutually_continues(b):
   c = union_interval(a,b)[0]
   ai += 1
   bi += 1
  elif a.before(b):
   c = a
   ai += 1
  elif b.before(a):
   c = b
   bi += 1
  # check merging with preliminary union
```

```
  if not presult:
   presult = c
  elif presult and not presult.mutually_continues(c):
   res.append(presult)
   presult = c
  elif presult and presult.mutually_continues(c):
   presult = union_interval(presult,c)[0]
 # flush remaining items
 if ai == len(sa):
  while bi < len(sb):
   c = sb[bi]
   if presult:
    if presult.mutually_continues(c):
     presult = union_interval(presult, c)[0]
    else:
     res.append(presult)
     presult = c
   else:
    res.append(c)
   bi += 1
 elif bi == len(sb):
  while ai < len(sa):
   c = sa[ai]
   if presult:
    if presult.mutually_continues(c):
     presult = union_interval(presult, c)[0]
    else:
     res.append(presult)
     presult = c
   else:
    res.append(c)
   ai += 1
 if presult:
  res.append(presult)

 return res
```

Intersection join

The intersection join algorithm is conceptually similar to the union join algorithm, namely takes the same types of inputs, returns the same types of result (in all cases, non-overlapping interval sequences) and traverses the two sequences in parallel. The main difference between them is in the strategy for combining intervals: in the case of the intersection join no sub-cases for keeping preliminary results are needed in the merging phase. The decision part of the algorithm consists on dealing with the following cases:

- **case 1**: if a_i before b_j, then schedule next pair to analyze as (a_{i+1}, b_j);
- **case 2**: if a_i after b_j, then schedule next pair to analyze as (a_i, b_{j+1});
- **case 3**: if a_i intersects b_j, then append $a_i \cap b_j$ to res and replace the earliest interval when building the next pair to analyze.

An implementation in Python is presented in Algorithm 4.4.

Listing 4.4: Python implementation of the intersection join algorithm

```
def intersection_join(sa, sb):
 ai = 0
 bi = 0
 res = []
 presult = None

 while ai < len(sa) and bi < len(sb):
  a = sa[ai]
  b = sb[bi]

  # check merge among the two intervals
  if a.before(b):
   ai += 1
  elif b.before(a):
   bi += 1
  elif a.intersects(b):
   c = intersect_interval(a,b)[0]
   # discard only the earliest interval
   if a.end > b.end:
    bi += 1
   elif a.end < b.end:
    ai += 1
   else:
    ai += 1
    bi += 1
   res.append(c)

 return res
```

Difference join and complement join

Analogously to the strategies employed by the intersection join and the union join algorithms, the difference join also traverses two input sequences in parallel and merge them into a final result. But a different characteristic is that the final result is initialized with one of the sequences

and afterward it is incrementally subtracted with elements from the other sequence (given the difference join operation $s_a \bowtie\!\!\!- \; s_b$, in the beginning of the algorithm $res = s_a$, and in the end of the algorithm $res = s_a \bowtie\!\!\!- \; s_b$). In summary, the general idea of the algorithm consists on initializing the result sequence with the left-hand operator ($res \leftarrow s_a$) of the difference and traverse the sequences, analyzing each pair (a_i, b_j) according to the following cases:

- **case 1**: if a_i before b_j, then keep a_i in res, next pair is (a_{i+1}, b_j);
- **case 2**: if a_i after b_j, then skip b_j, next pair is (a_i, b_{j+1});
- **case 3**: if a_i intersects b_j, check the difference $a_i - b_j$; depending on the number of intervals generated:
  - **case 3.1**: if no interval, remove a_i from res;
  - **case 3.2**: if one interval, replace a_i for this interval in res;
  - **case 3.3**: if two intervals, replace a_i by the two intervals in res.

  In all sub-cases, build the next pair considering the current position in res and s_b.

The following code snippet presents an implementation of the difference join algorithm in Python.

Listing 4.5: Python implementation of the difference join algorithm

```
def difference_join(sa, sb):
 ai = 0
 bi = 0
 la = sa[:] # copy by value

 while ai < len(la) and bi < len(sb):
  a = la[ai]
  b = sb[bi]

  # c1
  if a.before(b):
   ai += 1
  # c2
  elif a.after(b):
   bi += 1
  # c3
  elif a.intersects(b):
   c = difference_interval(a, b)
   # no interval, remove from list
   if len(c) == 0:
```

```
21      la.pop(ai)
22     # one interval, replace in the list
23     elif len(c) == 1:
24      la[ai] = c[0]
25     # two intervals, replace 1st and insert 2nd in the list
26     elif len(c) == 2:
27      la[ai] = c[0]
28      ai += 1
29      la.insert(ai, c[1])
30
31  return la
```

The complement join algorithm is defined by means of the difference join algorithm. For that to be possible it is required that an "universe sequence", i.e. an unary sequence holding the interval that represents the whole temporal scope being considered. Assuming that the universe sequence is provided as input, the algorithm is simply the difference of the input sequence from the universe sequence. A python implementation is presented in the following snippet:

Listing 4.6: Python implementation of the complement join algorithm

```
1  def complement(t, u):
2     return difference([u], t)
```

In summary, the final purpose of a temporal join algorithm is to generate derived timelines from a set of Interactor timelines given as input. These results have several applications to the transformation operators. For instance, a temporal join might be necessary if a document transformation task entails extracting only the fragments of a document that contains the annotations of two Interactors simultaneously. In this case, a union join operation should be executed and the result of this operation could be used as input to the a transformation operation. In the following Sections, the transformation operators are discussed.

4.6 Transformation operators

The transformation operators are classified into *enrichment operators* and *editing operators*. Enrichment operators do not materialize a new version of a document, i.e. they modify their target document by incorporating additional constructs. Editing operators involve more intrusive operations that require generating a new version of the document. Therefore, whereas an enrichment operator outputs the same document taken as input (but with enrichments applied), an editing operator outputs a new document. Naturally, an enriched document, for most practical purposes, could be regarded as a new document version as well. However, for live document transformation purposes, this differentiation is useful. This is because enrichment operations,

due to their limited intrusion in a document, can be applied via a live editing operation without a complete re-computation of the document layout. Editing operations, on the other hand, can impact the document in profound ways that their application via live editing most likely will require a complete layout computation, having, as a consequence, the same cost of rendering a new document version.

In the remaining of this Section, these two classes of operators are discussed. First, in Section 4.6.1, it is discussed the effect operators, which are enrichment operators. Then editing operators are discussed in Section 4.6.2, which discusses the projection operator, and Section 4.6.3 which discusses the slicing operator.

4.6.1 Effect operators

An effect operator is the most basic type of operator in IAMMDocT. It is an operator that can be applied to any annotation of an interaction event. These operators can be unary or binary. Each effect operator takes as input one or more annotations and modifies the temporal behavior of the presentation around the annotation intervals. Consequently, an effect operator is a transformation operator that acts over a group of annotations. An unary[6] effect operator is defined by Equation 4.6, where:

$$\varepsilon_\gamma : (D, a) \rightarrow D' \tag{4.6}$$

- $\gamma \in \{pause, skip, loop\}$ is the type of operator;
- D is the original document;
- $a = (g, i, t)$ is an annotation as per Definition 4.4.2;
- D' is the derived document.

Based on the variation of γ, different operators can be defined. Each one of them cause a different effect in the temporal flow of the document when traversed, namely:

- ε_{pause}: the pause effect, which pauses the document when traversed;
- ε_{skip}: the skip effect, which ignores the interval i of a, by jumping straight to the end of the interval when traversed;
- ε_{loop}: the loop effect, which repeatedly traverses the interval i of a until another condition interrupts it.

[6]For the effect operators, unary or binary denotes the number of annotations that are taken as parameters for the operators: unary are the ones that take one annotation, and binary are the ones that take two annotations.

The only binary effect operator is the jump effect, which connects two annotations by a jump in the document flow. In practice, the jump effect in the document is similar to that of an "automatic link" which, when activated by the document schedule, is automatically traversed. The jump operator is defined in Equation 4.7, where:

$$\varepsilon_{jump} : (D, a_s, a_t) \rightarrow D' \tag{4.7}$$

- D and D' are original and derived documents, respectively;
- a_s is the source annotation;
- a_t is the target annotation.

In order to demonstrate the application of the effect operators in a concrete document, implementations of their effect in SMIL are demonstrated. Each operator realize a transformation in the document by inserting a document pattern that enforces its behavior. The absence of an effect, i.e. the default behavior, leads to the patterns demonstrated in Listings 4.1 and 4.2. The following discussion illustrates how these original patterns are transformed via an effect-related pattern.

Listing 4.7: SMIL excerpt for the *pause* operator

```
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  (...)
  <item id="video84" select="#video84">
   <area id="ie431" begin="15" end="28" actuate="onload" sourcePlayState="pause" href="#">
    <metadata>
     @prefix : <http://(...)/iwac> .
     :note431 :hasBody :ie431 ;
      :hasGroup "participantSpoke" ;
          :hasEffect "pause" .
    </metadata>
   </area>
  </item>
  (...)
</timesheet>
```

Listing 4.7 illustrates the result of transforming Listing 4.1 with the *pause* operator. The modifications encompassed by this operator are minimal, restricted to the inclusion of the `actuate` and `sourcePlayState` attributes in the `area` element. The value "onload" in the `actuate` attribute will make the link be traversed as soon as it is activated. As soon as the link is traversed, the `sourcePlayState` will make the element be paused, a state that will be propagated to the ancestors, consequently pausing the presentation.

Listing 4.8: SMIL excerpt for the *jump* operator

```
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  (...)
  <item id="video84" select="#video84">
   <area id="ie431" begin="15" end="28" actuate="onload" href="#ie395">
    <metadata>
     @prefix : <http://(...)/iwac> .
     :note431 :hasBody :ie431;
      :hasGroup "participantSpoke";
          :hasEffect "jump" .
    </metadata>
   </area>
  </item>
  (...)
</timesheet>
```

The transformations required by the jump operator are also minimal, as demonstrated in Listing 4.8. This operator embodies the basic behavior of internal links, consequently, the only modification required is assigning the *onload* value to the `actuate` attribute and indicating the target of the link via the `href` attribute. In the example, the target of the link is the link corresponding to Listing 4.2. As a consequence of this pattern, when the interval of the annotation is activated, the link is immediately traversed and the presentation "jumps" to the interval of the second annotation.

Listing 4.9: SMIL excerpt for the *loop* operator

```
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  (...)
  <item id="video84" select="#video84">
    <par id="ie431" begin="15" end="28">
      <area id="link_ie431_l1" dur="13" />
      <area id="link_ie431_l2" actuate="onload" begin="link_ie431_l1.end" dur="0.5"
        href="#link_ie431_l1" />
      <metadata>
        @prefix : <http://(...)/iwac> .
        :note431 :hasBody :ie431 ;
          :hasGroup "participantSpoke" ;
          :hasEffect "loop" .
      </metadata>
    </par>
  </item>
  (...)
</timesheet>
```

Listing 4.9 demonstrates the effect of the loop operator, which demands a more elaborate transformation. The first one of them is replacing the original area link by a time container `par` that acts as wrapper to the document pattern. This time container assumes the interval of the annotation. Then, two links are included inside the container, in this order: first, a link with the duration of the annotation interval (`link_ie431_l1`); and then a link with a symbolic duration of $0.5s$ (`link_ie431_l2`). The second link is responsible for triggering the loop. It is activated as soon as the first link ends: this is achieved by assigning a syncbase constraint to its `begin`

attribute. Because of its actuate attribute, as soon as the link is activated, it is also traversed, and the target is the first link. Consequently, a loop occurs.

Listing 4.10: SMIL excerpt for the *skip* operator

```
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  (...)
  <item id="video84" select="#video84">
    <par id="ie431" begin="15" end="28">
      <area id="link_ie431_l1" dur="0.5" actuate="onload" href="#link_ie431_l2" />
      <area id="link_ie431_l2" begin="13" />
      <metadata>
        @prefix : <http://(...)/iwac> .
        :note431 :hasBody :ie431;
          :hasGroup "participantSpoke";
          :hasEffect "skip".
      </metadata>
    </par>
  </item>
  (...)
</timesheet>
```

Listing 4.10 illustrates the effect of the skip operator, which also follows a double-link, composite pattern. As a consequence, the original link is replaced by a par container with two child links. The first link is traversed as soon as it is activated, having as target the second link. The second link, by its turn, is scheduled to the end of the first link. Consequently, the duration of the annotation is skipped.

Being enrichment operators, all effect operators can be applied while the document is active. At least one incremental update of the document schedule for each transformation is expected. The number of updates can be minimized, for instance, by batch-executing modification of multiple attributes, as is applicable to the pause and jump operators. In the other cases, more than one timegraph update is expected, provided that each one of them encompass multiple editing operations in the SMIL document (e.g. create the time container and links, then append the structures to the document).

4.6.2 Projection operator

The projection operator filters the set of elements in the document to include only a subset of them. This operator is useful for customizing presentations which have multiple media elements, for instance, by extracting only a group of videos from a multi-video presentation. When an element is projected into a new presentation, its associated annotations, if any, are included as well. Definition 4.8 introduces the operator, where:

$$\pi : (\{M_1, M_2, ..., M_k\}, D) \rightarrow D' \tag{4.8}$$

- $\{M_1, M_2, ...M_k\}$ is a set document fragment identifiers;
- D is the original document;
- D' is the derived document.

In order to illustrate the applicability of the operator in a concrete document, consider as base document the one presented in Listing 3.2. Assume that only a single video element, the one containing the lecturer, and all the slides should be included in a new version. This will correspond to the following projection in Equation 4.9.

$$\pi : (\{\texttt{\#video\_82}, \texttt{\#slides}\}, \texttt{(...)/timesheet/1304.xml}) \rightarrow D' \tag{4.9}$$

Listing 4.11: An example document generated via the projection operator

