

UNIVERSIDADE FEDERAL FLUMINENSE

JOEL ANDRÉ FERREIRA DOS SANTOS

**MULTIMEDIA DOCUMENT VALIDATION
ALONG ITS LIFE CYCLE**

NITERÓI

2016

UNIVERSIDADE FEDERAL FLUMINENSE

JOEL ANDRÉ FERREIRA DOS SANTOS

**MULTIMEDIA DOCUMENT VALIDATION
ALONG ITS LIFE CYCLE**

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Sistemas de Computação.

Orientador:
Profa. Débora Christina Muchaluat Saade, D.Sc.

Co-orientador:
Prof. Christiano de Oliveira Braga, D.Sc.

NITERÓI

2016

Ficha Catalográfica elaborada pela Biblioteca da Escola de Engenharia e Instituto de Computação da UFF

S237 Santos, Joel André Ferreira dos

Multimedia document validation along its life cycle / Joel André

Ferreira dos Santos. – Niterói, RJ : [s.n.], 2016.

154 f.

Tese (Doutorado em Computação) - Universidade Federal Fluminense, 2016.

Orientadores: Débora Christina Muchaluat Saade, Christiano de Oliveira Braga.

1. Multimídia interativa. 2. NCL (Linguagem de marcação de documento). I. Título.

CDD 006.6

Multimedia document validation along its life cycle

Joel André Ferreira dos Santos

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Computação da Universidade Federal Fluminense como requisito parcial para a obtenção do Grau de Doutor em Computação. Área de concentração: Sistemas de Computação.

Aprovada em Fevereiro de 2016.

BANCA EXAMINADORA

Profa. D.Sc. Débora Christina Muchaluat Saade - Orientadora, UFF

Prof. D.Sc. Christiano de Oliveira Braga - Co-orientador, UFF

Prof. Ph.D. Célio Vinicius Neves de Albuquerque, UFF

Prof. D.Sc. Esteban Walter Gonzalez Clua, UFF

Prof. D.Sc. Edward Hermann Haeusler, PUC-Rio

Profa. D.Sc. Cláudia Maria Lima Werner, UFRJ

Niterói

2016

Everything should be made as simple as possible, but not simpler.
(Einstein)

à Alina

Agradecimentos

Agradecer corretamente todos que contribuíram com esse trabalho não é uma tarefa simples. Afinal, este é um trabalho desenvolvido ao longo de seis anos.

Primeiramente agradeço aos meus orientadores Débora Saade e Christiano Braga. Graças a sua paciente orientação, este trabalho possui a qualidade que possui. Além disso, gostaria de agradecer à Cécile Roisin e Nabil Layaïda pelas suas grandes contribuições neste trabalho.

Àqueles que conviveram comigo nesses últimos quatro anos, trocando ideias e dando conselhos. Um agradecimento especial aos amigos Diego Passos, Douglas Mattos, Glauco Amorim e Júlia Varanda do laboratório MídiaCom na UFF e à Guilherme Lima e Roberto Azevedo do laboratório Telemídia da PUC-Rio.

Viver um ano longe da família e amigos seria uma tarefa muito mais difícil, não fosse pelos amigos Damian Graux, Louis Jachiet, Thibaud Michel, Guillaume Dupraz-Canard e Mathieu Razafimahazo da equipe Tyrex do laboratório Inria. Obrigado pelo companheirismo e, claro, pelas partidas de *coinche*.

À minha família e meus amigos do Graham Bell e da Universidade Federal Fluminense por sempre estarem presentes.

À minha esposa Alina, que passou por tudo isso ao meu lado.

À você, por estar lendo esta tese.

Resumo

Um documento multimídia declara uma coleção de itens de mídia e relações entre eles no tempo e espaço. Desde sua *autoria/geração* até sua *execução*, um documento passa por diferentes etapas. Tais etapas representam o ciclo de vida de um documento multimídia. A especificação contida em um documento pode mudar de etapa a etapa, pois um documento é especificado (ou buscado), instanciado, adaptado e pode ter seu conteúdo e estrutura editada dinamicamente. A consistência de um documento multimídia ao longo do seu ciclo de vida significa que o documento sempre seguirá as diretrizes expressas em tempo de *autoria/geração*. É importante, portanto, garantir que um dado documento continue consistente. Este trabalho trata deste problema por meio da validação espaço-temporal de um documento em diferentes etapas de seu ciclo de vida.

O esquema de validação proposto se baseia em um modelo geral chamado Simple Hypermedia Model (\mathcal{SHM}), que captura a semântica de documentos multimídia. A partir de \mathcal{SHM} duas representações de um documento são possíveis. A primeira captura o leiaute de um documento em termos de seu estado ao longo de sua execução por meio de uma teoria de reescrita. Nós a chamamos *RWT* (*rewrite theory*). Validação em *RWT* é feita através de *model-checking*. A segunda representação captura o leiaute de um documento em termos de intervalos e ocorrências de eventos por meio de fórmulas em Satisfatibilidade Módulo Teorias (SMT). Nós a chamamos *SMT*. Validação em *SMT* é feita através de um solver SMT.

Devido às diferentes características de *RWT* e *SMT*, cada técnica de validação complementa a outra em termos da expressividade de \mathcal{SHM} e do tipo de propriedades a serem investigadas.

Palavras-chave: Validação de documentos multimídia, Ciclo de vida de documentos multimídia, Apresentação multimídia interativa, Autoria multimídia, Teoria de reescrita, Satisfatibilidade módulo teorias, SHM, Maude, NCL, SMIL.

Abstract

A multimedia document declares a collection of media items and relations among them in time and space. From document *authoring/generation* to *execution*, a document passes through different phases. Such phases represent the life cycle of a multimedia document. The specification contained in a document may change from phase to phase, as a document is specified (or queried), instantiated, adapted and may have its content or structure dynamically edited. By consistency of a multimedia document along its life cycle we mean that it always follows the guidelines expressed at *authoring/generation* time. It is important, therefore, to guarantee that the document remains consistent. In this work, we address this issue by performing spatio-temporal consistency validation at different steps.

The proposed validation scheme relies on a general model called Simple Hypermedia Model (\mathcal{SHM}), which can capture the semantics of multimedia documents. From \mathcal{SHM} two representations of a document are possible. The first captures the layout of a document in terms of its state throughout its execution by means of a rewrite theory. We call it *RWT* (*rewrite theory*). Validation in *RWT* is performed through model-checking. The second representation captures the layout of a document in terms of intervals and event occurrences by means of Satisfiability Modulo Theories (SMT) formulas. We call it *SMT*. Validation in *SMT* is performed through an SMT solver.

Due to different characteristics of *RWT* and *SMT*, each validation technique complements the other in terms of expressiveness of \mathcal{SHM} and the kind of properties to be investigated.

Keywords: Multimedia document validation, Multimedia document life cycle, Interactive multimedia presentation, Multimedia authoring, Rewriting theory, Satisfiability modulo theories, SHM, Maude, NCL, SMIL.

List of Figures

2.1	Possible presentations due to viewer interaction	8
2.2	Multimedia document life cycle	9
2.3	Case 1 spatio-temporal layout	19
2.4	Case 2 spatio-temporal layout	20
4.1	State changes	37
4.2	Fragment projection	38
4.3	Paused fragment projection	39
4.4	Allen's relations between time intervals	40
4.5	Presentation example	40
4.6	Possible temporal layouts	41
4.7	Randell, Cui and Cohn (RCC) spatial relations between active regions . . .	42
4.8	Possible cases of <i>A pover B</i>	42
4.9	Angle and distance between regions	42
4.10	(Intra)Cardinal directions	43
4.11	Value in relation to time	44
4.12	Validation approach architecture	45
4.13	State changes	50
4.14	State changes	64
4.15	Paused element projection	64
5.1	aNaa API architecture	71
5.2	<i>MaudeManager</i> package	73
5.3	<i>ValidationManager</i> package	74

5.4	Property editor interface	75
5.5	aNaa4Web interface	76
5.6	Document information in aNaa4Web	76
5.7	Media item duration	77
5.8	Error and warning messages	77
5.9	Satisfiability Modulo Theories (SMT) Validator architecture	80
5.10	Relation $A \ sep-x(d) B$	83
5.11	Relation $align-y(end, A, B, C)$	83
5.12	Relation $dist-x(middle, d, A, B, C)$	83
5.13	Relation $flow(A, B, C)$	84
5.14	Use case document presentation	85
5.15	Validation approach architecture	93
A.1	Nested Context Language (NCL) document example	109
A.2	Synchronized Multimedia Integration Language (SMIL) document example	115
B.1	Confluence property	123
B.2	Coherence property	124
B.3	Transition systems and rewriting logic	127
D.1	State changes	134
D.2	Extended state machine	134
D.3	Spatio-temporal layout for the first test	140
D.4	Test results	141
F.1	First João spatio-temporal layout	146

List of Listings

4.1	<i>Rel</i> example	51
4.2	Document example	52
5.1	NCL element example	72
5.2	NCL element representation in \mathcal{E}_{NCL}	72
5.3	Document representation using SMT Validator	81
A.1	NCL document example	109
A.2	NCL document example in <i>RWT</i>	111
A.3	<i>Syncbase</i> overwrite	113
A.4	SMIL document example	115
A.5	SMIL document example in <i>RWT</i>	117
B.1	Set-rewriting example	125
B.2	Meta representation of set-rewriting example	126
B.3	Set-rewriting example	127
B.4	Example of term search	127
B.5	LTL Property definition	128
B.6	<i>modelCheck</i> result	128
F.1	<i>First João NCL code</i>	146
F.2	<i>First João RWT code</i>	148
F.3	<i>First João module metarepresentation</i>	150
F.4	<i>First João SMT code</i>	152

List of Tables

2.1	Validation requirements for life cycle steps	15
3.1	Related work comparison	33
5.1	Document sizes and validation time comparison	79
D.1	Document sizes and validation time comparison	140
D.2	Document sizes and validation time comparison	140

Acronyms

aNa API for NCL Authoring

aNaa API for NCL Authoring and Analysis

AST Abstract Syntax Tree

API Application Programming Interface

caT context aware Trellis

DOM Document Object Model

DTV Digital Television

HMBS Hypermedia Model Based on Statecharts

HTML HyperText Markup Language

IPTV IP Television

ITU International Telecommunications Union

LTL Linear Temporal Logic

MPEG Moving Picture Experts Group

NCL Nested Context Language

NCM Nested Context Model

OWL-DL Web Ontology Language - Description Logic

RCC Randell, Cui and Cohn

RT-LOTOS Real-Time LOTOS

SHM Simple Hypermedia Model

SMT Satisfiability Modulo Theories

SMIL Synchronized Multimedia Integration Language

SUS System Usability Scale

TDD Test-driven Development

XML eXtensible Markup Language

XSLT XML Stylesheet Language Transformations

Contents

1	Introduction	1
1.1	Motivation	2
1.2	Goals	4
1.3	Contributions	6
1.4	Thesis Structure	7
2	Multimedia Document Life Cycle and Validation	8
2.1	Multimedia Document Life Cycle	9
2.1.1	Use Cases	11
2.1.2	Validation Requirements	13
2.2	Multimedia Document Validation	15
2.2.1	Structural Properties	15
2.2.2	Behavioral Properties	17
2.2.3	Structural and Behavioral Validation	18
3	Related Work	21
3.1	Structural Validation Works	21
3.1.1	NCL-Inspector	21
3.1.2	NCL-validator	22
3.1.3	VAMP	23
3.1.4	Schema Validation	24
3.2	Behavioral Validation Works	25