```
<timesheet xmlns="http://www.w3.org/ns/SMIL">
  <par id="sliced_presentation">
    <par id="group_me">
      <timesheet id="mevideo_82" src="(...)/timesheet/1304.xml#video_82"/>
      <timesheet id="slides" src="(...)/timesheet/1304.xml#slides"/>
    </par>
  </par>
</timesheet>
```

Listing 4.11 illustrates the result of this operation, i.e., the representation of document D'. In this document, the projection operator made use of reuse features in order to generate a new document. The selected video, represented in the original document as an `item` element identified as "video_82", is reused in the new document via the `src` attribute of the first `timesheet` element. It makes use of the element reuse feature in ActiveTimesheets (discussed in Section 3.6.1). The same is true for the second element, which reuses the time container identified as "slides" in the original document. This is also done by including a `timesheet` element whose `src` attribute references the reused element via a fragment identifier. Naturally, the projection operator only projects the temporal specification of the projected media elements. As a consequence, some transformation in the spatial layout should be provided as well so that only the projected media elements be visually presented. As IAMmDocT applies transformations only to the temporal layout of a document, spatial transformations should be done via a separate mechanism.

A simple alternative for connecting spatial and temporal transformation, without modifying the spatial document, is to take advantage of CSS. ActiveTimesheets, for instance, upon scheduling a media element in the spatial document, assigns an attribute `data-tstate` whose value is the state of the timegraph node which controls the media element, i.e., values such as "playing",

"paused", "idle", and so on. CSS rules could be defined to hide the media element, when its `data-tstate` attribute has the default "idle" value, and to display it, when the attribute has the "playing" value. A consequence of this solution is that, from the visual standpoint of the presentation, only the elements that are activated are shown. Further, it allows to connect the temporal transformation to the spatial document without enforcing a spatial transformation. Naturally, for more elaborate use cases alternative solutions must be adopted, for instance via selection of spatial layout templates.

4.6.3 Slicing operator

The slicing operator take as an input a set of interactor timelines and fragments a document or a media element based on the input intervals. The result of this operator is a "playlist" of fragments, i.e. a list of concatenated document fragments. The slicing operator is defined in Equation 4.10, where:

$$\sigma : (\Theta(g_1), \Theta(g_2), ..., \Theta(g_n), M, \gamma) \rightarrow ((F_1, F_2, .., F_k), M) \tag{4.10}$$

- $\Theta(g_i)$ is an interactor timeline given the label i;
- M is a media element or a document;
- $\gamma \in \{-, \cup, \cap\}$ is a binary operation;
- (F_1, F_2, F_k) is a list of fragments, where each F_i is a fragment of M;
- the structure $((F_1, F_2, ..., F_k), M)$ is the new document.

In order to derive a sequence of fragments based on interactor timelines, the slicing operator takes advantage of the temporal join operators. Therefore, all interactor timelines are merged into a single timeline before the transformation is applied. Which temporal join operator will be used for this task can be selected as a parameter. Once the operator has a single resulting timeline, it builds a list of fragments for the document or media element M and creates a new document concatenating these fragments.

In order to demonstrate the applicability of the slicing operator in a concrete document, suppose the document represented in Listing 3.2, denoted as D, has been annotated with two interactor timelines, $\Theta(l1)$ and $\Theta(l_2)$, which define the following intervals:

$$\begin{aligned} \Theta(\text{group\_1298}) &= ((\text{link46421}, [8, 15]), (\text{link46420}, [20, 35]), (\text{link46242}, [46, 57])) \\ \Theta(\text{group\_1297}) &= ((\text{link46238}, [32, 50]), (\text{link46239}, [54, 65])) \end{aligned} \tag{4.11}$$

Now suppose the slicing operator has been called for $D =$ (...)/timesheet/1304.xml#presentation with the union-join semantics. The result will be:

$$\sigma(\{\Theta(\text{group\_1298}), \Theta(\text{group\_1297})\}, D, \cup) \rightarrow (((8, 15), \#\text{f1}), ((20, 65), \#\text{f2}))$$

The transformed document, which is a new version generate from scratch, is illustrated in Listing 4.12.

Listing 4.12: An example of document generated via the slicing operator

```
1  <timesheet xmlns="http://www.w3.org/ns/SMIL">
2    <par id="sliced_presentation">
3      <seq id="group_me">
4        <timesheet id="f1" clip-begin="8" clip-end="15"
5          src="(...)/timesheet/1304.xml#presentation"/>
6        <timesheet id="f2" clip-begin="20" clip-end="65"
7          src="(...)/timesheet/1304.xml#presentation"/>
8      </seq>
9      <par id="group_1298">
10       <area begin="8" end="15" id="link46241">(...)</area>
11       <area begin="20" end="35" id="link46240">(...)</area>
12       <area begin="46" end="57" id="link46242">(...)</area>
13     </par>
14     <par id="group_1297">
15       <area begin="32" end="50" id="link46238">(...)</area>
16       <area begin="54" end="65" id="link46239">(...)</area>
17     </par>
18   </par>
19 </timesheet>
```

The document in Listing 4.12 applies to aforementioned slicing operation to the document illustrated in Listing 3.2. The generated document takes advantage of reuse features provided by ActiveTimesheets. First, the source presentation, identified in the original document as "presentation" is reused via a fragment identifier in the `src` attribute of the `item` element. This is done for every resulting interval of the slicing operator, which in the example encompass two intervals. In addition, each reused element is fragmented to the values of the sliced interval: this is done via the extended clipping attributes provided by ActiveTimesheets. As a consequence, to every reused element the attributes `clip-begin` and `clip-end` are applied, having as values the begin and end of each slicing interval, respectively. In this case, the whole presentation is being reused and clipped, including multiple videos, images, text and so on. The fragmented clips are integrated into a `seq` container, having as effect the concatenation of the fragments. As a consequence, the duration of the derived presentation will correspond to the sum of the durations of each fragment. Finally, the annotations included in the original document are transferred to the new document (lines 7-16) as well: this measure allows their semantics to be incorporated in new document versions. In case annotations contain sensitive content that should not be exposed, then the implementation of the operator can be parameterized not to include the annotations in a new version.

4.7 Final remarks

This Chapter has presented the Interactors Algebra for Multimedia Document Transformations and its associated issues. A graph-based model for abstracting annotations in multimedia production has been presented. This model, based on the concept of interaction event, is used, first, as a schema to document captured annotations and, then, its instances are used for subsidizing document transformation. A group of operators have been defined to perform transformations in a multimedia document: these operators have their semantics defined and their algorithms discussed where applicable. As far as relationships with other Chapters are concerned, it has been demonstrated the applicability of the ActiveTimesheets language (discussed in Chapter 3) to the problem of document transformations. In particular, the importance of live editing features have been demonstrated in the realization of the effect operators, which can be applied in a active multimedia document and the changes perceived immediately. Additionally, the importance of the inclusion of linking functionality in ActiveTimesheets has been demonstrated as well, given that concrete representations of these transformations make heavy use of linking. The importance of the reuse features provided by ActiveTimesheets have been shown as well, particularly in the case of the projection operator, which reuses elements from external documents, and of the slicing operator, which reuses fragmented elements from external documents.

The operators defined in this Chapter are applied, in this dissertation, to manual authoring mostly. However, a detailed formalization of transformation operations over multimedia documents brings several benefits to support increasingly automated approaches. Semi-automatic authoring approaches, for instance, can take a set of initial input from the user and apply this input in order to orchestrate a series of selection, fragmentation and composition operations, interactively fine-tuned by the user at key points of the workflow. Additionally, an automatic approach would attempt to execute the whole workflow without user intervention. The operators defined in this Chapter have potential to subsidize both semi-automatic and automatic approaches, given that they provide a formalization of timeline-oriented presentations. The utility of the operators is more outstanding in document extension scenarios, i.e., those in which an existing multimedia document is customized in focused ways. In these cases, operations such as extracting fragments, trimming the presentation, and so on, are recurrent.

The operators here have been defined strongly based on a composite timeline model, i.e. most of the operations are based on the combination of timelines from various annotation scopes. This model is useful for a number of multimedia presentations, especially the ones that are recurrent in capture and access productions: lectures, webconference recordings, talks, meetings and so on. In these cases, the final recording, even in its composite form, have a “master timeline” over which all media elements can be aligned to. This timeline not always can be embodied in a separate media element, as such it is important that the underlying document format allows the definition of timelines. In certain types of presentations, however, such setup might not be possible, for instance in the case of purely constraint-based presentations, e.g. hypervideos in

which interactivity can alter the flow of the story and, consequently, its duration. Tackling such harder use cases require deeper investigation, which will likely demand adaptations to the model and the definition of new operators as well. The results derived from such an effort will allow, for instance, the application of IAMMDocT to other popular multimedia document formats, such as NCL.

Even though the I+WaC-IE model defines a spatial dimension, the IAMmDocT does not include spatial transformation operators. As far as spatial attributes are concerned, in IAMmDocT they are currently used in the annotation operator, which can use spatial attributes in its propositional formula in order to select a group of interaction events. However, spatial transformation operators would also be useful for authoring purposes, for instance for panning and zooming a presentation over individual media elements. Additionally, in case some media element is projected in a new document, it is interesting that this projection be reflected in the spatial layout. Even though such operators have not been defined, the foundation provided by the I+WaC-IE model can be a starting point for future investigation on this issue.

The I+WaC-IE model tackles basically low level events, such as the ones obtained from physical and virtual sensors. Even though such events might prove useful in many situations (e.g. to know when a slide was switched or when someone talked), in many other situations a user might be interested in getting information in a higher level. Multimodal analysis of a recording can generate interesting events that describe the semantics of a presentation, for instance social behaviors, visual events, and so on. However, obtaining such events represents a hard problem, provided that high level information derived from low level sensors (be it physical sensors, video-based sensors and so on) is far from being a solved problem. In fact, just the problem of recognizing objects from visual streams is a ample field by itself, with plenty of open problems. Provided these difficulties, the contributions of this Chapter are not focused on providing an complete event recognition framework that could be used for retrieval tasks inside a document. Instead, the focus here is on defining a family of operators that take advantage of these events, provided that they are available. Most importantly, the research in the Chapter contributes with a document engineering approach to generate, given a set of annotations, a customized version of a multimedia document.

A compelling use of IAMmDocT is for providing enrichments, especially via the effect operators. Defining other types of enrichments, for instance overlaying additionally content, can significantly improve the user experience of a tool that employs the operators. Naturally, this contribution would also benefit from spatial transformation operations.

Finally, a feature that is recurrent in algebraic data manipulation mechanisms, take query languages for instance, is the optimization of a query before executing it. This measure has the objective of computing an optimal execution plan before the data is manipulated, given that manipulation can be a costly operation with big amount of data involving, also, retrieval of data chunks from secondary memory. In case of IAMmDocT, the most important bottleneck is the transformation operators, which perform costly manipulations in the document in order

to generate new versions. The cost of these manipulations can be assessed in live editing and offline editing. Currently, IAMmDocT does not provide optimizations for the execution of the expressions, being prevalent the precedence of the operators. Consequently, investigation of the performance of different expression evaluation algorithms is also an issue for future work.

Chapter 5

Interactors+WaC-Editor: a Web-based tool for extending active multimedia documents

Interactors+WaC-Editor (I+WaC-Editor) is a Web-based multimedia authoring tool that focuses on extension of multimedia documents. Differently from a general-purpose authoring tool, in which multimedia presentations are created from scratch, in I+WaC-Editor multimedia documents are ingested from external sources, converted to the ActiveTimesheets language and enriched with the IAMmDocT operators. Consequently, I+WaC-Editor integrates the contributions of this dissertation into an amateur-oriented authoring tool.

The Editor was originally conceived to be a playback and enrichment tool for recordings of synchronous communication sessions. Examples of these sessions are the ones recorded from webconference systems, videoconference systems and general multimedia-based social communication tools. These tools are extensively used for communication between small groups, in domains such as meetings, lectures, social gatherings (families, friends, etc.), and so on. In some situations, synchronous communication sessions are recorded for later reference, for instance for reviewing in more details, for archiving purposes, or for publication. These recordings are generally composed by multiple continuous and discrete media streams whose temporal constraints reconstruct the original timing observed at capture. Spatial layout, however, is up to the playback engine that makes use of these sessions. Consequently, recordings of synchronous collaboration tools are compelling datasets for experimenting on enrichment of temporal multimedia documents.

In this Chapter it is demonstrated the most important issues of I+WaC-Editor and how it integrates the research of this Chapter into its functionalities. The remaining of this Chapter is organized as follows: Section 5.1 provides an overview of its architecture; Section 5.2 documents the ingestion process of sessions in the architecture; Section 5.3 reports how multimedia documents are automatically generated in the architecture; Section 5.4 describes the client-side

issues of the editor, so that the remaining Sections report the most important functionalities of the player, such as document playback, browsing, annotation, enrichment, transformation and sharing.

5.1 I+WaC-Editor architecture

As I+WaCEditor was designed to be independent of a specific capture and access infrastructure, it has its own separate client-server architecture. This architecture is illustrated in Figure 5.1.

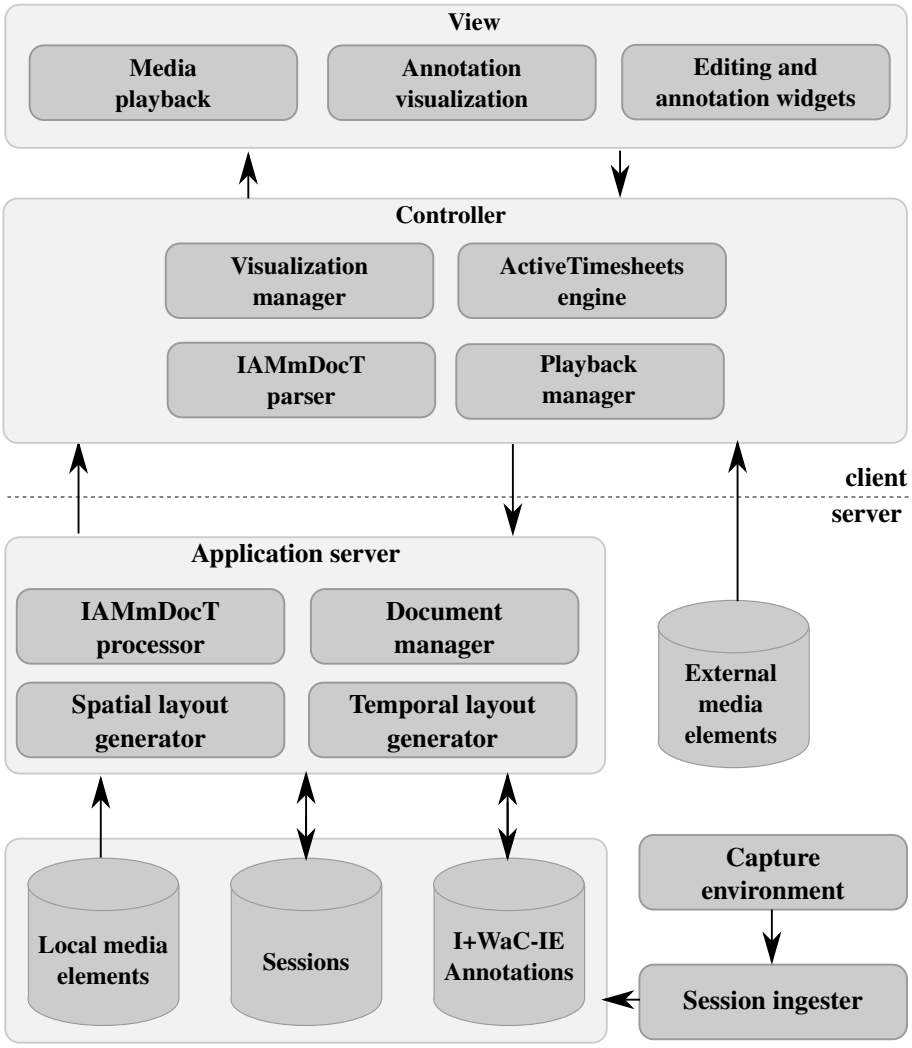


Figure 5.1: I+WaC-Editor architecture

In the architecture of Figure 5.1, on server-side occur processes related to data ingestion, document generation and document transformation, whereas at client-side occur processes of document rendering, enrichment and editing operations. For all these issues, separate components have been developed. The following discussion provides an overview of the general behavior of

the architecture and its components. Then, next Sections of the Chapter provide more details on each of the involved processes.

Media elements in I+WaCEditor are referenced from a local source, in the case of ingested sessions, or from external sources, in the case of enrichment of Web videos. Besides media elements, the ingestion process also feeds a repository of capture sessions, which stores the ingested sessions in an internal format, and annotations associated to the ingestion session, if any. Some capture environments are capable of logging important interaction events during capture, and in some cases these events are exported with the session. In these cases, the data is ingested as instances of the I+WaC-IE model.

Once capture sessions have been ingested they can be served to clients as multimedia documents. For this purpose, a group of components in the I+WaC-Editor application server is defined. The spatial layout manager generates an HTML5 document for every session, on demand. The generated spatial document follows one out of a group of predefined layout templates. The temporal layout manager automatically creates a multimedia document in the ActiveTimesheets language. For this purpose, it takes advantage of the timing information and media elements ingested with the session. Naturally, the generated ActiveTimesheets document is binded to the spatial document via selection expressions and, consequently, the expressions in the temporal document must reflect the structural compositions and identifiers used in a particular spatial document template. Upon request, the spatio-temporal multimedia document is generated and served to the client.

The I+WaC-Editor client has been implemented using only native Web technologies (HTML, CSS and Javascript), thus it does not require the use of user agent plugins. At client side, the received document is rendered in the playback infrastructure. Spatial layout is a concern of the user agent, whereas temporal layout is achieved by the ActiveTimesheets engine. Additionally, annotations associated with the multimedia document, represented as instances of the I+WaC-IE model, are visually rendered in a timeline-based visualization by the visualization manager component. The purpose of rendering these annotations is twofold: first, the visualization can be used for browsing purposes, i.e., as annotations act as links in the document, a user can perform non-linear access to the session; second, the annotations are also the reference for enriching and editing the document, consequently the visualization of each annotation also acts as a widget for editing the document. General playback functionality, such as VCR controls, are managed by a separate playback component, which manipulates software interfaces provided by the ActiveTimesheets engine.

The client interface also includes functionalities for using the IAMmDocT operators. In the I+WaC-Editor, these operators have two uses: *i)* for browsing purposes, they are used to visually combine annotations in the temporal scope, as a mechanism to support the analysis of the session; *ii)* for authoring purposes, they are used as visual surrogate to Interactor timelines, consequently they are used as input for document transformation operations. For both uses, a client component tackles the problem of parsing an IAMmDocT expression, converting it to the

syntax expected by the server, and then sending the expression to be evaluated on the server. On server-side, the IAMmDocT processor evaluates the expression. In the case of an expression generated for browsing purposes, only the results are returned; in the case of an expression generated for authoring purposes, the result is taken by the document manager component that generates new documents based on the transformation operations defined in the expression. As a consequence, the generated document is stored in the sessions repository and can then be requested by the client as a regular document. The following Sections details the processes involved in this architecture.

5.2 Ingestion process

As the I+WaC-Editor is designed to be used by several capture environments, sessions exported by these environments must be ingested in the I+WaC-Editor infrastructure. The ingestion process involves interpretation of the session exchange format and distribution of the enclosed data to the various repositories. This process is described in Figure 5.2.

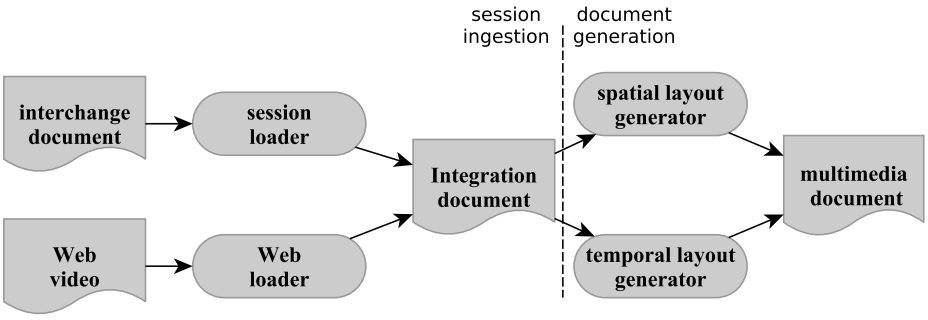


Figure 5.2: I+WaCEditor document conversion workflow

Figure 5.2 describes the ingestion process in the context of the whole document generation workflow. The ingestion process consists on converting an input session into an integration format, which is the representation used for internal storage in the architecture. Two types of sources are supported in the architecture: interchange documents, which are exported by capture environments; and web videos, which can be any video with a URL. The integration format is later used to store document enrichments and generate multimedia documents.

Capture environments that export their recorded sessions, such as BigBlueButton[1], Matterhorn[2] and DiGaE [Linhalis et al., 2010b], generally do so using a structured format, hereafter

[1]BigBlueButton web conferencing environment. URL: `http://bigbluebutton.org/`. Accessed September, 2013

[2]Matterhorn lecture webcasting environment. URL: `http://opencast.org/matterhorn`. Accessed September 2013

called interchange format. The interchange format is a bundle containing an event description document and media elements (embedded or as references). The event description document reports all synchronization events between media elements as logged at capture-time: this includes activation intervals for videos, audio files, images, text messages, and so on. Additionally, the interchange document can contain annotations of interaction events, such as voice activity, participants joining and leaving the session, "raise hands" events (denoting a request for asking a question, for instance), and so on. All these events are synchronized to the duration of the session, which in most cases does not coincide with the duration of any of its media elements. As a consequence, the session has its own internal timeline which corresponds to the time interval in wallclock time when the session was captured. A sample interchange document is illustrated in Listing 5.1.

Listing 5.1: Excerpt of an interchange document exported by the BigBlueButton environment

```
<?xml version="1.0" encoding="UTF-8"?>
<recording meeting_id="6e35e3b2778883f5db637d7a5dba0a427f692e91-1367694383645">
  <metadata description="3ª tentativa" email="Diogo Pedrosa"
    title="English 101" meetingId="English 101" meetingName="English 101"/>
  <event timestamp="1367694388025" module="PARTICIPANT"
    eventname="ParticipantJoinEvent">
    <userId>31</userId>
    <status>{raiseHand=false, hasStream=false, presenter=false}</status>
    <role>MODERATOR</role>
  </event>
  (...)
  <event timestamp="1367694398716" module="VOICE"
    eventname="StartRecordingEvent">
    <bridge>77414</bridge>
    <recordingTimestamp>1367694398715599</recordingTimestamp>
    <filename>(...)-1367694398710.wav</filename>
  </event>
  <event timestamp="1367694399750" module="VOICE"
    eventname="ParticipantTalkingEvent">
    <bridge>77414</bridge>
    <participant>10</participant>
    <talking>true</talking>
  </event>
  <event timestamp="1367694401690" module="VOICE"
    eventname="ParticipantTalkingEvent">
    <participant>10</participant>
    <bridge>77414</bridge>
    <talking>false</talking>
  </event>
  (...)
</recording>
```

The interchange document illustrated in Listing 5.1 structures the presentation in "events" that occur in time. Each event has a timestamp in wall clock time and has been generated by a specific module. Additionally, each event has variable attributes, depending on which module is used. By taking advantage of timestamps, module identifiers and event attributes, the synchronization of

all media elements in the session can be reconstructed. In the example, the event corresponding to the start of a media element is reported at lines 12-17 (the event corresponding to the end of the media element occurs further in the document has been suppressed in this excerpt). Additionally, events corresponding to annotations, such as voice activity (lines 24-29) and participant activity (lines 5-10) are also included. This is document is converted into a integration document, of which an example is demonstrated in Listing 5.2.

Listing 5.2: Excerpt of an integration document generated via ingestion process