3.2.1	Reachability of States	25
3.2.1.1	Santos et al Approach	25
3.2.1.2	caT	26
3.2.1.3	HMBS	27
3.2.1.4	Felix Approach	28
3.2.1.5	Gaggi and Bossi Approach	28
3.2.1.6	SMIL Builder	29
3.2.1.7	Junior et al Approach	29
3.2.1.8	King et al Approach	30
3.2.2	Constraint Checking	30
3.2.2.1	Bertino et al Approach	30
3.2.2.2	Laborie et al Approach	31
3.2.2.3	Elias et al Approach	31
3.2.2.4	Belouaer and Maris Approach	32
3.3	Related Work Comparison	32
4	Proposed Validation Approach	35
4.1	Simple Hypermedia Model	35
4.1.1	Temporal Axis	37
4.1.2	Spatial Axis	41
4.1.3	Variable Handling	43
4.2	Validation Approach Architecture	44
4.3	Validation Based on Document State	46
4.3.1	Rewrite Theory \mathcal{R}_{RWT}	47
4.3.2	Modeling Spatio-temporal Relations	52
4.3.3	RWT Document Validation	54
4.4	Validation Based on Constraints	58

4.4.1	Satisfaction Problem d_{SMT}	59
4.4.2	<i>SMT</i> Document Validation	69
5	Implementation and Use	70
5.1	<i>RWT</i> Implementation	70
5.1.1	aNaa	70
5.1.2	Property editor	75
5.1.3	aNaa4Web	76
5.2	<i>RWT</i> Implementation in Practice	77
5.3	<i>SMT</i> Implementation	80
5.4	<i>SMT</i> Implementation in Practice	82
5.5	Use Case	84
5.5.1	Multimedia Document Example	84
5.5.2	Representation in Simple Hypermedia Model (\mathcal{SHM})	85
5.5.3	Example Validation	90
5.6	Closing Remarks	91
5.6.1	Validation Technique Choice	92
5.6.2	Tool Limitations	93
6	Conclusion	95
6.1	Contributions	96
6.2	Future Works	97
References		99
Appendix A - Multimedia languages mapping		106
A.1	NCL	106
A.2	Mapping NCL to <i>RWT</i>	110

A.3 SMIL	112
A.4 Mapping SMIL to <i>RWT</i>	115
Appendix B – Rewriting logic	118
B.1 Rewriting logic calculus	118
B.2 Rewrite metatheories	120
B.3 Maude	121
B.3.1 Syntax	121
B.3.2 Executability constraints	123
B.4 Set-rewriting theories and Reflection in Maude	124
B.5 Representing finite transition systems in rewriting logic	126
B.6 Searching and model checking	127
Appendix C – Satisfiability Modulo Theories	130
C.1 SMT Overview	130
C.2 Yices2	132
Appendix D – <i>RWT</i> Refinements	133
D.1 Previous <i>RWT</i> version	133
D.2 Theory refinements	138
D.3 Practical results	139
Appendix E – Module RRWT	143
Appendix F – Document example	146

Chapter 1

Introduction

Multimedia documents are present in our daily life as web pages, digital TV programs, smartphone applications, advertising screens and so on. A multimedia document¹ describes media items and relations among them either in time, space or both [Hardman 1998]. Such a description is usually textual, by means of some *integration* or *authoring language*. In the literature, both terms *integration language* and *authoring language* are used as synonyms. In this work, we use the former.

When executed, the specification contained in a document yields a particular arrangement of media items in time and space. This arrangement is called a multimedia document presentation or just multimedia presentation². Different from a document, a presentation is the result that is actually presented to the viewer. Usually, both terms *document* and *presentation* are used as synonyms. In this text, whenever such a distinction is not necessary, we may use the term *document* for meaning either a *document* or a *presentation*.

Multimedia environments have evolved in the past years due to the arrival of new types of displays, besides the increase in computer power and storage capacity. Among such evolutions we highlight:

- personalization of a document presentation for a given viewer [Laborie et al. 2011];
- splitting a presentation in an array of displays (for example a TV set and a second screen) [Sarkis et al. 2014, ITU 2009];
- new displays (such as touch screens or with embedded motion sensors) improving the

¹Hereinafter, we will use just the word “document” meaning a multimedia document.

²Hereinafter, we will use just the word “presentation” meaning a multimedia document presentation.

interface of a presentation with the viewer and enabling new kinds of inputs [Guedes et al. 2015];

- mixed reality browsers enabling content synchronization according to the viewer geographical position [Lemordant et al. 2013];
- dynamic editing of a presentation (for example by means of annotation [Teixeira et al. 2012] or live editing [Soares et al. 2012]);
- dynamic generation of documents according to user queries (such as dynamic web documents).

In order to cope with those changes, we update in this work the definitions presented in [Hardman 1998]. A document no longer describes just one presentation³, but a set of possible presentations. This set is restricted according to the viewer context so that the most suited presentation to the viewer is chosen. The viewer context contains information about its exhibition device, viewing preferences, and anything else that may influence the presentation. Although a document may yield different presentations, they are not completely different among each other, as they must follow the document specification.

1.1 Motivation

The specification contained in a document may vary along its life cycle, i.e., from its creation (authoring or generation) to its execution. It ranges from synchronization relations expressed in a high-level abstraction integration language to low-level executing events and incremental modifications may be performed over it. It is important, however, to guarantee that a document remains consistent to its specification, i.e., to the guidelines expressed at its creation, along its life cycle.

We believe that the consistency of a document can be maintained along its life cycle by means of validation. The idea is to combine the document specification with properties representing its main guidelines and validate it in order to guarantee that any change performed at a given life cycle step does not make a document inconsistent.

Several works published in the literature address the validation of multimedia documents. Most of them perform document validation at authoring, either by means of the automatic correction of the document (when authoring is performed through constraints)

³Let us assume all presentations of a given document differing just by viewer interactions as one for a moment.

or by providing to the author messages to help correcting the document. An example of the former is presented in [Elias et al. 2006] and an example of the latter is presented in [Gaggi and Bossi 2011].

It is also possible to find approaches for validating the document and adapting it to the viewer context. Two examples are seen in [Bertino et al. 2005] and [Laborie et al. 2011]. [Bertino et al. 2005] focuses on the document validation but also provides adaptation to the viewer context. [Laborie et al. 2011], on the other hand, focuses on the document adaptation. The proposed adaptation approach, however, takes into account author's guidelines and tries to maintain a "minimum distance" between the behavior of the adapted document in relation to the original one.

The validation provided by such works has a preventive role. Although they provide validation not only at authoring, they investigate if the document is executable and if no inconsistency may arise during execution. Such approaches, however, do not take into consideration the case where a document is dynamically edited. Besides, given that great part of the works need some kind of human interaction to correct the document, the case where a document is generated is not covered.

In this work we define a sequence of steps for representing the life cycle of multimedia documents. It relies on ideas presented by works related to document authoring and adaptation, generalizing them in order to identify steps where validation is necessary. The life cycle proposed here comprises three main steps, which are called: *authoring/generation*, *instantiation* and *execution*.

At the *authoring/generation* step, a document is created either by an author or by an automatic generation process. A document created at this step should be able to be instantiated into one or several presentations, thus it is important to guarantee that no misuse of constructs of the integration language(s) used occurs and no inconsistency arises from the set of relations among media items composing the document.

At the *instantiation* step, a document is instantiated into a possible presentation according to the viewer context as given through the querying process. Thus it is important to guarantee that the instantiation process chooses the most suited presentation to the viewer, considering the set of possible presentations described by that document.

At the *execution* step, the presentation unfolds and reacts to viewer interactions. During execution, it is possible that a presentation improves its content and behavior according to the occurrence of executing events, such as, for example, viewer interaction

or live editing. At this point, it is important to guarantee that the presentation stays consistent according to properties defined by the author, which we refer as author guidelines. For example, suppose we have a “two videos in sequence” presentation and an author guideline stating that videos should not be presented in parallel. If we add a third video, the presentation should evolve into a “three videos in sequence” presentation and not something else.

As seen, different steps in the document life cycle demand different guarantees. Each guarantee acts locally, at one step, in order to keep the document globally *consistent*, i.e., throughout its life cycle. The research presented in this work fits in the main idea of maintaining the consistency of a document along its life cycle. It proposes an approach for validating documents that is intended to be used in different steps of a document life cycle.

1.2 Goals

The main goal of this work is to provide the **validation** of a document along its life cycle. By validation we mean the process of verifying if a set of properties over a multimedia document can be satisfied together.

The types of multimedia document taken into consideration for the validation approach proposed in this work are the ones specified using a declarative authoring language. Applications such as interactive Digital TV programs are good use cases for the application of the validation techniques presented in this work. On the other hand, applications such as games, where the application logic is better specified using a general-purpose programming language, are not good use cases. It is worth highlighting that a web document can also be considered as a good use case in the case where its synchronization is described in a declarative fashion, either by embedding Synchronized Multimedia Integration Language (SMIL) [W3C 2008b] or Nested Context Language (NCL) [ITU 2009] tags in it or even using Web Animations. The case where the synchronization is specified using Javascript code is out of the scope of this work.

Properties to be validated over a document may express (but are not limited to):

- (i) syntax rules of an integration language;
- (ii) constraints associated to a given content;
- (iii) temporal and spatial relationships among content;

- (iv) author’s design choices; and
- (v) the viewer’s context.

Relying on the life cycle defined in this work we identify where, i.e., in which step, validation is necessary and its requirements. Based on such information, we design a validation approach to be used at different moments in a document life cycle.

This work proposes a general model called Simple Hypermedia Model (\mathcal{SHM}) for representing a document layout in either time and space for the purpose of validation. In the temporal dimension, media items are represented by media fragments and, in the spatial dimension, by rectangular regions. Relations are expressed in \mathcal{SHM} by the set of operators defining either causal relations or constraints.

The \mathcal{SHM} model evolved from previous work [dos Santos 2012, dos Santos et al. 2013a], where we proposed a validation approach based on a rewrite theory [Meseguer 2012] we call here RWT . It captures the layout of a presentation in terms of its state throughout its execution. A document in RWT is described by a rewrite theory \mathcal{R}_{RWT} , which induces a transition system \mathcal{S}_{RWT} , where each state represents the state of the presentation as a whole in a given moment of its execution, and each transition models the viewer interaction or a time lapse. Validation is performed by model-checking over \mathcal{S}_{RWT} .

In this work we have refined theory RWT in order to improve its efficiency [dos Santos et al. 2012a, dos Santos et al. 2013b, dos Santos et al. 2015a] and enable the spatio-temporal validation of multimedia documents [dos Santos et al. 2015b]. The refined theory RWT will be presented in Chapter 4.

Moreover, we also refine the \mathcal{SHM} model in order to enable the definition of constraints. Moreover, we extend \mathcal{SHM} enabling a second validation technique called SMT [dos Santos et al. 2015b]. It captures the layout of a document in terms of intervals and event occurrences. A document in SMT is described by Satisfiability Modulo Theories (SMT) [Barrett et al. 2009] formulas and validation is performed through an SMT solver.

RWT is more suited for expressing documents where events and different occurrences of media fragments are important features. On the other hand, SMT is more suited for expressing documents where numeric dependencies among media fragment intervals or regions are important features. Besides its power of expressiveness, the choice for a given validation approach may depend on the type of validation to be performed. Thus, one

approach or the other is chosen according to the presentation life cycle step.