```
1  {
2    "begin": 1367694388.025,
3    "date_created": 1367694388.025,
4    "date_recorded": 1367694388.025,
5    "date_updated": 1367694388.025,
6    "end": 1367696188.795,
7    "groups": [],
8    "id": "6e35e3b2778883f5db637d7a5dba0a427f692e91-1367694383645",
9    "media_elements": {
10     (...)
11     "6e35e3b2778883f5db637d7a5dba0a427f692e91-1367694383645-1367694398710": {
12       "begin": 1367694398.716,
13       "end": 1367696188.795,
14       "filename": "6e35e3b2778883f5db637d7a5dba0a427f692e91-1367694383645-1367694398710.mp3",
15       "session_begin": 10.69099998474121,
16       "session_end": 1800.769999980927,
17       "type": "audio",
18       "user_id": "31"
19     }
20   },
21   "metadata": {
22     "description": "3\u00aatentativa",
23     "email": "Diogo Pedrosa",
24     "meetingId": "English 101",
25     "meetingName": "English 101",
26     "title": "English 101"
27   },
28   (...)
29   "timelines": {
30     (...)
31     "voice_31": {
32       "events": {
33         "voice_31_1367694399750": {
34           "begin": 1367694399.75,
35           "behavior": "default",
36           "comment": "",
37           "end": 1367694401.69,
38           "session_begin": 11.72499990463257,
39           "session_end": 13.66499996185303,
40           "user_id": "31"
41         },
42 (...)
43 }
```

I+WaC-Editor adopts JSON (JavaScript Object Notation) [IETF, 2006] as an integration

format, a structured document format considered concise when compared to more verbose alternatives, such as XML. As the integration document is also used for all management purposes in the architecture, it includes management attributes, such modification timestamps. What is worth stressing is the representation of media elements, as composed of identifiers, reference to the corresponding file, original time (lines 12-13) and derived time (lines 15-16). Additionally, metadata-oriented events, logged by the capture environment, are also represented (lines 31-40).

Another possibility for an integration document to be created is via ingestion of Web videos. This is a special case of ingestion in which, instead of having an interchange document as output, it is given a reference to a video on the Web. This situation occurs because I+WaC-Editor also includes functionalities for annotating web videos. In this case, instead of converting an interchange document, an integration document is generated from a template, then filled with metadata from the ingested video. Additionally, the video file is not ingested into the architecture, being retrieved from the original location by the user agent upon rendering a generated multimedia document. After the integration document has been generated and stored in the session repository, other processes can take advantage of it for manipulating documents.

5.3 Generating multimedia documents

The generation of a multimedia document to be served to the client consists on two processes: the generation of a spatial document and the generation of a temporal document. The integration of the two processes can be seen as one of automatic authoring, in which the authoring specifications are represented in the integration document.

The spatial layout generation process fills a predefined spatial template by traversing the list of media elements in the integration document and filling the template variables. Different templates can used upon client request. In I+WaC-Editor spatial documents are generated in HTML5 via a regular template language.

The temporal layout generation process builds an ActiveTimesheets document by traversing the integration document while populating the DOM of the temporal document. As the ActiveTimesheets document must be bound the spatial document via structure-specific expressions, the temporal layout process is subordinate to the spatial layout process; consequently, if a different template is applied to the document, as default a different temporal layout is applied as well. After the DOM of the temporal layout is constructed, it is serialized to an XML document.

It is worth stressing that, upon a multimedia document generation process, only the spatial document is initially served to the client. The temporal document will be served only after rendering the spatial document, when the ActiveTimesheets engine, in the client, will asynchronously fetch the timesheet informed in the spatial document. Only at this moment the temporal layout is generated and served to the client.

A document generated via these processes follows the general structure already presented in previous Examples in this dissertation, i.e. along the lines of Listing 3.1, in the case of a spatial

document, and Listing 3.2, in the case of a temporal document. As these two examples have been thoroughly discussed in previous Chapters, details of these documents will not be reproduced here.

5.4 Client-side document management

Managing documents on the client consists on performing file manager -like operations with documents stored in the I+WaC editor infrastructure. Figure 5.3 illustrates one of the interfaces for document management.

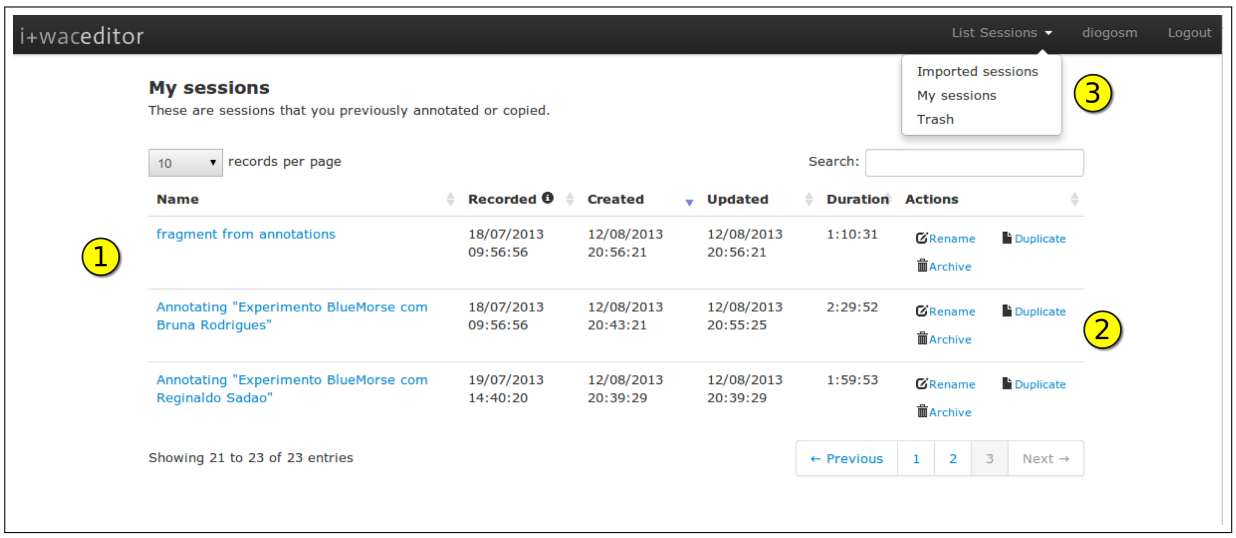


Figure 5.3: Managing personal documents in the I+WaC-Editor client

The interface in Figure 5.3 allows users to access the document stored in their account, whose list if the component (1) in the Figure. For each document, users can rename, duplicate or send a document to the trash (2). However, a document cannot be permanently deleted. From a menu at the top of the webpage (3), users can switch another administrative interfaces, such as the "imported session" list and the "trash" session list. The list demonstrated in the Figure is the "my session" list, which corresponds to sessions a user already start to annotate.

For sessions that have never been annotated, they need to be copied from imported list, illustrated in Figure 5.4. An imported session cannot be annotated directly, as a consequence, once a user chooses a session from the list (1), then a copy of the session is made in the "my sessions" list. Additionally, the interface allows to import a new session as video from the Web: this is triggered via a specific button (2).

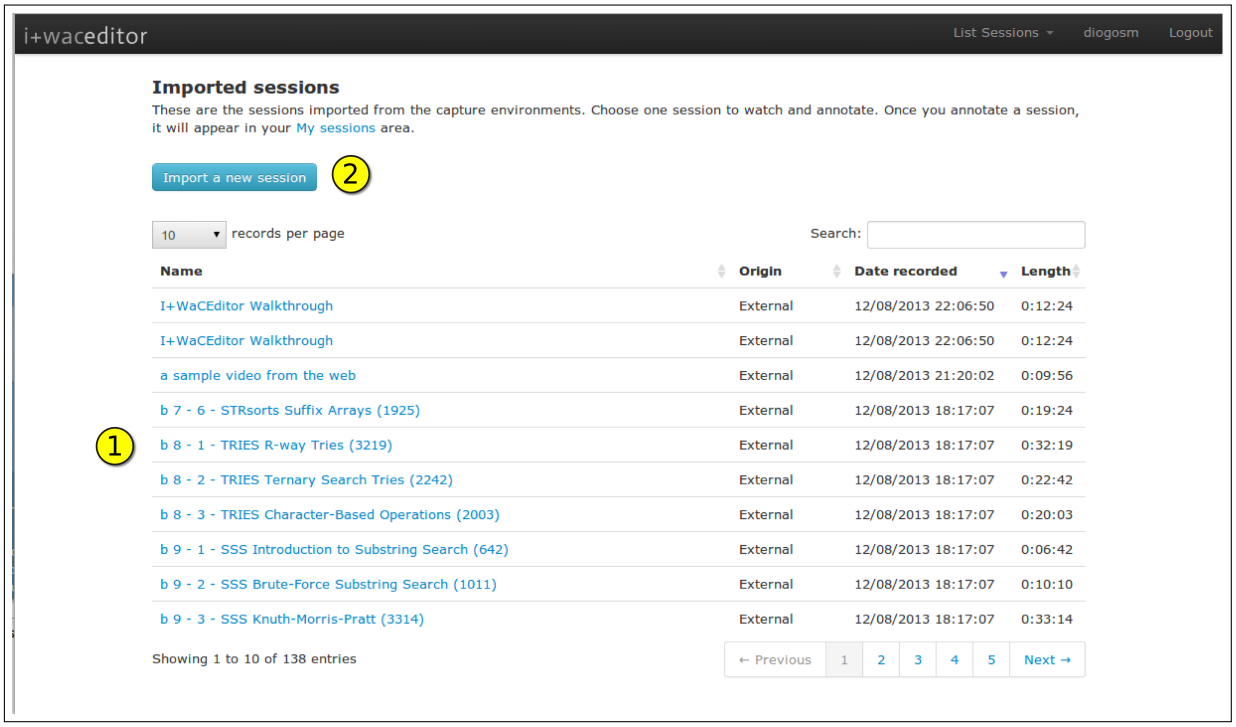


Figure 5.4: Managing imported documents in the I+WaC-Editor client

5.5 Document playback

The Editor has two playback modes: the simple mode and the annotation mode. The simple mode presents only playback controls whereas the annotation mode complements the interface with annotation visualization that can be used for browsing purposes.

The simple playback mode, depicted in Figure 5.5, is represented by multiple regions, one of them being the document rendering area (1), where the media elements are spatially laid out: in the example, these are slides, chat messages and videos. Another region is the control are (2), where several buttons are provided for presentation control (from left to right): play, seek backward, seek forward, play only selected fragments (this control will be illustrated in the next Section), stop, speed, volume, time counter and time slider. Additionally, the annotations button (3) toggle the visualization of annotations by triggering the annotation mode. This mode is discussed in the next Section.

5.6 Document browsing

An important issue when watching long sessions is to count of on a group of browsing features which allow to navigate the session is customized ways. In I+WaC-Editor, these features are provided in the annotation mode. The most basic feature is represented by the interactive timelines (1). Each interactive timeline represents an interactor timeline and is composed by a label and set of non-overlapping intervals. Besides visualizing the temporal

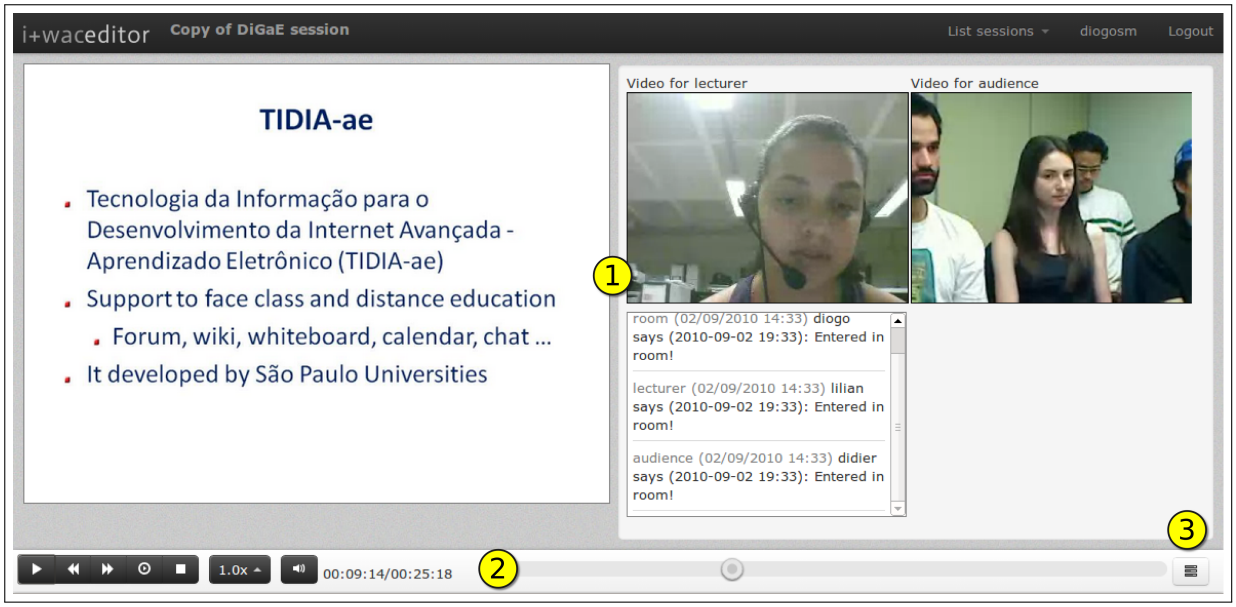


Figure 5.5: I+WaC-Editor simple playback mode

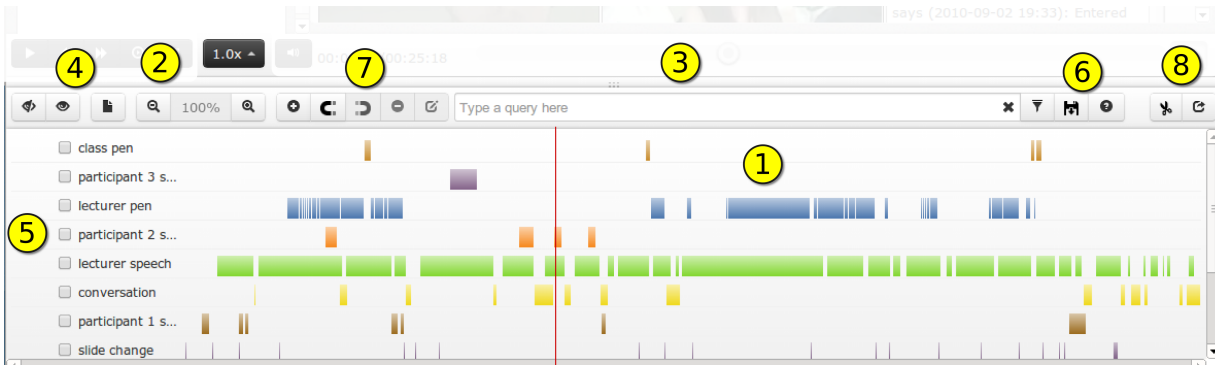


Figure 5.6: Browsing and annotation functionalities in the I+WaC-Editor

distribution of annotations in the duration of the session, when an annotation is clicked, it jumps directly to the begin of the interval of the annotation. This is achieved via the ActiveTimesheets linking functionality, taking advantage of its integration with Media Fragments (as discussed in Section 3.5.2). Additionally, timelines can be zoomed (2) in case their spatial mapping is too thin (as in the case of too long sessions). Another useful operation is reordering timelines: this can be achieved by dragging a timeline to a new position. The speed of the whole presentation can be manipulated via a specific control. This change is enforced via a live editing operation in the outermost element of the presentation, by changing the `speed` of the element. As a consequence of this operation, the schedule and sampling of presentation is modified according to timing manipulation procedures documented in Section 3.4.2.

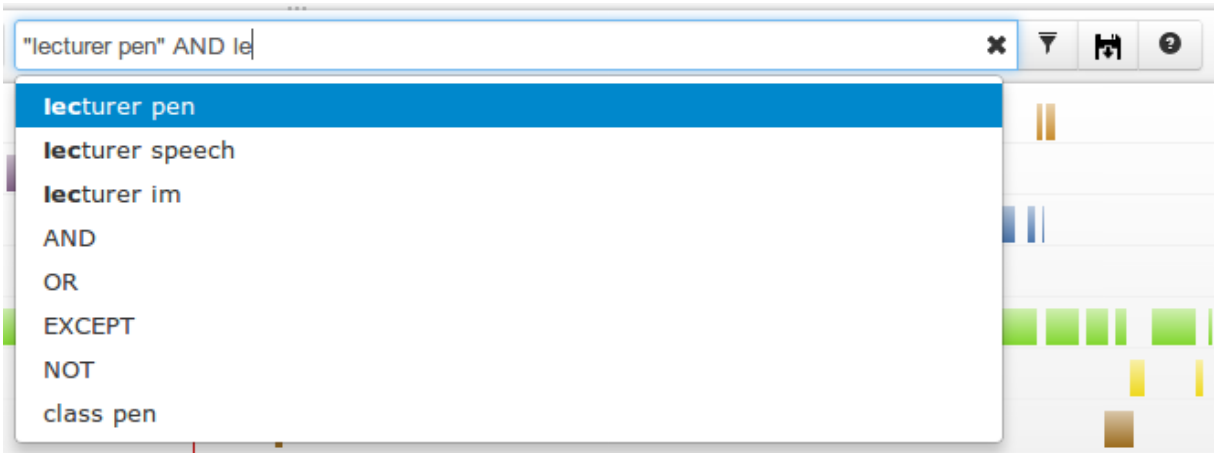


Figure 5.7: Advanced query mode in I+WaC editor

Another feature in the browsing mode is the annotation filtering box (3) 5.7, which allows the combination of annotations using the temporal join operators discussed in Section 4.5. In the notation used in the tool, the set-based joining (*intersection*, *union*, *difference*, *complement*) are mapped to their logical equivalents (*and*, *or*, *except* and *not*). When a query is processed the resulting intervals are rendered in red in the presentation playback slider. Additionally, upon entering expressions, the component auto-completes with suggestions of timelines available in the document and possible operators. Naturally, this query mode is advanced and it is not supposed to be used by all target users. As an alternative, a simple query mode is available, using drag and drop functionality, and icons as surrogates to operations: this query mode has been presented elsewhere [Martins and Pimentel, 2012].

5.7 Annotating and enriching documents

Annotating a document in the I+WaC-Editor consists on extending the existing annotations that were defined upon session ingestion. This can be done in two ways: creating new timelines and creating new annotations. A new blank timeline can be created via a specific button (4),

giving a label for the new timeline. Another possibility is to create, instead of a blank timeline, a timeline derived from a "saved query" (6). This type of timeline will consist of all the intervals resulting of an expression entered in the query box. Timelines can be removed and renamed via specific controls (5).

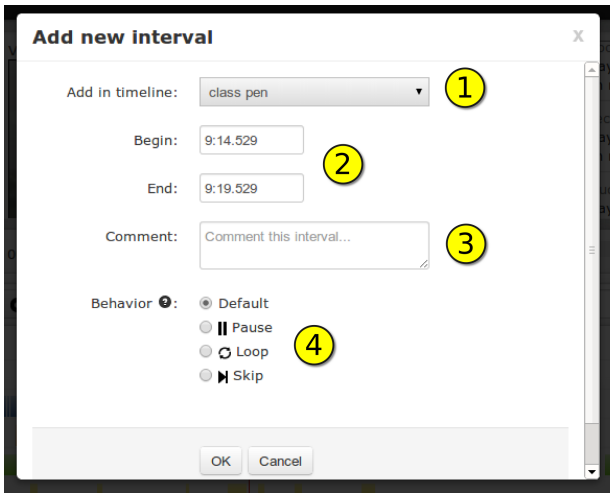


Figure 5.8: Annotation creation dialog

Regarding annotations, they can be created using a number of options (7): from a dialog, where a dialog asks for information about the interval; or with the magnet operation, which allows creating intervals while the document is playing by pressing a button to begin and button to end the interval. When an interval is created via a dialog (Figure 5.8), several options can be assigned: the timeline the annotation will be assigned to (1); the interval of the annotation (beginning at the current playback mode) (2); the comment to be associated with the annotation (3); the behavior of the annotation (which correspond to the effect operator discussed in Section 4.6.1).

All manipulations of timelines and annotations are performed via live editing, as a consequence the annotations can be used immediately, for instance, to seek to in the document. Live editing is also useful in the case in which an annotation has an effect operator associated with it. In this case, the annotation adopts a icon indicating the type of behavior and the effect can be noticed as soon as the interval of the annotation is traversed.

5.8 Generating new document versions

This functionality consists on a mapping of the slicing operator, discussed in Section 4.6.3. In order to execute this operation, a user must select one or more timelines (by "checking" their combo boxes) and press the fragment button (8), represented with a scissor icon. When the button is pressed, a dialog appears requesting name of the new document that will be generated and asking whether to include the participating annotations in the new document or not. The option of not including annotations is important in the cases where the annotations have a private

connotation. In summary, differently from the enrichment operations (creating new timelines and intervals), the *fragment* operation is an editing operation: consequently, it always generates a new document that must be opened separately in order that the results can be perceived.

5.9 Document sharing

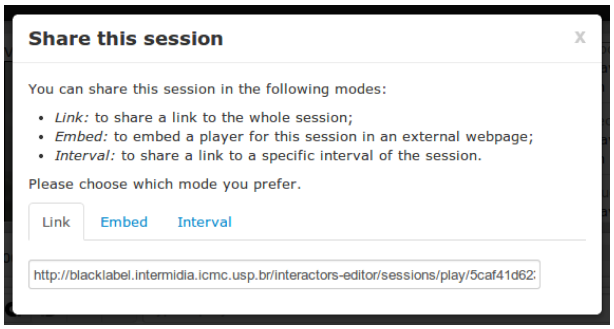


Figure 5.9: Dialog for a sharing a document

Once a presentation has been analyzed (via simple playback and annotation modes), annotated (via annotation and enrichment operations) and edited (via the fragment extraction operation), the resulting document can be shared. In fact, any document in I+WaC-Editor can be shared, in different modes. In order to share a document, the share button is pressed (8) and a new dialog displays the sharing options (Figure 5.9). As the I+WaC-Editor is a tool whose access is controlled via user accounts, the sharing option allows the audience to interact with a playback-only interface, in which all editing controls and document management functionalities are unavailable. The document can be shared in playback-only mode as a whole either in full screen (by providing an URL) or by embedding the player into a web page: this can be done by including an HTML fragment generated by the tool. Another sharing option is by selecting an annotation in the tool and sharing based on the selected annotation: the effect is the presentation, when played, will begin with an offset to the begin of the annotation. This particular functionality is realized via Media Fragments integration in the linking module, discussed in Section 3.5.2.

5.10 Final remarks

This Chapter presented the I+WaC-Editor tool, its most important functionalities and how it integrates the contributions of the previous Chapters. In particular, it has been demonstrated how the live editing, reuse and linking features of ActiveTimesheets have been mapped to several interaction techniques in the authoring tool. Additionally, it has been demonstrated how the IAMmDocT operators have been integrated in a functionality to extract fragments of a document based on annotations.

An interesting future addition to the tool is support for multimodal annotations given that, currently, the tool only support text-based access time annotations. A useful feature would be to incorporate multimodal annotations on it, in order to provide richer annotation functionalities. This could be done by taking advantage of the new extensions for media capture available in HTML5 (in order to avoid plugins).

In many cases, amateurs are unwilling to use query-like operators to manipulate annotations, because it is a low level formalism. Even in the alternative, simple mode provided by I+WaC-Editor, the notion of set-based operations might not be properly understood by users. A possible solution for this problem is not to delegate to the user the composition of annotations. this could be enabled via investigation of methods to automatically derive higher level events from low level events. In many situations there is a gap between a high level user information need and the low level events that are indexed in the session. Even though the user-in-the-loop approach adopted in the I+WaC-Editor via the temporal join operators have potential of helping users to bridge this gap, we believe that not all users are willing to engage in formulating query-like information requests. In this direction, we plan to build upon the formalization of the annotation and composition operators in order to develop methods for pattern mining of higher level events and further develop simpler navigation mechanisms for multimedia sessions. This effort will be greatly helped by the execution of user studies to properly evaluate the interaction issues with the multimedia editor.

Another interesting feature for the tool would be collaborative access to the sessions, synchronously and asynchronously. In the synchronous case, this would allow that multiple distribute researchers could annotate a session while discussing it via a separate communication channel. In the asynchronous case, multiple researchers could collaborate on the document in different time spans. An interesting development of this scenario, is that it would require both client-side and server-side live editing, so that the document is kept consistent on both ends.

Chapter 6

Discussion and conclusion

6.1 Synthesis of the contributions

The most important contributions of this dissertation are the following:

- The ActiveTimesheets language and engine

  The dissertation contributed with a SMIL-Based language designed to model web-based active multimedia documents. The language, combined with its implementation as an engine, provides a number of important extensions, such as live editing, extended linking, media fragments support and reuse of elements and fragments. In order to demonstrate this extensions, an improved formalization of the timegraph model has been provided, detailing its various processes, such as scheduling, sampling and resolution. Such contribution has several important implications. First, it has potential to improve the adoption of SMIL as long it enables scripting of this language, a feature until now unsupported. Second, reuse features have several syntax advantages, potentially reducing the verbosity of the final document. Third, linking functionality, as realized in the ActiveTimesheets model, enables a better integration of SMIL in HTML-based host languages.

- The Interactors Algebra for Multimedia Document Transformations (IAMmDocT)

  The dissertation contributed with an algebra for performing transformation in multimedia documents. As a basis to define the operators that compose the algebra, a conceptual model, based on interaction events, has been defined. Having this basic component, a group of operators have been defined, encompassing operators for annotation, enrichment and transformation of multimedia documents. The integration of these operators with the ActiveTimesheets engine has also been demonstrated. This contribution has secondary implications, for instance, for automatic and semi-automatic authoring, as far as it provides a semantics to describe document transformations.

- The Interactors+WaC authoring tool

In order to realize the previous contributions, the dissertation also presented the Interactors+WaC tool, that provides several authoring functionalities to be applied over multimedia documents. The tool can operate over documents imported from capture environments or general videos on the Web. Its interaction techniques map the potential of the Active-Timesheets language (as it extensively uses live editing, linking and reuse features) and the IAMmDocT (given that it allows temporal join queries over annotations and extraction of fragments based on annotations). More importantly, the tool demonstrates how the components reported in this dissertation fit together in a practical authoring tool.

6.2 Limitations

A critical analysis of the contributions reported on this dissertation reveals several issues over which the research can be improved on the quest for greater applicability and generalizability. Such issues are structured mostly under the themes of semantic content of multimedia annotations and expression power of multimedia operators. The following list presents the most important limitations of the research and their associated demands:

- **Generation of content-based annotations**: currently the kinds of annotations regarded in the research are mostly obtained via physical and virtual sensors (e.g. via instrumentation of the tools for capture and access) or via manual crafting (intentional annotations). Examples of these annotations are marks of slide changes, tracking of operations using electronic ink, identification of participants, etc. Even though such kinds of annotations have shown their value for several types of browsing and authoring use cases, the experiments have demonstrated the need to count on additional types of annotations, representing intentional user actions or multimedia events, the obtainment of which will require content analysis of the underlying multimedia elements. Examples of compelling content-based annotations are gestures and actions, voice segmentation, text transcription, just to name a few.

- **Improvements on the semantic level of annotations**: still related to semantics of annotations, in many cases there is gap between the semantic content of an annotation applied over a media fragment and the meaning a user assigns to the same fragment. Even though this semantic gap is widely dependent on a multitude of contextual factors, the reliance on annotations with low-level semantics is not expected to contribute much on bridging this gap. This is particularly noticeable in situations where the user have an information need that comprises relationships between several low level annotations (e.g. causal, structural and contextual relationships, just to name a few). Such requirements create demands for derivation of annotations with semantic content of a higher level, for instance: identification of discussions and arguments, attention, confusion, among others. Accomplishing such "semantic lifting" has great potential for making the access and reuse of the information more meaningful to the user.

- **Enhancements on expressive power for the operators**: the family of operators at its current state has potential to support *selection* and *fragmentation* tasks over media elements, in simple mode (in the case of the operators which select specific families of annotations) or in complex mode (in the case of the algebraic operations for composite selection). However, from the standpoint of presentation *composition*, currently the only composition operation is the sequential scheduling of media fragments, which imposes a considerable limit on expressive power while authoring. Provided these limitations, there is a need to define more operators to specify different types of composition relationships, taking for instance spatial relationships (e.g. for representation of selection restrictions over visual media elements), causal relationships (e.g. to represent dependence restrictions between several annotated events), associative relationships (e.g. to establish richer alternatives for linking media elements and consequently influence their presentation), among others.

6.3 Future work

The contributions of the dissertation establish a basis upon which several future developments can be built. In this section we highlight the most important issues for future research, structuring them into two aspects that can be investigated in conjunction: *i)* the automatic annotation of events of interest; and *ii)* the development of operators that facilitate semi-automatic authoring of multimedia information via these annotations.

1. **Investigation of content-based annotation methods**.

   Involves research on content analysis with purposes of recognizing basic events of interest from complex multimedia objects, encompassing the orchestrated application of computer vision, speech analysis and text analysis. Research on this direction implies a numbers of challenges, most of them related to the difficulties of extracting semantic information from multimodal streams. Taken alone, as an example, the problem of recognizing objects, actions and events from visual streams, it is an intense research field in the multimedia community, with several open problems. For the cases where solutions are reported, most of them are highly specialized to datasets from specific domains, such as news reports [Snoek and Worring, 2009], sports tournaments [Hannon et al., 2011], surveillance [Poppe, 2010], and so on. Provided such well known challenges and limitations, it is important, for purposes of turning the problem manageable, that future work on this theme stands upon strong rationale on the methods that are most suitable for the problem at hand, namely transparent annotation and authoring of multimedia information.

2. **Investigation of methods for derivation of events with high level semantics**

   Consists on the development of methods for abstraction of complex events from content-based annotations representing objects, actions and basic events. Complex events are

abstractions more difficult to represent, as they encompass higher level semantics, generally associated with several basic events. As a consequence, their recognition, analogously to recognition of basic events, is also affected by low generalizability. The literature suggests the opportunity of combining events of low level semantics in order to generate events of higher level semantics, for instance by detecting patterns of composition and by applying knowledge from specific domains [Jiang et al., 2012]. For instance, hypotheses on the occurrence of these patterns have been investigated in the context of the analysis of voice expressions to derive dialog acts [Yu et al., 2012] and the discrimination among brainstorming sessions and decision making sessions in meetings [Jayagopi et al., 2012]. Analysis of event composition has also been explored in the context of interviews between physicians and patients [Koyama et al., 2010]. Much of this research has been based on machine learning methods and data mining in order to derive complex events, since a set of basic events is available. Such results suggest the viability of combining low level events and detailed knowledge representation from specific domains in order to discriminate high level representations in multimedia information.

3. **Refinement and evolution of the multimedia operators**

   The availability of content-based annotations opens up opportunities for defining new operators that encompass the richer semantics enabled by this information. This direction involves the definition of new operators that enable not only sequential composition, but also other modes, either low-level composition methods (e.g. temporal-parallel and spatial composition variations) and high-level compositions (e.g. structural and causal compositions). Such operators will enable reconfiguration of fragments from multimedia sessions, that yield new versions of these sessions, potentially enriched with added-value content from users. Even though research on low-level composition relations, which represent spatial and temporal relationships between media elements, are reasonably mature in context of multimedia authoring [Bulterman and Hardman, 2005], the explicit specification of these relations, via authoring tools designed for amateur users, might impose challenges on user interaction: these problems have motivated considerable research in the context of interaction techniques for multimedia authoring [Meixner et al., 2012]. Similar evidences are observed regarding the use of associative relationships, mostly in the case of the creation of hypermedia links in hypervideo research [Sadallah et al., 2012]. Additionally, from the standpoint of high-level composition relations, research on automatic authoring has demonstrated the need to specify causal relations (e.g. in order to establish cause-effect relationships among media elements) and structural relations (e.g. in order to specify that groups of media elements participate in a semantic aggregation), all of them as requirements to generate consistent narratives [de Lima et al., 2012; Zsombori et al., 2011].

4. **Incorporation of methods for capture-time live annotations**

Complementarily to applications in education and qualitative research, other potential domains for application are worth exploring. One of these domains is journalism, where the capture, analysis and reuse of multimodal information is recurrent. As with other disciplines oriented to data collection from humans, interviews are in journalism a primary technique. A common complaint of journalists is the difficulty of reviewing non-textual (audio or video) recordings, as it represents a big time investment, in many cases twice the recording time [Cohen et al., 2011]. Speech recognition might alleviate this problem to a certain extent, but whereas speech recognition will potentially help to document the linguistic content of the recording, the annotations of the reporter, collected while capturing the interview, are commonly kept as separate documents, potentially out of sync from the main recording. Live intentional annotations to discriminate important moments in a recording have been largely explored in domains such as lectures and meetings, with associated results positively demonstrating the utility of these annotations for highlighting important moments in the captured experience [Hsu et al., 2012; Teevan et al., 2012]. But, on the multimedia authoring side, these annotations have been explored mostly on the automatic authoring of presentations for access purposes, for instance to generate an unedited session with the main video synchronized with slides and timelines of annotations. These annotations have not been much explored as a cue for supporting users on actively editing a multimedia piece. The combination of live annotations and good authoring metaphors has great potential to assist amateur-professional users from several domains on the tasks of browsing, reusing and analyzing recorded multimedia, in data collection or research tasks.

6.4 Publications

This Section lists the publications originated from the PhD research. These are classified in directly related, i.e., those derived in the theme of the dissertation, and indirectly related, i.e., those originated from extra projects like, for instance, class projects.

6.4.1 Directly related to the dissertation

Journal papers

1. **Martins, D. S.**, Vega-Oliveros, D. A., & Pimentel, M. G. C. (2011). *Automatic authoring of interactive multimedia documents via media-oriented operators*. ACM SIGAPP Applied Computing Review, 11(4), 26–37. doi:10.1145/2107756.2107759

Book chapters

2. **Martins, D. S.**, & Pimentel, M. G. C. (2011). *TV digital e a EAD*. In F. M. Litto & M. M. M. Formiga (Eds.), Educação a Distância - o Estado da Arte (1st ed., pp. 26–34). Pearson Education.