1.3 Contributions

This thesis main contributions are:

- The refinement of *RWT* theory for improving its efficiency;
- The definition of a multimedia document life cycle, which brings the following contributions:
 - The definition of a general sequence of steps for multimedia document handling;
 - The definition of validation requirements for different steps of a document life cycle;
 - The discussion of which technique is more suited for different steps of a document life cycle.
- The definition of a validation approach based on two different validation techniques, which brings the following contributions:
 - The definition of a formal representation of the document presentation state throughout its execution;
 - The definition of a validation approach based on the document state;
 - The definition of a formal representation of the document layout based on constraints;
 - The definition of a validation approach based on constraints;
 - The design of a validation approach to be used at different steps of a document life cycle.
- The implementation of tools for performing document validation using each proposed validation technique.

1.4 Thesis Structure

The remainder of this text is structured as follows.

Chapter 2 defines the life cycle of a multimedia document and presents how some works published in the literature fit in the proposed life cycle and the validation requirements for different life cycle steps. This chapter also discusses different types of validation for a given document, which are the validation of the document structure and the validation of the document behavior. The latter may consider just the temporal relations declared in a document, the spatial ones or the combination of both.

Chapter 3 presents related work published in the literature considering the validation of multimedia documents. Related works are divided according to the type of validation they provide and the approach used for validation.

Chapter 4 introduces the abstract definition of the \mathcal{SHM} model for both temporal and spatial axes. It also presents the general architecture of the validation approach proposed in the work with its two validation techniques. For the validation technique based on the document state, it presents the rewriting logic theory used to represent a multimedia document together with its validation through model-checking. For the validation technique based on constraints, it presents the representation of a multimedia document as SMT formulas together with its validation through SMT solving.

Chapter 5 presents implementation details of the validation approach proposed in this work. It presents the tools that implements each validation technique and their practical results. The chapter also presents a document example and its representation in the \mathcal{SHM} model together with a discussion about the use of each validation technique.

Chapter 6 concludes this thesis, restating its contributions and presenting future works.

Chapter 2

Multimedia Document Life Cycle and Validation

A multimedia document specifies a collection of media items - also known as its content - and a set of relations in time, space or both - also known as its layout [Hardman 1998]. Media items represent human-consumable information units, such as audio, video, image, etc. Relations in a document are defined among media items and may take into account event occurrences, such as viewer interactions (including motion recognition), the result of a computation (e.g., by an auxiliary script) or a query (e.g., to an external server) performed at runtime or even arrival of the user at some geographical position.

A document's content and relations describe a set of possible presentations. This set of presentations is restricted according to the viewer context, in order to choose a subset of presentations more suited to the viewer. For example, adjusting the presentation size to the viewer exhibition device size, suppressing or not an audio presentation due to the availability of an audio channel, etc.

In addition, this subset of presentations are further restricted according to viewer interactions during the presentation. For example, suppose a document where a given media item m_2 is presented, given that a viewer interacts with media item m_1 . Figure 2.1 presents two possible presentations obtained from that document differing if the viewer interaction happens or not.

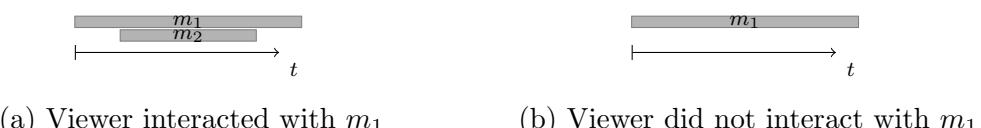


Figure 2.1: Possible presentations due to viewer interaction

The specification contained in a document is described with an integration language. In the declarative programming paradigm, focus of this work, integration languages provide constructs with a high degree of abstraction for declaring the document content and its spatio-temporal layout. The idea of a declarative integration language is to separate the description of a document from the specificities of its execution [Hardman 1998]. The level of abstraction varies according to the paradigm followed by the integration language. A discussion about paradigms can be seen in [Blakowski and Steinmetz 1996, Boll 2001, dos Santos 2012].

Another benefit of declarative programming, in the multimedia scenario, is to ease the task of creating multimedia documents. Such goal is important since multimedia documents can be used in different areas, such as web, digital TV and IP Television (IPTV); and by different author profiles, such as developers and content producers.

2.1 Multimedia Document Life Cycle

The specification contained in a document may vary from its creation to its execution. It ranges from synchronization relations expressed in a high-level abstraction integration language to low-level executing events. Different works related to documents commonly assume a document life cycle composed of *conception*, *storage* and *execution* steps (authoring-related works [Muchaluat-Saade 2003, Bulterman et al. 2013]) or of *conception*, *adaptation* and *execution* steps (adaptation-related works [Lemlouma and Layaïda 2004, Na and Furuta 2001]). Relying on such ideas we define a document life cycle that is a generalization of such approaches. The life cycle presented here was also inspired by [Bulterman et al. 2013] and was designed so that approaches like [Sarkis et al. 2014, Lemlouma and Layaïda 2004, Laborie et al. 2011, Teixeira et al. 2012, Soares et al. 2012] could fit into it. Figure 2.2 depicts the proposed life cycle of a document.

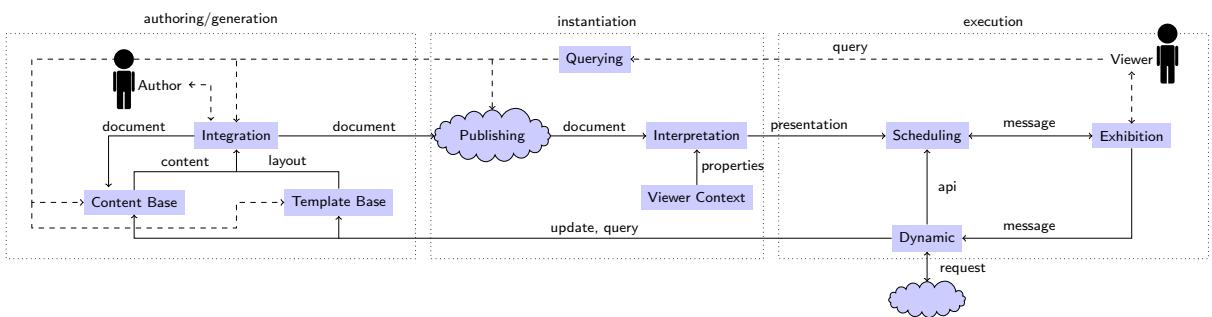


Figure 2.2: Multimedia document life cycle

The life cycle presented here comprises three main steps, which are called: *author-*

ing/generation, instantiation and *execution*, represented by dotted rectangles, which are subdivided into smaller steps, represented by filled rectangles.

Regarding the *authoring/generation* step, it is important to highlight three aspects. (i) Documents are also considered as content. (ii) A template is considered as a general layout that is instantiated at *Integration* step for a given content. Thus it may represent from constructions in a given integration language to “documents with holes”. (iii) A document may be created either by an author or by an automatic generation process. In the former it is possible for the author of a document to use templates in order to improve its authoring as discussed in [Damasceno et al. 2014]. In the latter, documents are created according to the viewer query by combining existent content into predefined templates. It is worth noticing that the author has an important role in the creation of such templates.

According to a query/request from the *Viewer*, the *Integration* step will either (i) organize the content available in the *Content Base* according to the layout described in the *Template Base* or (ii) if the user query/request yields a document already stored in the *Content Base*, the *Integration* step relays such document and does not create a new one. It is worth noticing that when a new document is created at the *Integration* step, the resulting document is both stored in the *Content Base* for future use and used as input for the *Publishing* step.

Publishing represents the transmission of a document to the viewer according to the received query/request. The published document, at the *Interpretation* step, is instantiated into a presentation, according to the viewer context. The viewer context is composed by properties expressing (but are not limited to):

- (i) viewer information such as language, location, gender, age, etc;
- (ii) characteristics of the exhibition device(s) available such as screen size, playable media item types, network availability, etc;
- (iii) viewer preferences such as predisposition to interaction, preferred media items, etc.

Therefore, the document to be executed is the one most suited to the viewer characteristics and preferences and the available exhibition device(s). Moreover, at the interpretation step, it is also possible for a document to be translated from the language it was created (and stored) into a language to be used in its presentation as presented in [Silva et al. 2013].

Scheduling keeps the presentation synchronization, exhibiting each content at the correct time, space and exhibition device. *Exhibition* acts as an interface between the presentation and the viewer, relaying viewer interactions back to *Scheduling*. During execution, in reaction to event occurrences, the *Dynamic* step accesses Application Programming Interfaces (APIs) available at *Scheduling* for editing the presentation. Examples of such APIs are Document Object Model (DOM) [W3C 2000], SMIL DOM [W3C 1999a], NCL Live Editing Commands [Soares et al. 2006] and Web Animations [W3C 2014]. Besides gathering or exchanging information with the network, the *Dynamic* step may store new content in the *Content Base* and templates in the *Template Base* or may query new information from them.

2.1.1 Use Cases

Usually, when talking about a multimedia document life cycle, the literature breaks it down into two sides: the *server/broadcaster* side and the *client/viewer* side. In the *client* (or *viewer*) side, we have essentially the steps related to the document execution. The client can be made as complex as possible - on the other hand as simple as possible - regarding how many steps we push into it. The *server* (or *broadcaster*) side keeps the remaining steps. The simpler the client is, the more complex the server is, and vice-versa.

A common division seen in Digital TV and IPTV environments [ETSI 2008, ATSC 2009, ARIB 2014, ITU 2009] and also in the web, is to use the *Publishing* step as a division between the server and client sides. The work in [Lemlouma and Layaïda 2004] presents a different division, following the idea of having simple clients, where just the *Exhibition* step is in the client. In such a scenario, a document is created and its execution is handled by the server. The client receives media object samples to be presented, for a given moment, and communicates presentation and viewer interaction events back to the server.

This section presents works that implement parts of the life cycle presented in Figure 2.2. The works presented cover different areas in multimedia research and inspired the life cycle proposed here. The following paragraphs present those works together with a discussion about the steps they implement.

The work in [Sarkis et al. 2014] presents an approach for splitting a multimedia application in two screens. This work fits in the life cycle presented in Figure 2.2 as follows. At the *Integration* step, document elements are annotated with information about the screen where they will be presented, according to the received query. The annotated

document is sent, in the *Publishing* step, to both clients identified in the received query. Following, at the *Interpretation* step, a given client splits the document maintaining the elements to be presented in the client screen and “hiding” the remaining elements. During execution, the *Dynamic* step receives viewer interaction and maintains the communication among both screens so that the document as a whole maintains its synchronization.

The work in [Laborie et al. 2011] proposes an approach for the client-side adaptation of documents. It implements the *Interpretation* step so that a given document is translated into an abstract model, as a set of objects and spatio-temporal relations among them. Relations in this model are represented by constraints. Besides relations, constraints may also represent author guidelines. The adaptation process combines document constraints with constraints representing the *Viewer Context*. Given that the resulting constraint set is satisfiable, the document can be adapted without changes. On the opposite case, document relations have to be changed until the constraint set turns satisfiable.

Both works presented in [Teixeira et al. 2012] and [Soares et al. 2012] act at the *Dynamic* step. The work in [Teixeira et al. 2012] proposes an approach for document annotation and the work in [Soares et al. 2012] proposes an approach for client-side adaptation of documents.