Full papers in conferences

3. Vega-Oliveros, D. A., **Martins, D. S.**, & Pimentel, M. da G. C. (2010). *Interactors: operators to automatically generate interactive multimedia documents from captured media*. In Proceedings of the XVI Brazilian Symposium on Multimedia and the Web (Webmedia 2010) (pp. 163–170). URL: `http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=9909`

4. **Martins, D. S.**, & Pimentel, M. G. C. (2011). *End-user ubiquitous multimedia production: process and case studies*. In Proceedings of 4th International Conference on Ubi-media Computing (U-Media 2011) (pp. 197–202). DOI: 10.1109/U-MEDIA.2011.18

5. Vega-Oliveros, D. A., **Martins, D. S.**, & Pimentel, M. G. C. (2011). *Media-oriented operators for authoring interactive multimedia documents generated from capture sessions*. In Proceedings of the 26th Symposium on Applied Computing (SAC 2011) (pp. 1267–1272). DOI: 10.1145/1982185.1982461

6. Vega-Oliveros, D. A., Oliveira, L. S., **Martins, D. S.**, & Pimentel, M. G. C. (2011). *Viewing by interactions: media-oriented operators for reviewing recorded sessions on TV*. In Proceddings of the 9th International Conference on Interactive Television (EuroITV '11) (pp. 213–222). DOI: 10.1145/2000119.2000163

7. **Martins, D. S.**, & Pimentel, M. G. C. (2012). *Browsing interaction events in recordings of small group activities via multimedia operators*. In Proceedings of the 18th Brazilian Symposium on Multimedia and the Web - WebMedia '12 (pp. 245–252). Best paper in the Multimedia Track. DOI: 10.1145/2382636.2382689

Short papers in conferences

8. Vega-Oliveros, D. A., **Martins, D. S.**, & Pimentel, M. G. C. (2010). *"This conversation will be recorded": automatically generating interactive documents from captured media*. In Proceedings of the 10th ACM Symposium on Document Engineering (DocEng '10) (pp. 37–40). DOI:10.1145/1860559.1860568

6.4.2 Indirectly related to the dissertation

Full papers in conferences

9. Oliveira, L. S., **Martins, D. S.**, & Pimentel, M. G. C. (2009). *EducaTV: uma abordagem para execução de objetos de aprendizagem em TVDi compatível com Ginga-J*. In Proceedings of the XX Simpósio Brasileiro de Informática na Educação - SBIE 2009 (Vol. 1, pp. 1–10). Florianópolis - SC, Brazil. URL: `http://www.niee.ufrgs.br/eventos/SBIE/2009/conteudo/artigos/completos/62113_1.pdf`

10. **Martins, D. S.**, Oliveira, L. S., & Pimentel, M. G. C. (2010). *Designing the user experience in iTV-based interactive learning objects*. In Proceedings of the 28th ACM International Conference on Design of Communication - SIGDOC '10 (pp. 243–250). DOI:10.1145/1878450.1878491

Short papers in conferences

11. Oliveira, L. S., **Martins, D. S.**, Goularte, R., & Pimentel, M. G. C. (2009). *EducaTV: an architecture to access educational content through IDTV*. In Proceedings of the XV Brazilian Symposium on Multimedia and the Web - WebMedia '09 (pp. 1–4). DOI:10.1145/1858477.1858525

Bibliography

G. D. Abowd. What next, ubicomp?: celebrating an intellectual disappearing act. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing - UbiComp '12*, pages 31–40, 2012.

G. D. Abowd, C. G. Atkeson, J. Brotherton, T. Enqvist, P. Gulley, and J. LeMon. Investigating the capture, integration and access problem of ubiquitous computing in an educational setting. In *ACM CHI'98*, pages 440–447, 1998.

G. D. Abowd, E. D. Mynatt, and T. Rodden. The human experience [of ubiquitous computing]. *IEEE Pervasive Computing*, 1(1):48–57, 2002.

B. Adams, S. Venkatesh, and R. Jain. IMCE: Integrated media creation environment. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(3):211–247, 2005.

B. Adams, S. Greenhill, and S. Venkatesh. Temporal semantic compression for video browsing. In *Proceedings of the 13th international conference on Intelligent user interfaces*, pages 293–296, 2008.

J. Adcock, M. Cooper, L. Denoue, H. Pirsiavash, and L. A. Rowe. TalkMiner: a lecture webcast search engine. In *Proceedings of the 18th ACM International Conference on Multimedia*, pages 241–250, 2010.

M. Agosti and N. Ferro. A formal model of annotations of digital content. *ACM Transactions on Information Systems*, 26(1):1–57, 2007.

J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

M. Ames and M. Naaman. Why we tag: motivations for annotation in mobile and online media. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 971–980, 2007.

W. Bailer and P. Schallauer. Metadata in the Audiovisual Media Production Process. In M. Granitzer, M. Lux, and M. Spaniol, editors, *Multimedia Semantics — The Role of Metadata (Studies in Computational Intelligence Series, v. 101)*, volume 101, chapter Metadata i, pages 65–84. Springer, 2008.

A. Behera, D. Lalanne, and R. Ingold. DocMIR: An automatic document-based indexing system for meeting retrieval. *Multimedia Tools and Applications*, 37(2):135–167, 2007.

F. Berthouzoz, W. Li, and M. Agrawala. Tools for placing cuts and transitions in interview video. *ACM Trans. Graph.*, 31(4):1–8, 2012.

M. Bertini, A. Bimbo, A. Ferracani, L. Landucci, and D. Pezzatini. Interactive multi-user video retrieval systems. *Multimedia Tools and Applications*, pages 1–27, 2011.

E. Bertino and E. Ferrari. Temporal synchronization models for multimedia data. *IEEE Transactions on Knowledge and Data Engineering*, 10(4):612–631, 1998.

P. Bertolotti and O. Gaggi. A study on multimedia documents behavior: a notion of equivalence. *Multimedia Tools and Applications*, 33(3):301–324, 2007.

G. Blakowski and R. Steinmetz. A media synchronization survey: reference model, specification, and case studies. *IEEE Journal on Selected Areas in Communications*, 14(1):5–35, 1996.

H. Blanken, L. Feng, M. van Keulen, and H. E. Blok. *Multimedia Retrieval, chapter: Introduction*, pages 1–22. Data-Centric Systems and Applications. Springer Berlin Heidelberg, 2007.

A. Bossi and O. Gaggi. Enriching SMIL with assertions for temporal validation. In *Proceedings of the 15th international conference on Multimedia - MULTIMEDIA '07*, pages 107–116, 2007.

M.-M. Bouamrane and S. Luz. Meeting browsing. *Multimedia Systems*, 12(4-5):439–457, October 2006.

S. Bouyakoub and A. Belkhir. SMIL builder: An incremental authoring tool for SMIL documents. *ACM Trans. Multimedia Comput. Commun. Appl.*, 7(1):1–30, 2011.

D. C. Bulterman, J. Jansen, K. Kleanthous, K. Blom, and D. Benden. Ambulant : A Fast , Multi-Platform Open Source SMIL Player. In *Proceedings of the 12th annual ACM international conference on Multimedia - MULTIMEDIA '04*, pages 492–495, 2004.

D. C. A. Bulterman and L. Hardman. Structured multimedia authoring. *ACM Trans. Multimedia Comput. Commun. Appl.*, 1(1):89–109, 2005.

D. Cabral and N. Correia. Videoink: a pen-based approach for video editing. In *Adjunct proceedings of the 25th annual ACM symposium on User interface software and technology*, pages 67–68, 2012.

D. Cabral, J. . Valente, J. . Silva, U. A. Ao, C. Fernandes, and N. Correia. A creation-tool for contemporary dance using multimodal video annotation. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 905–908, 2011.

A. Carlier, G. Ravindra, V. Charvillat, and W. T. Ooi. Combining content-based analysis and crowdsourcing to improve user interaction with zoomable video. In *Proceedings of the 19th ACM international conference on Multimedia*, MM '11, pages 43–52, 2011.

R. G. Cattelan. *Captura e acesso na produção, distribuição, apresentação e extensão de conteúdo multimídia*. Thesis (phd), Universidade de São Paulo, 2009.

R. G. Cattelan, C. Teixeira, R. Goularte, and M. G. C. Pimentel. Watch-and-comment as a paradigm toward ubiquitous interactive video editing. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4(4):1–24, 2008a.

R. G. Cattelan, C. Teixeira, H. Ribas, E. Munson, and M. G. C. Pimentel. Inkteractors: interacting with digital ink. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1246–1251, 2008b.

F. Cazenave, V. Quint, and C. Roisin. Timesheets.js: when SMIL meets HTML5 and CSS3. In *Proceedings of the 11th ACM Symposium on Document Engineering*, pages 43–52, 2011.

A. Celentano, O. Gaggi, and M. L. Sapino. Retrieval in multimedia presentations. *Multimedia Systems*, 10(1):72–82, 2004.

J. R. Cerqueira Neto, R. C. Mesquita Santos, C. S. Soares Neto, and M. M. Teixeira. Eventline: representation of the temporal behavior of multimedia applications. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web - WebMedia '12*, pages 215–222, 2012.

P. Cesar and K. Chorianopoulos. The Evolution of TV Systems, Content, and Users Toward Interactivity. *Found. Trends Hum.-Comput. Interact.*, 2(4):279–373, 2009.

P. Cesar, D. C. A. Bulterman, J. Jansen, D. Geerts, H. Knoche, and W. Seager. Fragment, tag, enrich, and send: Enhancing social sharing of video. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(3):1–27, 2009.

S. Chandra. Experiences in personal lecture video capture. *Learning Technologies, IEEE Transactions on*, 4(3):261–274, 2011.

K. Y. Cheng, S. J. Luo, B. Y. Chen, and H. H. Chu. SmartPlayer: user-centric video fast-forwarding. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 789–798, 2009.

M. G. Christel, S. M. Stevens, B. S. Maher, and J. Richardson. Enhanced exploration of oral history archives through processed video and synchronized text transcripts. In *Proceedings of the international conference on Multimedia - MM '10*, page 1333, 2010.

S. Cohen, J. T. Hamilton, and F. Turner. Computational journalism. *Commun. ACM*, 54(10): 66–71, 2011.

S. H. Cooray, H. Bredin, L. Q. Xu, and N. E. O'Connor. An interactive and multi-level framework for summarising user generated videos. In *Proceedings of the 17th ACM international conference on Multimedia*, MM '09, pages 685–688, 2009.

E. S. de Lima, B. Feijo, A. L. Furtado, C. Pozzer, and A. Ciarlini. Automatic video editing for Video-Based interactive storytelling. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 806–811, 2012.

M. Del Fabro, K. Schoeffmann, and L. Böszörmenyi. Instant video browsing: a tool for fast non-sequential hierarchical video browsing. In *Proceedings of the 6th international conference on HCI in work and learning, life and leisure: workgroup human-computer interaction and usability engineering*, pages 443–446, 2010.

M. Delest, A. Don, and J. Benois-Pineau. DAG-based visual interfaces for navigation in indexed video content. *Multimedia Tools and Applications*, 31(1):51–72, 2006.

A. K. Dey. Understanding and using context. *Personal Ubiquitous Computing*, 5(1):4–7, 2001.

N. Diakopoulos, S. Goldenberg, and I. Essa. Videolyzer: quality analysis of online informational video for bloggers and journalists. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 799–808, 2009.

P. E. Dickson, D. I. Warshow, A. C. Goebel, C. C. Roache, and W. R. Adrion. Student reactions to classroom lecture capture. In *Proceedings of the 17th ACM Annual Conference on Innovation and Technology in Computer Science Education*, pages 144–149, 2012.

B. Diemert, A. Pinzari, C. Moulin, M.-H. Abel, and M. Shawky. Mapping semantic script with image processing algorithms to leverage amateur video material in professional production. *Multimedia Tools and Applications*, 62(2):333–358, 2013.

A. Duda and C. Keramane. Structured temporal composition of multimedia data. *Multimedia Database Management Systems, International Workshop*, pages 1–13, 1995.

C. Dufour, J. C. Bartlett, and E. G. Toms. Understanding how webcasts are used as sources of information. *J. Am. Soc. Inf. Sci.*, 62(2):343–362, 2011.

ECMA-262, 2011. ECMA-262: ECMAScript Language Specification (Edition 5.1), June 2011. URL `http://www.ecma-international.org/publications/standards/Ecma-262.htm`. Accessed September, 2013.

D. Edge, J. Savage, and K. Yatani. HyperSlides: dynamic presentation prototyping. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 671–680, 2013.

P. Ehlen, M. Purver, J. Niekrasz, K. Lee, and S. Peters. Meeting adjourned: off-line learning interfaces for automatic meeting understanding. In *Proceedings of the 13th international conference on Intelligent user interfaces - IUI '08*, pages 465–474, 2008.

S. Elias, K. S. Easwarakumar, and R. Chbeir. Dynamic consistency checking for temporal and spatial relations in multimedia presentations. *Proceedings of the 2006 ACM symposium on Applied computing - SAC '06*, pages 1380–1384, 2006.

B. Encelle, M. O. Beldame, S. Pouchot, and Y. Prié. Annotation-based video enrichment for blind people: a pilot study on the use of earcons and speech synthesis. In *The proceedings of the 13th international ACM SIGACCESS conference on Computers and accessibility*, pages 123–130, 2011.

A. Engström, O. Juhlin, M. Perry, and M. Broth. Temporal hybridity: footage with instant replay in real time. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, pages 1495–1504, 2010.

A. Engström, M. Perry, and O. Juhlin. Amateur vision and recreational orientation:: creating live video together. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 651–660, 2012.

ETSI. Broadcast and On-line Services: Search, select, and rightful use of content on personal storage systems ("TV-Anytime"); Part 2: Phase 1 - System description, 2007. URL `http://www.etsi.org/technologies-clusters/technologies/broadcast/tv-anytime`. Accessed September, 2013.

R. Fagá, B. C. Furtado, F. Maximino, R. G. Cattelan, and M. d. G. C. Pimentel. Context information exchange and sharing in a peer-to-peer community: a video annotation scenario. In *Proceedings of the 27th ACM international conference on Design of communication - SIGDOC '09*, pages 265–272, 2009.

R. Fagá, V. G. Motti, R. G. Cattelan, C. A. C. Teixeira, M. d. G. C. Pimentel, and R. Fagá-Jr. A social approach to authoring media annotations. In *Proceedings of the 10th ACM symposium on Document engineering - DocEng '10*, pages 17–26, 2010.

J. Fernquist, T. Grossman, and G. Fitzmaurice. Sketch-sketch revolution: an engaging tutorial system for guided sketching and application learning. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*, pages 373–382, 2011.

G. Friedland, L. Gottlieb, and A. Janin. Narrative theme navigation for sitcoms supported by fan-generated scripts. *Multimedia Tools and Applications*, 63(2):387–406, 2013.

O. Gaggi and A. Bossi. Analysis and verification of SMIL documents. *Multimedia Systems*, 17 (6):487–506, 2011.

O. Gaggi and L. Danese. A SMIL player for any web browser. In *Proc. of the 17th International Conference on Distributed Multimedia Systems (DMS 2011)*, pages 114–119, 2011.

D. Gatica-Perez. Automatic nonverbal analysis of social interaction in small groups: A review. *Image and Vision Computing*, 27(12):1775–1787, 2009.

W. Geyer, H. Richter, and G. D. Abowd. Towards a smarter meeting Record—Capture and access of meetings revisited. *Multimedia Tools and Applications*, 27:393–410, 2005.

A. Girgensohn, F. Shipman, T. Turner, and L. Wilcox. Flexible access to photo libraries via time, place, tags, and visual features. In *Proceedings of the 10th annual joint conference on Digital libraries*, pages 187–196, 2010.

C. Gkonela and K. Chorianopoulos. VideoSkip: event detection in social web videos with an implicit user heuristic. *Multimedia Tools and Applications*, pages 1–14, 2012.

R. Goularte, J. Camacho-Guerrero, V. Inacio, R. Cattelan, M. da Graca, and C. Pimentel. M4note: a multimodal tool for multimedia annotations. In *WebMedia and LA-Web, 2004. Proceedings*, 2004.

J. Guo, C. Gurrin, and S. Lao. Who produced this video, amateur or professional? *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval - ICMR '13*, pages 271–278, 2013.

M. Haesen, J. Meskens, K. Luyten, K. Coninx, J. Becker, T. Tuytelaars, G.-J. Poulisse, P. Pham, and M.-F. Moens. Finding a needle in a haystack: an interactive video archive explorer for professional video searchers. *Multimedia Tools and Applications*, (2):331–356, 2013.

J. Hagedorn, J. Hailpern, and K. G. Karahalios. VCode and VData: illustrating a new framework for supporting the video annotation workflow. In *Proceedings of the working conference on Advanced visual interfaces - AVI '08*, pages 317–321, 2008.

J. Hannon, K. McCarthy, J. Lynch, and B. Smyth. Personalized and automatic social summarization of events in video. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, pages 335–338, 2011.

L. Hardman, v. Obrenović, F. Nack, B. Kerhervé, and K. Piersol. Canonical processes of semantically annotated media production. *Multimedia Systems*, 14(6):327–340, 2008.

B. Haslhofer, R. Sanderson, R. Simon, and H. Sompel. Open annotations on multimedia web resources. *Multimedia Tools and Applications*, pages 1–21, 2012.

J. Hauglid and J. Heggland. Savanta—search, analysis, visualisation and navigation of temporal annotations. *Multimedia Tools and Applications*, 40(2):183–210, 2008.

M. Hildebrand and J. van Ossenbruggen. Linking user generated video annotations to the web of data. In *Proceedings of the 18th international conference on Advances in Multimedia Modeling*, MMM'12, pages 693–704, 2012.

B. Höferlin, M. Höferlin, D. Weiskopf, and G. Heidemann. Information-based adaptive fast-forward for visual surveillance. *Multimedia Tools and Applications*, 55(1):127–150, 2010.

R. Hong, J. Tang, H. K. Tan, C. W. Ngo, S. Yan, and T. S. Chua. Beyond search: Event-driven summarization for web videos. *ACM Trans. Multimedia Comput. Commun. Appl.*, 7, Dec. 2011.

F. Hopfgartner, T. Urruty, R. Villa, and J. Jose. Facet-based browsing in video retrieval: a simulation-based evaluation. In *Proceedings of the 15th Multimedia Modeling Conference*, pages 472–483, 2009.

Y.-C. Hsu, T.-S. Jeng, Y.-T. Shen, and P.-C. Chen. SynTag: a Web-based platform for Labeling Real-time Video. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work - CSCW '12*, pages 715–718, 2012.

J. Huber, J. Steimle, and M. Mühlhäuser. Toward more efficient user interfaces for mobile video browsing: an in-depth exploration of the design space. In *Proceedings of the international conference on Multimedia*, pages 341–350, 2010.

IETF. Uniform Resource Identifier (URI): Generic Syntax, June 2005. URL `http://tools.ietf.org/html/rfc3986`. Accessed September, 2013.

IETF. The application/json Media Type for JavaScript Object Notation (JSON), July 2006. URL `http://tools.ietf.org/html/rfc4627`. Accessed September, 2013.

ISO. Information technology – Computer graphics and image processing – The Virtual Reality Modeling Language – Part 1: Functional specification and UTF-8 encoding, 1997. URL `http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=25508`. Accessed September, 2013.

ISO. ISO/IEC 15938-1:2002; Information technology – Multimedia content description interface – Part 1: Systems, 2002a. URL `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=34228`. Accessed September, 2013.

ISO. nformation technology – Multimedia content description interface – Part 8: Extraction and use of MPEG-7 descriptions, 2002b. URL `http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=37778`. Accessed September, 2013.

ISO. Information technology – Coding of audio-visual objects – Part 11: Scene description and application engine, 2005. URL http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=38560. Accessed September, 2013.

ISO. Information technology – Coding of audio-visual objects – Part 20: Lightweight Application Scene Representation (LASeR) and Simple Aggregation Format (SAF), 2009a. URL http://www.iso.org/iso/iso_catalogue/catalogue_ics/catalogue_detail_ics.htm?csnumber=52454. Accessed September, 2013.

ISO. ISO 15836:2009; Information and documentation – The Dublin Core metadata element set, 2009b. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=52142. Accessed September, 2013.

ISO. ISO/IEC 19775-2:2010 Information technology – Computer graphics and image processing – Extensible 3D (X3D) – Part 2: Scene access interface (SAI), 2010. URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=44680. Accessed September, 2013.

ITU-T. Nested Context Language (NCL) and Ginga-NCL, June 2011. URL http://www.itu.int/rec/T-REC-H.761. Accessed September, 2013.

J. Jansen and D. C. A. Bulterman. SMIL State: an architecture and implementation for adaptive time-based web applications. *Multimedia Tools and Applications*, 43(3):203–224, 2009.

J. Jansen, P. Cesar, and D. C. A. Bulterman. A model for editing operations on active temporal multimedia documents. In *Proceedings of the 10th ACM symposium on Document engineering - DocEng '10*, pages 87–96, 2010.

J. Jansen, P. Cesar, R. L. Guimaraes, and D. C. A. Bulterman. Just-in-time personalized video presentations. In *Proceedings of the 2012 ACM symposium on Document engineering*, pages 59–68, 2012.

K. Järvelin and P. Ingwersen. User-Oriented and cognitive models of information retrieval. pages 5521–5534, 2010.

D. Jayagopi, T. Kim, A. Pentland, and D. Gatica-Perez. Privacy-sensitive recognition of group conversational context with sociometers. *Multimedia Systems*, 18(1):3–14, 2012.

JEITA. JEITA CP-3451; Exhangeable image file format for digital still cameras: Exif version 2.2, 2002. URL http://www.exif.org/Exif2-2.PDF. Accessed September, 2013.

Y.-G. Jiang, S. Bhattacharya, S.-F. Chang, and M. Shah. High-level event recognition in unconstrained videos. *International Journal of Multimedia Information Retrieval [Online first]*, pages 1–29, 2012. URL 10.1007/s13735-012-0024-2.

M. Jourdan. A formal semantics of SMIL: a web standard to describe multimedia documents. *Computer Standards & Interfaces*, 23(5):439–455, 2001.

S. Junuzovic, R. Hegde, Z. Zhang, P. A. Chou, Z. Liu, and C. Zhang. Requirements and recommendations for an enhanced meeting viewing experience. In *Proc. of the 16th ACM international conference on Multimedia*, pages 539–548, 2008.

S. Junuzovic, K. Inkpen, R. Hegde, Z. Zhang, J. Tang, and C. Brooks. What did i miss?: in-meeting review using multimodal accelerated instant replay (air) conferencing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 513–522, 2011.

V. Kalnikaite, A. Sellen, S. Whittaker, and D. Kirk. Now let me see where i was: understanding how lifelogs mediate memory. In *CHI '10: Proceedings of the 28th international conference on Human factors in computing systems*, pages 2045–2054, 2010. ISBN 978-1-60558-929-9.

V. Kalnikaitė, P. Ehlen, and S. Whittaker. Markup as you talk: establishing effective memory cues while still contributing to a meeting. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 349–358, 2012.

R. Kaptein and M. Marx. Focused retrieval and result aggregation with political data. *Information retrieval*, 13(5):412–433, 2010.

D. Kirk, A. Sellen, R. Harper, and K. Wood. Understanding videowork. In *Proceedings of the 2007 SIGCHI Conference on Human Factors in Computing Systems*, pages 61–70, 2007.

S. Kopf and W. Effelsberg. Mobile cinema: canonical processes for video adaptation. *Multimedia Systems*, 14(6):369–375, 2008.

S. Kopf, S. Wilk, and W. Effelsberg. Bringing videos to social media. In *Multimedia and Expo (ICME), 2012 IEEE International Conference on*, pages 681–686, 2012.

D. Korchagin, S. Duffner, P. Motlicek, and C. Scheffler. Multimodal Cue Detection Engine for Orchestrated Entertainment. In *Proceedings of the 18th International Conference on Multimedia Modeling (MMM'12) - LNCS 7131*, pages 662–670, 2012.

Y. Koyama, Y. Sawamoto, Y. Hirano, S. Kajita, K. Mase, T. Suzuki, K. Katsuyama, and K. Yamauchi. A multi-modal dialogue analysis method for medical interviews based on design of interaction corpus. *Personal and Ubiquitous Computing*, 14(8):767–778, 2010.

F. Kuijk, R. L. Guimarães, P. Cesar, and D. C. Bulterman. Adding dynamic visual manipulations to declarative multimedia documents. In *Proceedings of the 9th ACM symposium on Document engineering - DocEng '09*, pages 149–152, 2009.

K. Kurihara. CinemaGazer : a System for Watching Videos at Very High Speed. pages 108–115, 2012.

R. Laiola Guimarães, P. Cesar, and D. C. Bulterman. Creating and sharing personalized time-based annotations of videos on the web. *Proceedings of the 10th ACM symposium on Document engineering - DocEng '10*, pages 27–36, 2010.

R. Laiola Guimarães, P. Cesar, and D. C. Bulterman. "let me comment on your video": supporting personalized end-user comments within third-party online videos. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web*, WebMedia '12, pages 253–260, 2012.

M. LaRosa, D. Poole, and R. Schusteritsch. Designing and deploying usetube, google's global user experience observation and recording system. In *Proceedings of the 27th international conference extended abstracts on Human factors in computing systems - CHI EA '09*, pages 2971–2986, 2009.

Y. Li, M. Wald, G. Wills, S. Khoja, D. Millard, J. Kajaba, P. Singh, and L. Gilbert. Synote: development of a Web-based tool for synchronized annotations. *New Review of Hypermedia and Multimedia*, 17(3):295–312, 2011.

C. Liang, C. Xu, J. Cheng, W. Min, and H. Lu. Script-to-Movie: A Computational Framework for Story Movie Composition. *IEEE Transactions on Multimedia*, 15(2):401–414, 2013.

I. Lindstedt, J. Löwgren, B. Reimer, and R. Topgaard. Nonlinear news production and consumption: A collaborative approach. *Comput. Entertain.*, 7(3):42–59, 2009.

F. Linhalis, C. R. Moraes, and M. G. C. Pimentel. Using Sensors as an Alternative to Start-up Lectures in Ubiquitous Environments. *IEEE Learning Technology*, 12:14–16, 2010a.

F. Linhalis, C. R. Moraes, S. H. P. Silva, and M. G. C. Pimentel. Where and with whom do you wanna meet?: Session-based collaborative work. In *Webmedia 2010 - Brazilian Symposium On Multimedia and the Web*, volume 1, pages 123–130, 2010b.

X. Liu, M. Corner, and P. Shenoy. SEVA: Sensor-enhanced video annotation. *ACM Trans. Multimedia Comput. Commun. Appl.*, 5(3):1–26, 2009.

S. Luz and B. Kane. Classification of patient case discussions through analysis of vocalisation graphs. In *Proceedings of the 2009 international conference on Multimodal interfaces*, pages 107–114, 2009.

C.-X. Ma, Y.-J. Liu, H.-A. Wang, D.-X. Teng, and G.-Z. Dai. Sketch-Based Annotation and Visualization in Video Authoring. *Multimedia, IEEE Transactions on*, 14(4):1153–1165, 2012.

E. Mannens, D. Deursen, R. Troncy, S. Pfeiffer, C. Parker, Y. Lafon, J. Jansen, M. Hausenblas, and R. Walle. A URI-based approach for addressing fragments of media resources on the web. *Multimedia Tools and Applications*, 59(2):691–715, 2012.

M. Markkula and E. Sormunen. Video needs at the different stages of television program making process. In *Proceedings of the 1st international conference on Information interaction in context - IIiX*, pages 111–118, 2006.

D. S. Martins and M. G. C. Pimentel. End-User ubiquitous multimedia production: Process and case studies. In *Proceedings of the 4th International Conference on Ubi-Media Computing (U-Media)*, pages 197–202, 2011.

D. S. Martins and M. G. C. Pimentel. Browsing interaction events in recordings of small group activities via multimedia operators. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web - WebMedia '12*, pages 245–252, 2012.

D. S. Martins, D. A. Vega-Oliveros, and M. da Graça Campos Pimentel. Automatic authoring of interactive multimedia documents via media-oriented operators. *ACM SIGAPP Applied Computing Review*, 11(4):26–37, 2011a.

P. Martins, T. Langlois, and T. Chambel. MovieClouds: content-based overviews and exploratory browsing of movies. In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 133–140, 2011b.

K. Mase, K. Niwa, and T. Marutani. Socially assisted multi-view video viewer. *Proceedings of the 13th international conference on multimodal interfaces - ICMI '11*, page 319, 2011.

J. Matejka, T. Grossman, and G. Fitzmaurice. Swifter: improved online video scrubbing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1159–1168, 2013.

J. Mazzola Paluska, H. Pham, and S. Ward. ChunkStream: Interactive streaming of structured data. *Pervasive and Mobile Computing*, 6(6):607–622, 2010.

B. Meixner, K. Matusik, C. Grill, and H. Kosch. Towards an easy to use authoring tool for interactive non-linear video. *Multimedia Tools and Applications (Online first)*, pages 1–26, 2012. URL DOI:10.1007/s11042-012-1218-6.

E. L. Melo, C. C. Viel, C. A. C. Teixeira, A. C. Rondon, D. d. P. Silva, D. G. Rodrigues, and E. C. Silva. WebNCL: a web-based presentation machine for multimedia documents. In *Proceedings of the 18th Brazilian symposium on Multimedia and the web - WebMedia '12*, pages 403–410, 2012.

R. Mertens, S. Pospiech, C. Wiesen, and M. Ketterl. The Tempo-Structural fisheye slider for navigation in web lectures. In *Multimedia (ISM), 2011 IEEE International Symposium on*, pages 553–558, 2011.

S. Minneman, S. Harrison, B. Janssen, G. Kurtenbach, T. Moran, I. Smith, and B. van Melle. A confederation of tools for capturing and accessing collaborative activity. In *Proceedings of the 3rd ACM International Conference on Multimedia*, pages 523–534, 1995.

T. J. K. P. Monserrat, S. Zhao, K. McGee, and A. V. Pandey. NoteVideo: facilitating navigation of blackboard-style lecture videos. In *Proceedings of the 2013 ACM annual conference on Human factors in computing systems*, pages 1139–1148, 2013.

C. Moock. *Essential ActionScript 3.0*. O'Reilly Adobe Dev Library, 2007.

R. B. Musburger and G. Kindem. *Introduction to media production*. Focal Press, 4th edition, 2009.

F. Nack. Capture and transfer of metadata during video production. In *Proceedings of the ACM workshop on Multimedia for Human Communication: from Capture to Convey*, pages 17–20, 2005.

A. Nagargadde, V. Sridhar, and K. Ramamritham. Representation and processing of information related to real world events. *Knowledge-Based Systems*, 20(1):1–16, 2007.

M. Nathan, M. Topkara, J. Lai, S. Pan, S. Wood, J. Boston, and L. Terveen. In case you missed it: benefits of attendee-shared annotations for non-attendees of remote meetings. In *Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work*, pages 339–348, 2012.

N. Pattanasri, S. Pradhan, and K. Tanaka. Extending information unit across media streams for improving retrieval effectiveness. *Data & Knowledge Engineering*, 83:70–92, 2013.

B. Pellan and C. Concolato. Authoring of scalable multimedia documents. *Multimedia Tools and Applications*, 43(3):225–252, 2009.

M. Perez-Luque and T. Little. A temporal reference framework for multimedia synchronization. *IEEE Journal on Selected Areas in Communications*, 14(1):36–51, 1996.