In [Teixeira et al. 2012], the *Dynamic* step is implemented as an extension of the client middleware to allow the user to insert text, sound or images in a presentation. It captures viewer interaction events to create annotations about the TV content being presented. The annotations produced are synchronized to the main presentation by means of an NCL document. Although not explored in the paper, the resulting document, i.e., with the annotations created by the viewer, can be sent back to the *Content Base* so that it can be provided to other viewers as a response to future queries or even shared with other clients watching the same presentation.

In [Soares et al. 2012], the *Dynamic* step is implemented using a Lua script that, reacting to some presentation or viewer interaction event, triggers a request to the *Content Base* for gathering new video objects related to the one being presented at the time. It then creates a new document with the new content to be included in the presentation by comparing the old and new documents determining the changes to be produced in order to turn the former in the latter. Following, it will send a set of commands using the NCL Live Editing Commands [Soares et al. 2006] API, for producing the changes on the fly.

2.1.2 Validation Requirements

In order to keep the consistency of a document along its life cycle, validation is necessary at different steps. In this section, we discuss the role of validation across the steps in Figure 2.2 together with its requirements.

Initially it is important to guarantee that content and template(s) combined at the *Integration* step yield a consistent document. This can be achieved by performing validation at *Integration* step over properties associated to both content and templates, besides properties over the document being created, which we refer as author guidelines. Validation at *Integration* step should be able to support different multimedia integration paradigms (relation, event, composition, etc). It is worth mentioning, however, that some declarative multimedia languages provide the possibility of using auxiliary scripting languages (such as Lua or NCL and JavaScript for HyperText Markup Language (HTML)) for implementing facilities not supported by the given language. The validation approach proposed in this work does not tackle such scripting languages, focusing only on the declarative multimedia language used for the document creation.

According to the paradigm used at *Integration* step, it is possible to validate the document at the same time it is described or not. For example, in a constraint-based approach, such as [Jourdan et al. 1998], relations may be validated as they are created. On the other hand, in an event-based approach, such as [dos Santos et al. 2015a], validation may have to wait for the creation of a set of relations before being performed. At this step of the document life cycle, validation has a *preventive* role, investigating if a document can be instantiated and, once presented, may not lead to execution errors. In case an error is found, a validation tool can either: (i) automatically “correct” the document or (ii) give feedback to the author to “correct” the document (this is represented in Table 2.1 by +/-). Moreover, validation is also *partial* since the content (or even layout) of a document may be changed dynamically during execution.

After *authoring/generation*, performing validation along the document life cycle always (or almost always) has to yield an executable presentation, even by means of some automatic “error correction”. Errors at this point would interrupt the current requested document without presenting anything to the viewer. Although giving feedback to the author would be interesting for future improvements, it would not help correcting the document in the process of being presented. Moreover, time is an important matter, since taking a long time to yield a result would make the viewer wait too long for the start of the presentation (*Interpretation*) or lead to presentation lags (*Scheduling*).

During the instantiation process at *Interpretation*, it is necessary to guarantee that a document can be instantiated into a presentation taking into account the viewer context. This is achieved by validating the combination of both properties associated to the document and the ones expressing the viewer context. According to the way a document is described, and given that more than one presentation can be instantiated from a document, validation can be seen as a way to provide personalization and help document adaptation, by choosing the document instance more suited to the viewer's context. Besides, given that both adaptation and the validation we present may be done using the same techniques, we refer to adaptation as a special case of validation.

Once a document is executing, it may evolve either by viewer interaction or incremental changes performed by the *Dynamic* step. Similar to collaborative writing [Sun et al. 1998], at this point, it is important to provide *intention preservation*, to avoid turning the presentation into something incoherent with the author guidelines. The same way validation is performed in the *Interpretation* step, at *Dynamic* it should be able to combine properties representing the presentation to properties representing incremental changes in order to always return an executable presentation. Thus, an incremental change may be ignored if it yields an inconsistent presentation. Depending on the event that triggers the *Dynamic* step, time may be an important issue or not (this is represented in Table 2.1 by +/-). When the *Dynamic* step has to perform some incremental change due to viewer interaction, taking a long time to yield a result would lead to presentation lags. On the other hand, when an incremental change is not triggered by viewer interaction, it (and its validation) may be performed in advance, thus avoiding presentation lags.

It is important to highlight that author guidelines are properties to be preserved along the document life cycle. Due to personalization (at *Interpretation* step) or to incremental changes (at *Dynamic* step), relations expressing the document layout may be changed. Author guidelines represent a way to guide such changes in order to provide *intention preservation* [Sun et al. 1998] across the document life cycle.

During the *Scheduling* step, it is also important to support changes in the spatio-temporal layout of the presentation according to viewer interactions, such as interactive animations. One example of interactive animation is the change in the spatial layout of a presentation according to a dragging event. Given that the presentation (and its exhibition environment) supports such kind of viewer interaction, it is important to guarantee that the viewer will not perform an interaction that yields inconsistency in the presentation. This may be achieved by placing validation at the *Exhibition* step, combining properties

related to the document and the ones describing the interactive action. Thus interactive actions that turn the document layout inconsistent could be disabled.

Table 2.1 summarizes the validation requirements for each step in a multimedia document life cycle. Symbol ‘+’ indicates an important requirement and symbol ‘-’ indicates a less important requirement.

	Integration	Interpretation	Scheduling	Dynamic
Automatic correction	+/-	+	+	+
Author feedback	+	-	-	-
Response time	-	+	+	+/-

Table 2.1: Validation requirements for life cycle steps

2.2 Multimedia Document Validation

As presented in Section 2.1, a multimedia document describes a set of possible presentations, restricted at the execution step, when a single presentation is chosen. The resulting presentation, therefore, represents the execution of a given document. Considering authoring/generation and execution, a document may be defined in two different forms, one representing its description and another representing its execution. The first represents the set of constructs in a given integration language used to define a document. The second represents the resulting behavior of those constructs when such document is executed.

Section 2.1 also presented the importance of validation at different steps of a document life cycle, as a way to guarantee the consistency of a given document. As it happens for a document, inconsistencies may be perceived in two different forms, as certain properties are not satisfied. We shall classify document validation properties as *structural* and *behavioral*. In Section 2.2.1, we define *structural* properties and, in Section 2.2.2, we define *behavioral* properties. In the last section, *structural* and *behavioral* validation are discussed.

2.2.1 Structural Properties

Since its creation, the eXtensible Markup Language (XML) [W3C 2008a] has been used as concrete syntax for multimedia languages. XML elements are used for representing media items and their synchronization relationships. Examples of XML-based declarative

multimedia languages are NCL [ITU 2009] and SMIL [W3C 2008b]. NCL is part of the standard for Digital Television (DTV) in Brazil and Latin America [ABNT 2011]. It is also part of the ITU standard for IPTV services [ITU 2009]. SMIL is a standard for multimedia presentations on the web [W3C 2008b].

Syntax rules defined by a multimedia language grammar and its static semantics induce a set of structural invariants representing properties a document should follow to be considered structurally consistent. We identify in related work [Araújo et al. 2008, Neto et al. 2011] the following structural properties:

- The lexical and syntactic structure of a document should be well-formed and in accordance to the authoring language grammar. For example, XML tags [W3C 2008a] must be correctly closed and follow the language *namespace*.
- Every document element must only contain valid child elements and in the correct cardinality.
- Every document element must only contain valid attributes and the required ones must be defined.
- Every element identifier, when it is the case, must be unique.
- Attributes with related values must follow the constraints defined by the multimedia language. For example, NCL defines attributes *type* and *subtype* for the *transition* element [ITU 2009]. For every value of attribute *type*, NCL defines a set of values that may be used for attribute *subtype*. Another example are NCL attributes *component* and *interface*, where a given element referred in attribute *interface* has to be a child of the one referred in attribute *component*.
- References between elements must follow the constraints defined by the multimedia language. For example, NCL defines attributes that may refer to just one kind of element or a group of elements. In element *media*, attribute *descriptor* can only refer to *descriptor* elements, while attribute *refer* can only refer to *media* elements.
- Elements inside a composition can not refer to the ones outside the same composition.
- A composition can not create a nesting loop. That is, it can not nest itself directly or through other compositions.

- An element can not create a reuse loop. That is, it can not reuse itself directly or through other elements.

2.2.2 Behavioral Properties

Behavioral properties are used to verify the existence of inconsistencies in the resulting spatio-temporal layout of a document, which is given by the combination of the layout induced by the dynamic semantics of constructs in a multimedia language (among which we have relations) and properties representing author's guidelines or exhibition device characteristics.

The layout of a document is commonly defined in two axes, temporal and spatial. In the temporal axis, media items are placed in time either with absolute values or in relation to other media items or event occurrences, such as viewer interactions. In the spatial axis, media items are placed in relation to the screen size, another media item or into predefined channels.

Usually, media item positions are defined relative to the screen, in absolute values (pixels) or relative (percentage) values. Although their initial position is static, it may change over time. Some multimedia languages allow changing media item position in response to the occurrence of events in the presentation. Such a change may comprise: moving a media around by changing, for example, its left/top attributes; or resizing a media by changing, for example, its width/height attributes. Such changes may be instantaneous or incremental over a time interval.

It is possible that the resulting spatio-temporal layout does not fit the author's expectations due to incorrect use of multimedia language constructs. Author guidelines represent document expected behaviors and shall be defined together with a document in order to avoid mismatches between "what the author wants" and "what the author gets". We identify in related work the following guidelines:

- Every document element has to be reached during this document execution.
- A document element execution must end.
- The execution of the document as a whole must end. The document execution ends if every document element presentation ends and there is no execution loop (for example a media item restarting its presentation every time it ends).

- Two distinct document elements should not use the same presentation device resource (screen position or audio channel, for example) simultaneously, avoiding their superposition.

Although different approaches in the literature also identify those behaviors as expected [Santos et al. 1998, Oliveira et al. 2001, Júnior et al. 2012], they may not be the author's intention. For example, if a document represents a game, the author may expect it to never end its execution.

The previous guidelines represent general guidelines to be applied to every document. Besides, the author may define specific guidelines for a given document. For example, he/she may define that two given media objects A and B must be presented one after the other during document execution. In previous works, we refer to such kind of guidelines as author-defined properties [dos Santos 2012, dos Santos et al. 2013a].

At *authoring/generation*, a usual attempt to verify if a document follows the author's guidelines is through document simulation. In this process, the author executes the document several times and, taking the role of the *Viewer*, observes if the resulting spatio-temporal layout fits its expectations. This process, however, is usually not *effective* since several executions would be necessary for the verification of undesired behaviors, and may be *incomplete*, from a correctness perspective, since the computations representing the document presentation may be infinite. Moreover, in the case where a document is automatically generated inside a production cycle, simulating the execution of the document would be costly, if possible.

2.2.3 Structural and Behavioral Validation

Structural inconsistencies arise when a document does not follow the properties presented in Section 2.2.1. On the other hand, *behavioral* inconsistencies arise when document relations and/or author's guidelines in combination are inconsistent, as presented in Section 2.2.2.

The *structural* validation of a document is important since a structurally inconsistent document may not be executable or even readable. The *structural* validation should be performed before the *behavioral* one, since the former may turn the latter impossible to be performed.

In previous works [dos Santos 2012, dos Santos et al. 2013a], we presented an approach for the structural validation of documents based on an equational theory $\mathcal{E} = (\Sigma, E)$,

where Σ represents the abstract syntax and E the structural properties presented in Section 2.2.1 of a given multimedia language. Such validation was also refined in recent works about document validation [dos Santos et al. 2012a, dos Santos et al. 2013b, dos Santos et al. 2015a]. The validation approach presented here relies on those existent approaches for the structural validation, thus focusing only on the behavioral one.