M. Perry, O. Juhlin, M. Esbjörnsson, and A. Engström. Lean collaboration through video gestures: co-ordinating the production of live televised sport. In *Proceedings of the 27th International Conference on Human Factors in Computing Systems*, pages 2279–2288, 2009.

S. Perugini. Supporting multiple paths to objects in information hierarchies: Faceted classification, faceted search, and symbolic links. *Information Processing & Management*, 46 (1):22–43, 2010.

M. d. G. C. Pimentel, R. G. Cattelan, E. L. Melo, A. F. Prado, and C. A. C. Teixeira. End-user live editing of iTV programmes. *International Journal of Advanced Media and Communication*, 4(1):78–102, 2010.

M. G. C. Pimentel, G. D. Abowd, and Y. Ishiguro. Linking by interacting: a paradigm for authoring hypertext. In *Proceedings of the eleventh ACM on Hypertext and hypermedia*, pages 39–48, 2000.

M. G. C. Pimentel, Y. Ishiguro, B. Kerimbaev, G. Abowd, and M. Guzdial. Supporting educational activities through dynamic web interfaces. *Interacting with Computers*, 13(3): 353–374, 2001.

M. G. C. Pimentel, C. Prazeres, H. Ribas, D. Lobato, and C. Teixeira. Documenting the pen-based interaction. In *Proceedings of the 11th Brazilian Symposium on Multimedia and the web*, WebMedia '05, pages 1–8, 2005.

M. G. C. Pimentel, R. Goularte, R. G. Cattelan, F. S. Santos, C. Teixeira, and U. Federal. Enhancing Multimodal Annotations with Pen-Based Information. In *Ninth IEEE International Symposium on Multimedia Workshops (ISMW 2007)*, pages 207–213, 2007.

N. Pissinou, I. Radev, K. Makki, and W. Campbell. Spatio-temporal composition of video objects: representation and querying in video database systems. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1033–1040, 2001.

S. Pongnumkul, M. Dontcheva, W. Li, J. Wang, L. Bourdev, S. Avidan, and M. F. Cohen. Pause-and-play: automatically linking screencast video tutorials with applications. In *Proceedings of the 24th ACM Symposium on User Interface Software and Technology*, pages 135–144, 2011.

A. Popescu-Belis, D. Lalanne, and H. Bourlard. Finding information in multimedia meeting records. *IEEE Multimedia*, 19(2):48–57, 2012.

R. Poppe. A survey on vision-based human action recognition. *Image and Vision Computing*, 28 (6):976–990, 2010.

S. L. Presti, D. Bert, and A. Duda. TAO: Temporal Algebraic Operators for modeling multimedia presentations. *Journal of Network and Computer Applications*, 25(4):319–342, 2002.

D. Rijsselbergen, B. Keer, and R. Walle. The canonical expression of the drama product manufacturing processes. *Multimedia Systems*, 14(6):395–403, 2008.

B. Rogge, J. Bekaert, and R. Van de Walle. Timing Issues in Multimedia Formats: Review of the Principles and Comparison of Existing Formats. *IEEE Transactions on Multimedia*, 6(6): 910–924, 2004.

M. Sadallah, O. Aubert, and Y. Prié. CHM: an annotation- and component-based hypervideo model for the web. *Multimedia Tools and Applications (Online First)*, pages 1–35, 2012. URL DOI:10.1007/s11042-012-1177-y.

P. N. M. Sampaio and J. P. Courtiat. A formal approach for the presentation of interactive multimedia documents. In *Proceedings of the eighth ACM international conference on Multimedia - MULTIMEDIA '00*, pages 435–438, 2000.

S. Santosa, F. Chevalier, R. Balakrishnan, and K. Singh. Direct space-time trajectory control for visual media editing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1149–1158, 2013.

A. Scherp, T. Franz, C. Saathoff, and S. Staab. A core ontology on events for representing occurrences in the real world. *Multimedia Tools and Applications*, (Online first):1–39, 2010.

A. Schmidt. Ubiquitous computing: Are we there yet? *IEEE Computer*, 43(2):95–97, 2010.

K. Schoeffmann and L. Boeszoermenyi. Video browsing using interactive navigation summaries. In *Content-Based Multimedia Indexing, 2009. Seventh International Workshop on*, pages 243–248, 2009. URL http://dx.doi.org/10.1109/cbmi.2009.40.

P. Shrestha, P. H. N. de With, H. Weda, M. Barbieri, and E. H. L. Aarts. Automatic mashup generation from multiple-camera concert recordings. In *Proceedings of the international conference on Multimedia*, MM '10, pages 541–550, 2010.

E. C. O. Silva, J. A. F. dos Santos, and D. C. Muchaluat-Saade. NCL4WEB: translating NCL applications to HTML5 web pages. In *Proceedings of the 2013 ACM symposium on Document engineering - DocEng '13*, pages 253–262, 2013.

V. K. Singh, J. Luo, D. Joshi, M. Das, P. Lei, and P. Stubler. Dynamic media show drivable by semantics. In *Proceedings of the 19th ACM international conference on Multimedia*, pages 815–816, 2011.

L. Snidaro, I. Visentini, and G. L. Foresti. Fusing multiple video sensors for surveillance. *ACM Transactions on Multimedia Computing, Communications, and Applications*, 8(1):1–18, 2012.

C. G. M. Snoek and M. Worring. Concept-Based video retrieval. *Found. Trends Inf. Retr.*, 2(4): 215–322, 2009.

C. S. Soares-Neto, L. F. G. Soares, and C. S. Souza. The Nested Context Language reuse features. *Journal of the Brazilian Computer Society*, 16(4):229–245, 2010.

J. Teevan, D. Liebling, A. Paradiso, C. Garcia Jurado Suarez, C. von Veh, and D. Gehring. Displaying mobile feedback during a presentation. *Proceedings of the 14th international*

conference on Human-computer interaction with mobile devices and services - MobileHCI '12, pages 379–382, 2012.

C. Teixeira, E. Melo, R. Cattelan, and M. G. C. Pimentel. Taking advantage of contextualized interactions while users watch TV. *Multimedia Tools and Applications*, 50(3):587–607, 2010a.

C. A. C. Teixeira, G. B. Freitas, and M. d. G. Pimentel. Distributed discrimination of media moments and media intervals: A watch-and-comment approach. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1929–1935, 2010b.

C. A. C. Teixeira, E. L. Melo, G. B. Freitas, C. A. S. Santos, and M. d. G. C. Pimentel. Discrimination of media moments and media intervals: sticker-based watch-and-comment annotation. *Multimedia Tools and Applications*, 61(3):675–696, 2011.

J. Terken and J. Sturm. Multimodal support for social dynamics in co-located meetings. *Personal Ubiquitous Computing*, 14:703–714, 2010.

C. A. B. Tiellet, A. G. Pereira, E. B. Reategui, J. V. Lima, and T. Chambel. Design and evaluation of a hypervideo environment to support veterinary surgery learning. In *HT '10: Proceedings of the 21st ACM conference on Hypertext and hypermedia*, pages 213–222, 2010.

B. T. Truong and S. Venkatesh. Video abstraction: A systematic review and classification. *ACM Trans. Multimedia Comput. Commun. Appl.*, 3(1), 2007.

Q. Tung, R. Swaminathan, A. Efrat, and K. Barnard. Expanding the point: automatic enlargement of presentation video elements. In *Proceedings of the 19th ACM International Conference on Multimedia*, pages 961–964, 2011.

G. Tur, A. Stolcke, L. Voss, S. Peters, D. Hakkani-Tur, J. Dowding, B. Favre, R. Fernández, M. Frampton, M. Frandsen, and Others. The CALO meeting assistant system. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(6):1601–1611, 2010.

W. R. van Hage, V. Malaisé, R. Segers, L. Hollink, and G. Schreiber. Design and use of the Simple Event Model (SEM). *Web Semantics: Science, Services and Agents on the World Wide Web*, 9(2):128–136, 2011.

R. van Zwol, B. Sigurbjornsson, R. Adapala, L. Garcia Pueyo, A. Katiyar, K. Kurapati, M. Muralidharan, S. Muthu, V. Murdock, P. Ng, and Others. Faceted exploration of image search results. In *Proceedings of the 19th international conference on World wide web*, pages 961–970, 2010.

D. A. Vega-Oliveros, D. S. Martins, and M. d. G. C. Pimentel. Interactors: operators to automatically generate interactive multimedia documents from captured media. In

Proceedings of the XVI Brazilian Symposium on Multimedia and the Web (Webmedia 2010), pages 163–170, 2010a.

D. A. Vega-Oliveros, D. S. Martins, and M. G. C. Pimentel. "This conversation will be recorded": automatically generating interactive documents from captured media. In *Proceedings of the 10th ACM symposium on Document engineering (DocEng '10)*, pages 37–40, 2010b.

D. A. Vega-Oliveros, D. S. Martins, and M. d. G. C. Pimentel. Media-oriented operators for authoring interactive multimedia documents generated from capture sessions. In *Proceedings of the 26th Symposium On Applied Computing (SAC 2011)*, pages 1267–1272, 2011a.

D. A. Vega-Oliveros, L. S. a. Oliveira, D. S. Martins, and M. d. G. C. Pimentel. Viewing by interactions: media-oriented operators for reviewing recorded sessions on TV. In *Proceddings of the 9th International Conference on Interactive Television (EuroITV '11)*, pages 213–222, 2011b.

C. C. Viel, E. L. Melo, M. D. G. C. Pimentel, and C. A. C. Teixeira. Go beyond boundaries of iTV applications. *Proceedings of the 2013 ACM symposium on Document engineering - DocEng '13*, pages 263–272, 2013.

R. Vliegendhart, M. Larson, and A. Hanjalic. LikeLines: collecting timecode-level feedback for web videos through user interactions. In *Proceedings of the 20th ACM international conference on Multimedia*, pages 1271–1272, 2012.

W3C. XHTML+SMIL Profile (W3C Note), January 2002. URL `http://www.w3.org/TR/XHTMLplusSMIL/`. Accessed September, 2013.

W3C. SMIL 2.0 Extension for Professional Multimedia Authoring - Preliminary Investigation (W3C Note), April 2003. URL `http://www.w3.org/TR/SMIL2-AuthExt/`. Accessed September, 2013.

W3C. RDF Semantics, February 2004a. URL `http://www.w3.org/TR/rdf-mt/`. Accessed September, 2013.

W3C. RDF/XML Syntax Specification (Revised), February 2004b. URL `http://www.w3.org/TR/rdf-syntax-grammar/`. Accessed September, 2013.

W3C. Document Object Model (DOM) Level 3 Core Specification, W3C Recommendation, April 2004c. URL `http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/`. Accessed September, 2013.

W3C. Synchronized Multimedia Integration Language (SMIL 3.0), Recommendation: SMIL 3.0 Media Object, 2008a. URL http://www.w3.org/TR/smil/smil-extended-media-object.html. Accessed September, 2013.

W3C. Synchronized Multimedia Integration Language (SMIL 3.0), Recommendation, 2008b. URL http://www.w3.org/TR/smil. Accessed September, 2013.

W3C. Synchronized Multimedia Integration Language (SMIL 3.0), Recommendation: SMIL Linking, 2008c. URL http://www.w3.org/TR/smil/smil-extended-linking.html. Accessed September, 2013.

W3C. Synchronized Multimedia Integration Language (SMIL 3.0), Recommendation: SMIL 3.0 Metainformation, 2008d. URL http://www.w3.org/TR/smil/smil-metadata.html. Accessed September, 2013.

W3C. SMIL 3.0 smilText, 2008e. URL http://www.w3.org/TR/smil/smil-text.html. Accessed September, 2013.

W3C. Synchronized Multimedia Integration Language (SMIL 3.0), Recommendation: SMIL 3.0 Time Manipulations, 2008f. URL http://www.w3.org/TR/smil/smil-timemanip.html. Accessed September, 2013.

W3C. Synchronized Multimedia Integration Language (SMIL 3.0), Recommendation: SMIL 3.0 Timing and Synchronization, 2008g. URL http://www.w3.org/TR/SMIL3/smil-timing.html. Accessed September, 2013.

W3C. Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, November 2008. URL http://www.w3.org/TR/REC-xml/. Accessed September, 2013.

W3C. Cascading Style Sheets (CSS) Snapshot 2010, Working group note, May 2011a. URL http://www.w3.org/TR/css-2010/. Accessed September, 2013.

W3C. CSS Selectors Level 3, September 2011b. URL http://www.w3.org/TR/css3-selectors/. Accessed September, 2013.

W3C. Turtle - Terse RDF Triple Language, W3C Team Submission, March 2011c. URL http://www.w3.org/TeamSubmission/turtle/. Accessed September, 2013.

W3C. Scalable Vector Graphics (SVG) 1.1 (Second Edition), W3C Recommendation, August 2011d. URL http://www.w3.org/TR/SVG11/. Accessed September, 2013.

W3C. SMIL Timesheets 1.0, W3C Working Group Note, March 2012a. URL http://www.w3.org/TR/timesheets/. Accessed September, 2013.

W3C. Media Fragments URI 1.0 (basic), 2012b. URL http://www.w3.org/TR/media-frags/. Accessed September, 2013.

W3C. Ontology for Media Resources 1.0 (recommendation), 2012c. URL http://www.w3.org/TR/2012/REC-mediaont-10-20120209/. Accessed September, 2013.

W3C. HTML5: A vocabulary and associated APIs for HTML and XHTML (Candidate Recommendation), November 2012d. URL http://dev.w3.org/html5/spec/. Accessed September, 2013.

F. Wang, C.-W. W. Ngo, and T.-C. C. Pong. Exploiting self-adaptive posture-based focus estimation for lecture video editing. In *Proceedings of the 13th annual ACM international conference on Multimedia - MULTIMEDIA '05*, MULTIMEDIA '05, page 327. ACM Press, 2005. URL http://portal.acm.org/citation.cfm?doid=1101149.1101217http://dx.doi.org/10.1145/1101149.1101217.

M. Wang, B. Ni, X. S. Hua, and T. S. Chua. Assistive tagging: A survey of multimedia tagging with human-computer joint exploration. *ACM Comput. Surv.*, 44(4), 2012.

N. Weibel, A. Fouse, E. Hutchins, and J. D. Hollan. Supporting an integrated paper-digital workflow for observational research. In *Proceedings of the 16th International Conference on Intelligent User Interfaces*, pages 257–266, 2011.

M. Weiser. Some computer science issues in ubiquitous computing. *ACM SIGMOBILE Mobile Computing and Communications Review*, 3(3), 1999.

R. Weiss, A. Duda, and D. Gifford. Composition and search with a video algebra. *Multimedia, IEEE*, 2(1):12–25, 1995. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=368596.

U. Westermann and R. Jain. Toward a common event model for multimedia applications. *IEEE Multimedia*, 14(1):19–29, 2007.

S. Whittaker. Personal information management: From information consumption to curation. *Annual Review of Information Science and Technology*, 45(1):1–62, 2011.

S. Whittaker, S. Tucker, K. Swampillai, and R. Laban. Design and evaluation of systems to support interaction capture and retrieval. *Personal and Ubiquitous Computing*, 12(3): 197–221, 2007.

M. A. Winget. Annotations on musical scores by performing musicians: Collaborative models, interactive methods, and music digital library tool development. *J. Am. Soc. Inf. Sci.*, 59(12): 1878–1897, 2008.

C. I. Wu, C. M. J. Teng, Y. C. Chen, T. Y. Lin, H. H. Chu, and J. Y. J. Hsu. Point-of-capture archiving and editing of personal experiences from a mobile device. *Personal Ubiquitous Computing*, 11(4):235–249, 2007.

L. Xie, H. Sundaram, and M. Campbell. Event mining in multimedia streams. *Proceedings of the IEEE*, 96(4):623–647, 2008.

Z. Yu and Y. Nakamura. Smart meeting systems: A survey of state-of-the-art and open issues. *ACM Computing Surveys*, 42(2):1–20, 2010.

Z. Yu, N. Diakopoulos, and M. Naaman. The multiplayer: multi-perspective social video navigation. In *Adjunct proceedings of the 23nd annual ACM symposium on User interface software and technology*, pages 413–414, 2010.

Z. Yu, Z. Yu, X. Zhou, C. Becker, and Y. Nakamura. Tree-Based mining for discovering patterns of human interaction in meetings. *IEEE Transactions on Knowledge and Data Engineering*, 24(4):759–768, 2012.

J. Zhao, S. M. Drucker, D. Fisher, and D. Brinkman. TimeSlice: interactive faceted browsing of timeline data. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 433–436, 2012.

V. Zsombori, M. Frantzis, R. L. Guimaraes, M. F. Ursu, P. Cesar, I. Kegel, R. Craigie, and D. C. A. Bulterman. Automatic generation of video narratives from shared UGC. In *Proceedings of the 22nd ACM Conference on Hypertext and Hypermedia*, pages 325–334, 2011.