The type of *behavioral* validation provided depends on the type of relations and properties supported by the validation tool. In this section, in order to illustrate it, we present a series of small examples of a document d that presents two media items A and B . For each example, the position of A and B is defined in absolute values. Each figure represents an example, where the dashed rectangle represents the screen (at some moment) and solid rectangles represent the region where a media item is presented. Arrows between two screens represent a time lapse and arrows between regions inside the same screen represent movement, which can be done incrementally over a time period whose duration is presented over the arrow.

Case 1. This example describes a static spatial layout, i.e., A and B do not change their position over time. This case is a pure temporal example, where A is presented (just) before B in time. Figure 2.3 presents the spatio-temporal layout the author perceives from document d .

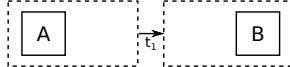


Figure 2.3: Case 1 spatio-temporal layout

Considering the above temporal layout example, most of the author's expectations can be described with temporal properties. For example, the author may want to ensure that, for this document, A *before* B , or that *not(A together B)*. However, it is also possible for the author to express spatial properties. For example, the author may wish to ensure that A *sideof* B , or A *samesize* B . Since the spatial layout is static, spatial validation can be done over the initial position of media items.

Case 2. This example involves the change over time of the spatial layout of a multimedia document. In this example, A has a fixed position and B moves across the screen, changing its position incrementally over t_1 time units. Figure 2.4 presents the spatio-temporal layout the author perceives from document d .

Validating the spatio-temporal layout of such an example is not so simple as statically validating the spatial layout and, in parallel, validating the temporal layout. For example,

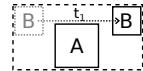


Figure 2.4: Case 2 spatio-temporal layout

suppose the author wishes to ensure that, at some point, while moving across the screen, B will overlap A . This requires verifying if, in at least one moment during the document execution, A and B will overlap. Such kind of property has to be encoded by composing temporal and spatial properties, such as $\text{somepoint}(A \text{ overlap } B)$.

The examples above present documents where (i) the spatial and temporal layout are independent or (ii) the spatial layout is dependent on the temporal one. As seen above, we call the second case a spatio-temporal layout.

A purely temporal validation acts just in the temporal axis, verifying the consistency of temporal relations among media items. On the other hand, a purely spatial validation acts just in the spatial axis, verifying the consistency of spatial relations among media items. The spatio-temporal validation acts in both axes simultaneously, considering the case where media items change their position in relation to time. In our proposal, behavioral properties can be used for spatio-temporal validation [dos Santos et al. 2015b].

Chapter 3

Related Work

The literature is rich on the discussion about document validation. In general, each work focuses *either* on structural or behavioral validation. Therefore, this chapter presents related work separating structural validation, in Section 3.1, from behavioral validation, in Section 3.2.

3.1 Structural Validation Works

3.1.1 NCL-Inspector

NCL-Inspector [Honorato and Barbosa 2010] is a tool based on other tools for code quality critique that supports the author in terms of code quality. The idea behind it is to search the NCL code for coding problems or even suggesting best programming practices. Its full documentation [Honorato 2010] presents several best practices identified in NCL authoring and problems reported by the NCL community.

The code validation, called author inspection, is done following a set of rules. Each rule corresponds to a coding problem or best practice the author should follow. Every rule set created can be used to extend the tool capabilities and is stored as a rule repository, making it possible for authors to exchange rules as needed.

Each rule represents an NCL code pattern and an action to be performed when that pattern is found. The specification of a rule may be done using XML Stylesheet Language Transformations (XSLT) [W3C 1999b] and Java languages. During a rule creation, the author may test it through an available mini-test framework. That framework follows a Test-driven Development (TDD) approach.

The validation of each rule can be done either in the document Abstract Syntax Tree (AST) or over the code text. The idea is to provide a way to validate NCL document elements as a whole and their hierarchy, or specific text details, for example, the use of the tabulation character (\t) for code indenting. In the first approach, the document is parsed creating the AST that represents its XML structure. Then the tool walks through the AST searching for patterns of existent rules. Whenever a pattern is found, the action described in the rule is executed presenting an error or warning message to the author. A similar approach is used when validating the document regarding its code text. In that case, the difference is that AST is not used.

3.1.2 NCL-validator

NCL-validator is a tool following the validation process presented in [Araújo et al. 2008]. The process, defined for validating documents created with the NCL language, is divided into four steps called: (1) lexical and syntactic validation, (2) structural validation, (3) contextual and reference validation, and (4) semantic validation. As we will discuss later, NCL-validator does not implement the semantic validation step.

The (1) lexical and syntactic validation investigates the lexical and syntactic structure of the XML document. It searches for XML tags that are not correctly written (e.g., a tag is not closed) and if there are not tags that are not defined in the NCL language grammar. The (2) structural validation investigates the structure defined by the XML elements inside a document. It validates if all NCL elements have only valid attributes and if the required ones are defined. It also validates if the children of a given NCL element are correct and in the correct cardinality. The (3) contextual and reference validation checks if references among elements are correct and if they are in the same scope. It means that attributes that make reference to other NCL elements must refer to an element that exists and of the type required in the NCL grammar. Besides, elements must be inside the same NCL context (which represents a scope). The (4) semantic validation investigates if document parts are not reached during the document execution. This can happen because of missing links in the NCL document or alternatives (in content control elements) that are never evaluated as *true*. It is worth highlighting that these steps were used as a base for the set of desirable properties we identified in [dos Santos 2012].

Although NCL-validator is an implementation of that validation process, the semantic validation step is not implemented. According to the paper, it was left as future work since it does not endanger the NCL document validation. NCL-validator is used as a library

by NCL-Eclipse [Azevedo et al. 2009] and NCL Composer [Lima et al. 2010] authoring tools. When authoring errors are found, the tool returns error or warning messages to the author identifying the correspondent problem found. It is worth to notice that this is one of the few validation tools currently available for NCL documents.

Focusing on extending NCL-validator for the incremental validation of NCL documents, a new version of the tool is presented in [Neto et al. 2011]. Different from the prior version, it now defines a metalanguage for representing NCL language grammar rules. The idea of using such metalanguage is to make the tool independent of the NCL profile used and possibly the multimedia authoring language used.

This metalanguage is composed by four primitives. The primitive *ELEMENT* describes an element of the NCL language. The primitive *ATTRIBUTE* describes an attribute of a given element. The primitive *REFERENCE* describes a reference between two elements. The primitive *DATATYPE* describes a data type used in an element attribute and a regular expression that gives its possible values.

The set of primitives describing the grammar of a language (NCL in the case) is used as a set of rules that must be satisfied by a document. Every rule is applied to the element it describes and when a rule is not satisfied, an error or warning message is presented to the author. For the incremental validation of NCL documents, only the rules related to the elements recently modified and the ones related to them are applied. An additional structure is used for identifying the elements that have changed since the last validation and the ones that may be influenced by their changes.

Different from the work in [Neto et al. 2011] our validation is not intended to be performed at document fragments, but in the complete document. Moreover, structural properties are coded directly in the metalanguage elements used to describe NCL, which makes it difficult for the user to create its own set of rules, as it is possible in [Honorio and Barbosa 2010] and in this work.

3.1.3 VAMP

VAMP [Troncy et al. 2010] is an approach to validate MPEG-7 descriptions [ISO/IEC 2001]. The MPEG-7 standard defines a set of constraints to the use of multimedia descriptions. Besides those constraints, the author also has to follow the ones defined by the MPEG-7 profile [ISO/IEC 2005] in use. However, as stated in [Troncy et al. 2010], those constraints do not prevent variability in the use of descriptions. Different descriptions can

be used with the same meaning and sometimes the same description can be used with different meanings, depending on the author and annotation tool used. The idea behind VAMP is to reduce the variability in the use of MPEG-7 descriptions, therefore, allowing the interoperability among different annotation tools. The tool also aims at validating descriptions use to verify constraints defined by the MPEG-7 standard and the MPEG-7 profile used. VAMP authors call a violation every time a description is used with a meaning different from the one specified in the standard or in the profile used.

Although both MPEG-7 and MPEG-7 profiles define an XML Schema [W3C 2004b], it is not enough to prevent all possible violations. In [Troncy et al. 2010], the authors present a set of violations that yield perfectly valid documents with respect to the MPEG-7 XML Schema. The approach used by VAMP is to use an ontology Web Ontology Language - Description Logic (OWL-DL) [W3C 2004a] for representing the concepts described in the standard and profile. Besides the ontology, logical rules (Horn clauses [Baral and Gelfond 1994]) are used to represent the constraints in the use of description elements.

3.1.4 Schema Validation

Whenever a multimedia language is XML-based, language elements and their hierarchy are usually defined using an XML Schema [W3C 2004b]. A common approach for validating the structure of an XML document is to use XML Schema-based validators. Those validators are used to verify if a given document satisfies the restrictions defined in the language Schema.

It is important to notice, however, that restrictions present in a language Schema are defined over element types, such as:

- an element of a given type must define a given attribute;
- an element of a given type must have a given element as child;
- the value of a given element attribute must be of a given type (string, number, another element id, etc).

One can notice that those restrictions do not fit for constraints about element instances, such as:

- both elements A and B must refer to element C;

- element A, child of B can only refer to C iff C is also a child of B.

That is why XML documents with authoring errors may yield perfectly valid documents with respect to a language Schema, such as the examples presented in [Troncy et al. 2010]. This kind of verification is important for our work and other works presented in this section. That explains why other techniques are used together or not with Schema-based validators.

3.2 Behavioral Validation Works

We classify the related work presented in this section according to the reasoning principle applied for document validation. The first group of papers (Section 3.2.1) relates to validation by investigating the document state over its execution. This may be done by reachability analysis or the application of axioms over the document state. The second group of papers (Section 3.2.2) relates to validation by checking the consistency of a set of constraints. Some of the approaches discussed do not primarily address validation but could be considered as such.

3.2.1 Reachability of States

3.2.1.1 Santos et al Approach

In [Santos et al. 1998], the authors present an approach for the temporal validation of multimedia documents. The approach presented in the paper did not assume, *a priori*, a specific model to express and compose temporal relations, but used generic authoring models. The document to be validated is translated, automatically, into the real-time process algebra framework RT-LOTOS. In order to translate the multimedia document into Real-Time LOTOS (RT-LOTOS) processes, general mapping rules were used. Also, the definition of RT-LOTOS process libraries were used for specifying the behavior of reusable document parts. The modularity and hierarchy of RT-LOTOS allow the combination of processes specifying the document presentation with other processes modeling the available presentation platform.

A minimum reachability graph is built from the RT-LOTOS formal specification such that each node in the graph represents a reachable state and each edge the occurrence of an event or temporal progression. The validation is achieved by verifying, for example, if the state corresponding to the end of the presentation can be reached from the initial

state. Similarly, verifying if a media item will be executed is performed by determining if a state where it is being executed is reachable from the initial state.

The paper presented different possible undesired behavior situations to be validated, which are: qualitative, if they do not depend on an object duration, and quantitative, if they depend on an object duration. The possible undesired behaviors can also be intrinsic if they do not depend on the platform where the document is presented and extrinsic if they depend on it. In the last case, it is considered if platform resources are blocking or non-blocking. Blocking resources are the ones that can not be used by two objects at the same time. An audio channel, according to the paper, is an example of blocking resource. In addition, presentation component delays were also considered. Regarding those delays, the document behavior may become undesired. The possible undesired situations listed in the paper were used as a basis for the set of desirable properties presented in Section 2.2.2 [dos Santos 2012].

The tool presented in [Santos et al. 1998] can verify Nested Context Model (NCM) [Soares et al. 2000] and SMIL [W3C 2008b] documents. It is worth noticing that the tool was created for earlier versions of both document types. Besides, the tool itself is not available for practical tests.

3.2.1.2 caT

The context aware Trellis (caT) system [Na and Furuta 2001] is an evolution of Trellis [Furuta and Stotts 2001] to provide adaptation of a document presentation according to the user context. Like in Trellis, in caT an author creates a multimedia document using Petri nets. Each place of the Petri net represents a media item, while transitions represent synchronization relationships among media items.

The work presented Petri nets as a good candidate for modeling multimedia documents, since its synchronization is easy to model and allows the validation of document properties. Basic Petri nets, however, are not convenient for representing and validating complex systems, since their tokens do not have identity. To overcome this limitation, the paper proposes Petri nets with identifiable tokens, called High-Level Petri nets.

The caT system provides the separation among document specification and presentation, allowing multiple presentations for a document specification. To reduce the authoring graphical complexity and improve net reuse, caT incorporates hierarchical Petri nets. The authoring tool supports a tool for the validation of hierarchical Petri nets, through

its reachability graph.

The validation tool builds the reachability tree of the validated document. The author defines limit values for the occurrence of dead links (transitions that may not be triggered), places with token excess, besides other options, as the validation maximum time. Then the tool investigates the existence of a terminal state, that is, if there is a state where no transitions are triggered. It also investigates the limitation property, that is, if no place in the net has an unlimited number of tokens and the safeness property, that is, if each place in the net has a token. The limitation validation is important since tokens may represent scarce system resources. It is worth to highlight that the properties validated by caT were used as a basis for the set of desirable properties presented in Section 2.2.2 [dos Santos 2012].

In the paper the authors also present a browser for the execution of documents created using caT. This browser takes into account information about the viewer in order to provide adaptation of the document to be presented. Adaptation is provided with tokens associated to viewer information. According to the document Petri net, those tokens will trigger the presentation of specific content.

3.2.1.3 HMBS

Hypermedia Model Based on Statecharts (HMBS) [Oliveira et al. 2001] is a model where an author creates multimedia applications through statecharts. HMBS is a generalization of hypertext models based on hypergraphs and, according to the authors, its use can be seen as a way to encourage a structured development, since the document structure is defined before content is added to the model. An HMBS presentation is described by a statechart, where states represent pages (i.e. the information presented to the user) and transactions and events represent a set of possible link activations.

The validation of an HMBS presentation is performed over a reachability tree, which is built from the presentation statechart. From the reachability tree, it is possible to determine if a given page is reachable or not from a given initial state. Similarly, it is possible to determine if a group of pages is presented simultaneously or not, by searching state configurations containing the states associated to those pages. The reachability graph also allows the detection of configurations from which no other page may be reached or that present cyclical paths.

The reachability graph also allows determining the maximum number of simultaneous

windows necessary to present the application. Analyzing the graph and determining the maximum number of active states, it is possible to determine a better layout for the application. It is worth to highlight that the properties validated by HMBS were used as a basis for the set of desirable properties presented in Section 2.2.2 [dos Santos 2012].

3.2.1.4 Felix Approach

In [Felix 2004], the author presents a formal approach for the verification of behavioral properties of concurrent systems. It presents a notation for the description the components of a system and their temporal constraints. Such a description is transformed into a timed automata net that indicates the system temporal behavior. Its verification is done through model checking using UPPAAL [Larsen et al. 1997].

As a use case, this work uses the notation it proposes for the description of NCL temporal relations and temporal dependencies among nodes and their anchors. Such a description is transformed into a timed automata net that indicates the document temporal layout. The transformation creates a state machine for each media item and a synchronizer machine for each link declared in the document. A synchronizer machine is used for tying together the occurrence of events in media state machines. In this work, only presentation and selection multimedia events are considered while creating the timed automata net representing a given document. Attribution events are not supported. User interaction itself is modeled as an automaton indicating when the user will interact with parts of the document.

The validation of an NCL document is performed over the timed automata net that represents it using temporal logic formulas defined by the author. The work also presents a tool where the author can define the temporal-logic formulas to be used for validation.

3.2.1.5 Gaggi and Bossi Approach

In [Gaggi and Bossi 2011], authors define a formal semantics for SMIL temporal aspects through a set of inference rules inspired by Hoare logic. The rules describe the document state before and after the execution of a given SMIL construct. Thus, in the authoring phase, the structure of a SMIL document may be enriched with assertions expressing temporal properties. The validation of the SMIL document, therefore, is based on those properties.

The verification is done during the authoring phase, whenever the author wants or

when he saves the application. This is done to diminish the occurrence of error messages during the application creation. The work presented the choice of inserting temporal assertions in a SMIL document as a way of diminishing the validation complexity, since this approach does not require the translation of the document being created to some formalism and then perform its validation.

The validation of a document is performed by the application of axioms, also defined in the proposed semantics, that verify if a given construct or set of constructs correctly changes the document state. Otherwise, it presents to the author the problem found so it can be corrected. Another application resulting from the defined formal semantics is the concept of equivalence, which guarantees that two sets of SMIL constructs may be replaced, without changing the presentation layout.

3.2.1.6 SMIL Builder

In [Bouyakoub and Belkhir 2011], authors present an incremental authoring tool for SMIL documents called SMIL Builder. It checks the consistency of the temporal layout of a document whenever the author performs a modification in the document. The idea is to check if a given change in the document may turn it inconsistent. If so, the change is rejected and an error report is presented to the author.

Both the incremental editing and consistency checking are supported by the H-SMIL-Net model [Bouyakoub and Belkhir 2008]. H-SMIL-Net extends Petri Nets in order to fully represent SMIL temporal concepts. Inconsistencies are detected either by construction over the Petri Net, for example, more arcs than possible for a transition, or by verification of the firing times of transitions in the model.

3.2.1.7 Junior et al Approach

In [Júnior et al. 2014], authors propose a model-driven approach for the behavioral validation of NCL documents. In the proposed approach, NCL documents are translated into Petri Nets in a two-step transformation. The transformation presented in [Júnior et al. 2012] is done as follows. In the first step, the document is represented in a language called FIACRE as a set of components and processes (representing the behavior of a component). The second step transforms the FIACRE representation into a Petri Net.

The document validation is then performed over the resulting Petri Net using a model-checking tool and temporal logic formulas representing the properties to be validated.

Spatial validation is briefly discussed in [Júnior et al. 2012] and is performed over the document initial positioning.

3.2.1.8 King et al Approach

In [King et al. 2004], they define extensions for the SMIL language allowing document authors to describe how the spatio-temporal layout should change in reaction to events. Changes in position and size are described by a set of expressions, which may consider the state of the presentation. The paper presents an approach for calculating at runtime the value of such expressions, and therefore spatial layout changes to be performed. Although such an approach does not refer to validation, it is an interesting example of how to parameterize the rules that specify the spatial layout of a presentation by time or event occurrence.

3.2.2 Constraint Checking

3.2.2.1 Bertino et al Approach

In [Bertino et al. 2005], they proposed an authoring model based on constraints. A document in that model consists of several topics, where each topic is composed by semantically-related media items. All relations, temporal, layout and structural, are specified in a single step. So, the document author defines a set of high-level constraints that will be used by the system to automatically group media items into topics. The presentation generation process is responsible for three main tasks: consistency checking, presentation structure generation and topics generation.

The system enlarges the set of constraints with others that, even not explicitly defined, are consequences of the constraints defined by the author. Consistency checking is then performed over the constraint set. If an inconsistency arises, the system applies relaxation techniques, to reduce the constraint set to a consistent one. When such a reduction is not possible, author review is required.

The presentation structure generation process creates a direct graph that represents its structure. Each vertex of such graph represents a topic and each edge a connection between topics. This process always returns a consistent graph, otherwise, the author should review the declaration. After this step, the system relates media items to topics and builds, for each topic, the spatial layout and the temporal sequence of media items belonging to it.

3.2.2.2 Laborie et al Approach

In [Laborie et al. 2011], they present an approach for the automatic adaptation of the layout of a document according to the exhibition device. The proposed approach creates an abstract description of a document that will be used for document adaptation. In such a description, a document is represented as a set of objects and constraints representing temporal and spatial relations among objects.

Besides the document itself, the proposed approach also takes into account a profile comprising device constraints together with viewer preferences. Device constraints may relate to the ability of the exhibition device to present multiple video objects at a time, for example.

Given the set of potential document executions \mathcal{M}_s given by the abstract description and the set of potential executions \mathcal{M}_p given by the profile, the adaptation process calculates $\mathcal{M}_s \cap \mathcal{M}_p$ to determine if some adaptation is required or not. In case the document has to be adapted, the goal is to change document relations such that it now complies with the given profile. Moreover, adaptation is performed such that the (behavioral) distance from the previous declaration is minimum.

3.2.2.3 Elias et al Approach

In [Elias et al. 2006], they propose an authoring model based on constraints. It defines two operators, *TEMPORAL* and *SPATIAL*, to model temporal and spatial relations, respectively. Each operator allows the author to define a priority value, so that in order to maintain the consistency of the constraint set, whenever necessary, constraints are removed according to this priority value. In case two inconsistent constraints have the same priority, relaxation techniques are applied to determine the one to be removed. Besides the verification of inconsistencies among constraints, this approach also enables the author to verify if the constraint set is incomplete, that is, if there is one or more media items that are not reached during presentation.

The consistency checking is done by finding the minimum spanning tree T in the constraint graph. Constraints that create cycles are removed to maintain the acyclic nature of T . Completeness checking is done by searching all media items reachable from the first media item. If this search returns the vertex set of T , then all items are reached directly or indirectly from the initial one. Otherwise, the author has to define constraints to make the constraint set complete.

With the use of the *SPATIAL* operator, it is possible to determine if A overlaps B and vice versa. The spatial consistency is checked the same way as the temporal one. Each spatial constraint is associated to an interval, such that a given (spatial) constraint has to be satisfied inside a given interval. Although [Elias et al. 2006] does not define spatial attributes in function of time, it represents an example of constraint parameterized by time.

3.2.2.4 Belouaer and Maris Approach

In [Belouaer and Maris 2012], they present an SMT [Barrett et al. 2009] approach for solving spatio-temporal planning problems. A set of constraints modeling the spatial disposition of items and their hierarchy is used to describe both the initial state of a given problem and its goal. Other constraints model actions that change the spatial position of items.

Actions may define an inherent duration and also at which moment (in time) they should be applied. By solving the problem, taking into account the constraints representing actions, it is possible to verify if the goal can be achieved or not. Although this work does not refer to documents, it is an interesting example on how spatial constraints can be parameterized by time in order to cover both spatial and temporal dimensions.

3.3 Related Work Comparison

This chapter presented related work considering both structural and behavioral validation of multimedia documents. The works presented in [Honorato and Barbosa 2010, Araújo et al. 2008, Neto et al. 2011, Troncy et al. 2010, W3C 2004b] are related to the structural validation of multimedia documents.

In Section 3.2, behavioral validation approaches are discussed according to the reasoning principle applied and its cover in both temporal and spatial axes. Table 3.1 summarizes those works classification and its position in the document life cycle according to its three main steps: *authoring/generation*, *instantiation* and *execution*.

The works presented in [Na and Furuta 2001, Oliveira et al. 2001] are related to the behavioral validation of multimedia documents. However, given that the document authoring is performed over Petri nets and statecharts, respectively, we may consider them to implicitly perform the structural validation. The remaining works do not discuss

	Temporal	Temporal + Spatial	Spatio-temporal	State reachability	Constraint checking	Authoring/generation	Instantiation	Execution
[Santos et al. 1998]	✓			✓		✓		
[Na and Furuta 2001]	✓			✓		✓	✓	
[Oliveira et al. 2001]	✓			✓		✓		
[Felix 2004]	✓			✓		✓		
[Gaggi and Bossi 2011]	✓			✓		✓		
[Bouyakoub and Belkhir 2011]	✓			✓		✓		
[Júnior et al. 2012]	✓			✓		✓		
[King et al. 2004]		✓		✓				✓
[Bertino et al. 2005]	✓			✓		✓	✓	
[Laborie et al. 2011]	✓			✓		✓	✓	
[Elias et al. 2006]		✓		✓		✓		
This work	✓	✓	✓	✓	✓	✓		✓*

Table 3.1: Related work comparison

if some kind of structural validation is performed before the behavioral one.

As can be seen in Table 3.1, the works presented in [Santos et al. 1998, Na and Furuta 2001, Oliveira et al. 2001, Felix 2004, Gaggi and Bossi 2011, Bouyakoub and Belkhir 2011] present a purely temporal approach, where the validation of a document is performed by investigating its state over its execution. In [Santos et al. 1998, Na and Furuta 2001, Oliveira et al. 2001, Felix 2004], it is done by reachability analysis, in [Bouyakoub and Belkhir 2011] it is done by verifying the consistence of the underlying Petri Net and in [Gaggi and Bossi 2011] by analyzing if the document state changes according to some axioms. Validation of the spatial layout of a document is not discussed in those papers.

The works presented in [Júnior et al. 2012, Bertino et al. 2005, Laborie et al. 2011] cover both temporal and spatial dimensions, where the validation of a document is performed by consistency checking over a set of constraints. In [Júnior et al. 2012], the authors briefly discuss the spatial validation, and present it as performed over the document initial positioning. In [Bertino et al. 2005, Laborie et al. 2011], the spatial dimension is static, since spatial constraints do not change over time. Moreover, reasoning about time and space is performed as two separated problems.

Finally, the approach we propose in this work and the one presented in [Elias et

al. 2006] provide a truly spatio-temporal validation, given that spatial constraints may change over time. Although [King et al. 2004] supports changes in spatial constraints over time, it is not intended for providing document validation.

The works presented in this chapter act in different parts of the document life cycle (Figure 2.2), as seen in Table 3.1, regarding the three main steps: *authoring/generation*, *instantiation* and *execution*.

Great part of related work, including those related to the structural validation act at *authoring/generation*, as expected, since they propose tools or approaches for improving document authoring. Both [Bertino et al. 2005, Laborie et al. 2011] take into account the viewer context for providing adaptation for documents prior to its execution and so they also act at *instantiation*. The validation provided by such works has a preventive role. They investigate if the document is executable and if no inconsistency may arise during execution. Such approaches, however, do not take into consideration the case where a document is dynamically edited.

[King et al. 2004] is the only work that acts at *execution* since it proposes a calculation at runtime of presentation attributes, however, its focus is not document validation. Finally, the work presented in [Belouaer and Maris 2012] is not taken into account in Table 3.1 as it is not related to multimedia documents.

In previous works [dos Santos 2012, dos Santos et al. 2013a], we presented a validation approach for documents that acts at *authoring/generation*. Such an approach focuses on both the structural and behavioral validation of document. First it performs the structural validation and, given that a document is structurally consistent, the behavioral one. The same approach is presented in this thesis [dos Santos et al. 2012a, dos Santos et al. 2013b, dos Santos et al. 2015a, dos Santos et al. 2015b] where we refine model \mathcal{SHM} and the rewrite theory RWT that will be presented in Chapter 4, providing validation in both time and space.

Besides the aforementioned approach, we also have experiments about using the validation approach proposed here for validating documents at *execution* as will be presented in Chapter 5. The idea is to provide validation at every step of a document life cycle.

In this work, we focus on the behavioral validation of multimedia documents. As will be presented in the following chapters, this work extends our previous validation approach by providing two validation techniques and proposing its use across different steps of a document life cycle.

Chapter 4

Proposed Validation Approach

This work follows a formal approach for the validation of multimedia documents. Thus, a document to be validated is described into a formal representation of its behavior. Such representation has to be as simple as possible (easing the document representation), yet expressive (so that it is possible to represent the document behavior).

4.1 Simple Hypermedia Model

The validation approach here proposed is based on a general and abstract model, called Simple Hypermedia Model (\mathcal{SHM}), for representing documents. Model \mathcal{SHM} is designed to address the following requirements:

- (i) deal with different authoring paradigms, such as event-based, constraint-based, etc.;
- (ii) allow describing both the temporal and spatial layout of a presentation;
- (iii) allow describing spatial attributes in function of time;
- (iv) be able to perform document validation at different steps of the document life cycle.

Model \mathcal{SHM} was designed based on the Nested Context Model (NCM) [Soares and Rodrigues 2005], the conceptual model of NCL. It is simpler than NCM since it aims at representing only the content and spatio-temporal layout of multimedia documents and not their structure and some presentation parameters (such as sound level, transparency and transitions). Thus, several NCM entities are not present in \mathcal{SHM} , such as the ones for defining presentation parameters (descriptors, transitions). In addition, \mathcal{SHM} does

not define *composite* nodes for improving document logical structure (context nodes) or content control (switch nodes).

\mathcal{SHM} represents the document content by its smallest pieces of information, which are called fragments.

Definition 1 (Fragment). A document fragment, or fragment for short, represents a subpart of a document. It may represent (i) a document composition as a whole, (ii) a subpart of a media item in time or space, or (iii) a document variable.

A subpart of a media item in time represents a subinterval of the media item presentation interval, possibly the whole media item presentation interval. A subpart of a media item in space represents a slice of a media item spatial region, possibly the whole media item region. \triangle

Both the temporal and spatial layout of a document in \mathcal{SHM} are described through relations among fragments. Similar to NCM, \mathcal{SHM} relations can also be defined using causal relations. \mathcal{SHM} temporal and spatial relations are described in Sections 4.1.1 and 4.1.2, respectively.

In previous works [dos Santos 2012, dos Santos et al. 2013a], we proposed a validation approach based on a rewrite theory we call here *RWT*. *RWT* is based on events, such that the temporal layout of a document is specified through causal relations among event occurrences in document fragments.

In this work [dos Santos et al. 2012a, dos Santos et al. 2013b, dos Santos et al. 2015a], *RWT* was refined in order to improve its efficiency and diminish the state explosion problem. Among such improvements we highlight (i) the redefinition of a document temporal progression, calculating a delta for which the document can be elapsed without compromising the occurrence of events; and (ii) enabling the user to restrict the number of interactions to occur over a fragment and using a predefined delay among interactions. Moreover, we also refined *RWT* [dos Santos et al. 2015b] enabling it to represent the spatial layout of a document and defining relations for changing the position of fragments either instantaneously or incrementally over a time interval. A more detailed description of *RWT* refinements is presented in Appendix D.

The spatial layout in *RWT* is specified by absolute positions of fragments on the exhibition device. Given that the complete description of a document layout in *RWT* must be specified for each instant of the document execution, we say that this approach works over *the whole* layout description.

Targeting on allowing the *partial* description of a presentation layout, in either temporal and spatial axis, and inspired by related work (See Chapter 3), this work presents an extension of the \mathcal{SHM} model. The \mathcal{SHM} 's abstraction level was increased by including in the model the possibility of defining constraints. The idea is to be able to describe the presentation layout either through causal relations and/or constraints. Moreover, such an abstract representation allows us to combine the representation of a given document with constraints from different steps in the document life cycle, thus being able to perform the validation at those different steps.

This section describes the modeling of multimedia documents for providing document validation. This is performed through model \mathcal{SHM} . Section 4.1.1 presents the modeling of temporal aspects of a document, including its temporal relations. Section 4.1.2 presents the modeling of spatial relations between media items. Section 4.1.3 discusses the representation of variables in \mathcal{SHM} and their use together with spatial and temporal relations. Finally, Section 4.2 describes the general architecture for the validation approach proposed in this work.

4.1.1 Temporal Axis

\mathcal{SHM} allows representing relations based on events and on intervals. An event occurrence represents a viewer interaction, a change in the state of a media fragment presentation or a change in the value of media fragment attribute.

\mathcal{SHM} assigns to each component a state machine representing its *presentation*, *selection* and *attribution* states. The presentation state machine is associated to media items, media item fragments and compositions, representing their presentation during execution. The presentation state of a composition is dependent on the presentation state of its inner elements. The selection state machine is associated to visual media items (video and image) and its fragments. It represents viewer selection over such elements. The attribution state machine is associated to document variables and represents changes in their value. Figure 4.1 presents the state machine used by \mathcal{SHM} , which is based on NCM.

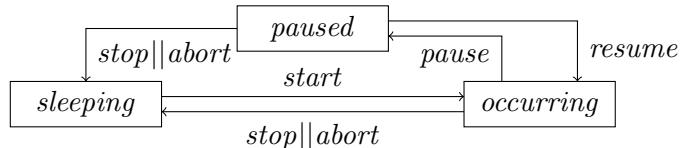


Figure 4.1: State changes

Each fragment presentation starts in the *sleeping* state. As the presentation unfolds,

it will eventually change to the *occurring* state. It remains in the *occurring* state for a given period of time, given by its duration. After its duration, it goes back to the *sleeping* state. This behavior is called the *natural end* of a fragment presentation. A similar behavior is seen for the attribution and selection state machines.

The attribution state machine also starts in the *sleeping* state and goes to the *occurring* state when the correspondent variable has its value changed. It will remain in the *occurring* state until the change is complete and then it will go back to the *sleeping* state. If the change is instantaneous, then it will spend no time at *occurring* going automatically back to *sleeping*. On the other hand, if the change is continuous, due to an animation for example, than it will remain in the *occurring* state until the change is done, or the animation finishes, for example. It is worth noticing that intermediate values can be accessed. It means that the player does not need to wait until the attribution state machine goes back to the *sleeping* state to check a variable value.

The selection state machine also starts in the *sleeping* state and goes to the *occurring* state when the viewer selects a component. It will remain in the *occurring* state until the viewer ends the selection, i.e., the button is released, and then it will go back to the *sleeping* state.

Given the behavior presented above for state machines, it is possible to project a fragment *presentation*, *attribution* or *selection* in the time axis. Figure 4.2 presents the projection of a fragment presentation in the time axis. In the figure, an interval I_f indicates when fragment f is presented along the document presentation. In the example shown in Figure 4.2, interval I_A is associated to fragment A .

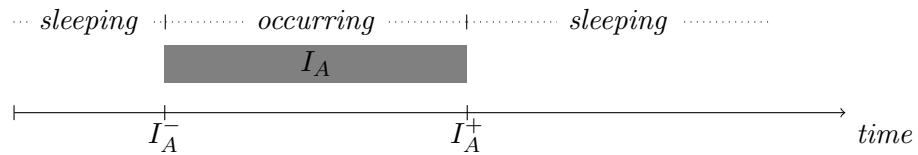


Figure 4.2: Fragment projection

Interval I_A endpoints (I_A^- and I_A^+) represent the begin and end times of the presentation of fragment A , according to its definition in a document. As it can be seen in Figure 4.2, A is in the *occurring* state inside interval I_A and in the *sleeping* state elsewhere.

During its presentation, it is possible for a fragment to be paused, thus entering the *paused* state. It occurs due to relations specified in the document, which may take into

account viewer interaction. While in the *paused* state, a fragment remains being presented and its duration counting clock is interrupted. Once *paused*, a fragment can go back to the *occurring* state according to relations specified in the document. Figure 4.3 modifies the example shown in Figure 4.2 by pausing fragment A for x seconds starting at t_1 seconds after its begin time. In the figure, interval I_{f_p} represents the amount of time fragment f remains paused.

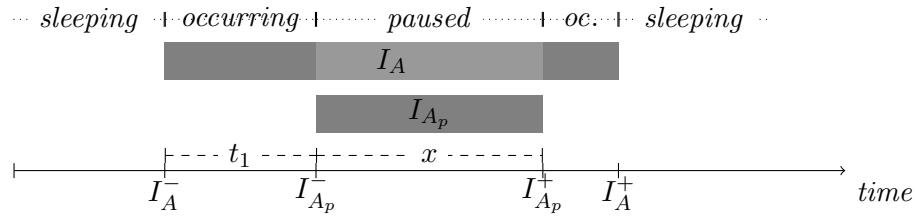


Figure 4.3: Paused fragment projection

As it can be seen in Figure 4.3, fragment A is in the *paused* state inside a given projection I_{A_p} ($I_{A_p}^- \leq t \leq I_{A_p}^+$) and either in the *occurring* or *sleeping* state elsewhere. In theory, selection and attribution state machines support pausing a viewer selection and an attribution, respectively. However, although theoretically possible, there is currently no real use for pausing a viewer selection or attribution. Therefore, both attribution and selection are represented just by a projection indicating when they are in the occurring state.

From Figure 4.3, it is possible to notice that, given the existence of a “pause interval”, the resulting duration of interval I_A will be its original value plus the duration of I_{A_p} .

It is possible for fragments to have an infinite duration. It can be achieved by two ways:

- (i) a media item does not have an inherent duration (e.g., an image) and no temporal relation defines a duration for it;
- (ii) a media item has an inherent duration, but once paused, it never returns to the occurring state.

In the second case, although an interval I_f , representing media fragment f , has an inherent duration, it will be infinite, given that I_{f_p} is infinite. A similar behavior arises when a fragment is paused once again whenever it resumes its presentation, in some kind of “pause loop”. This sequence of pause intervals, however, can be seen as just one pause interval with an infinite duration.

Given the possibility of projecting an element over the temporal axis, we are able to express relations in time either by Allen's relations [Allen 1983] between time intervals and/or causal relations among events. In Figure 4.4 we recall the set of Allen relations, where rectangles represent time intervals.

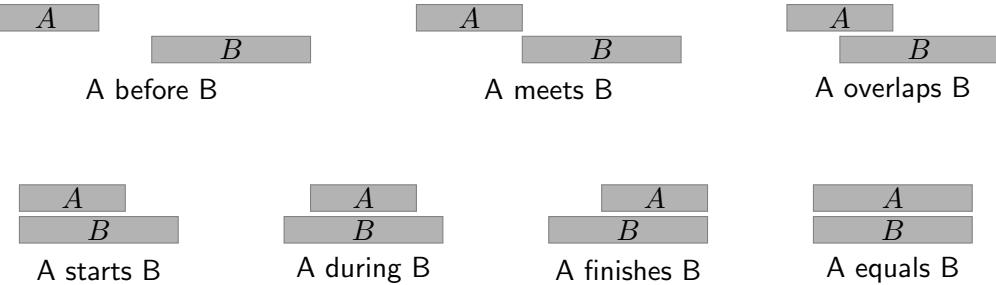


Figure 4.4: Allen's relations between time intervals

Causal relations are described among the occurrence of events. A causal relation

$$\{\epsilon_1, \dots, \epsilon_m\} \rightarrow \epsilon_n$$

expresses that event ϵ_n will occur when the first event in $\{\epsilon_1, \dots, \epsilon_m\}$ occurs. If no event in $\{\epsilon_1, \dots, \epsilon_m\}$ occurs during document execution, event ϵ_n will not occur either.

Causal relations may be applied from time to time whenever the document presentation reaches a configuration where an event occurs. During presentation, events may occur when the state of document fragments change. Such changes are produced either by the natural end of a fragment presentation, viewer interaction or as a result of the application of causal relations.

Figure 4.5 presents an example of temporal layout composed by two out of three media fragments. Media fragment m_1 is presented since the beginning of the document presentation and when it ends, either m_2 or m_3 will be presented. The choice is done taking into account the number of times viewer interactions are performed over m_1 (represented as \downarrow in the figure). Given that the viewer selects m_1 less than three times, m_2 is presented, otherwise, m_3 is presented.

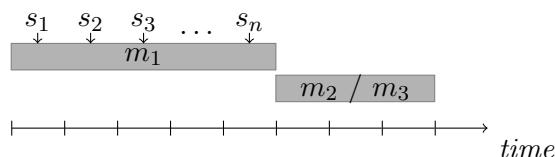


Figure 4.5: Presentation example

In this example, besides fragments, intervals are used to represent predicates. In this

case, two predicates are represented: $p_1 : s < 3$ and $p_2 : s \geq 3$, where s represents the amount of times m_1 is selected. Interval I_{p_1} , representing predicate p_1 , starts at the beginning of the document presentation and ends with the occurrence of event s_3 or with the end of the document presentation. Interval I_{p_2} , representing predicate p_2 , starts with the occurrence of event s_3 and ends with the end of the document presentation. Such behavior is represented by causal relations $\{s_3, I_D^+\} \rightarrow I_{p_1}^+$ and $s_3 \rightarrow I_{p_2}^-$, where I_D is an interval representing the whole document presentation and endpoint I_D^+ represents the document presentation end. Figure 4.6 presents the two possible temporal layouts for this example.

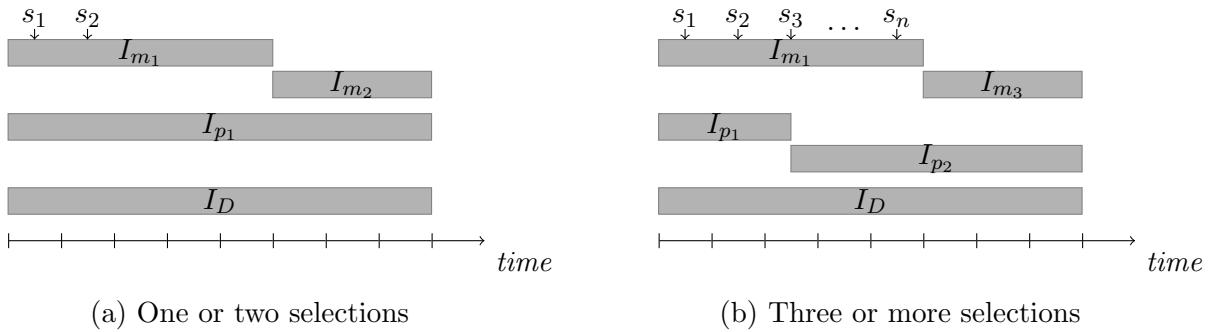


Figure 4.6: Possible temporal layouts

In Figure 4.6a, given that only two selections occurred, i.e., s_3 did not occur, then interval I_{p_1} ends with the document. Thus, interval I_{m_2} is presented. In Figure 4.6b, given that three or more selections occurred, i.e., s_3 did occur, then interval I_{p_1} ends with s_3 and interval I_{p_2} starts with s_3 and ends with the document. Thus, interval I_{m_3} is presented.

4.1.2 Spatial Axis

In the spatial axis, fragments are represented as rectangular regions. The region position and size is given by the author directly or by spatial relations. Relations in space are expressed by Randell, Cui and Cohn (RCC) relations [Randell et al. 1992]. RCC relations are presented in Figure 4.7, where rectangles represent spatial regions.

Although RCC relations enable regions to be arranged in space, they may not be enough to represent the exact spatial layout expected by an author. Suppose, for example, the four configurations presented in Figure 4.8. Each example represents a different spatial layout relating media fragments A and B .

It is worth noticing that all examples can be described by relation *over*, since in each

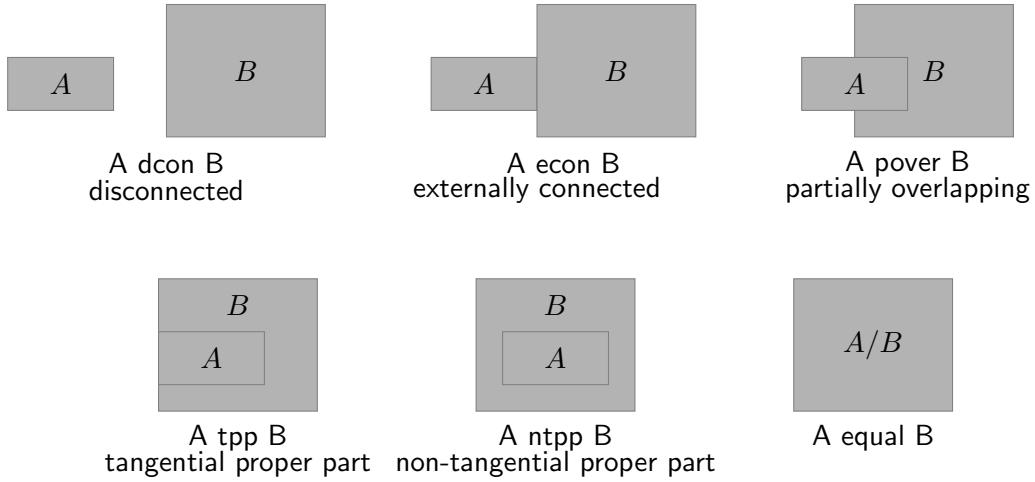
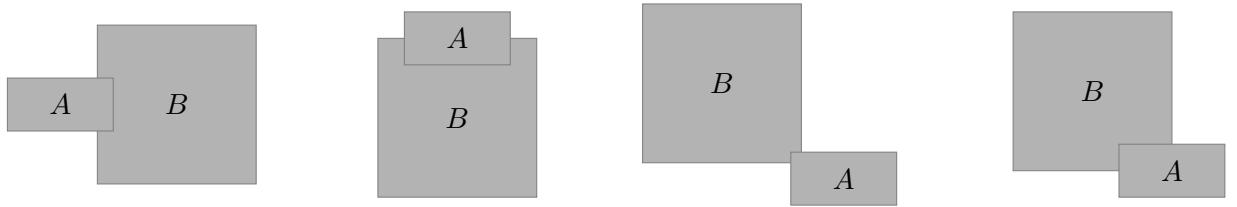


Figure 4.7: RCC spatial relations between active regions

Figure 4.8: Possible cases of $A \text{ pover } B$

one A partially overlaps B . However, they cannot be considered the same spatial layout since A and B assume different relative positions in each example.

To be able to describe the relative position between regions, we extended RCC spatial relations (except *equal*), so that relations are parameterized by the angle between region centers [dos Santos et al. 2015b]. Moreover, relations *dcon*, *pover* and *ntpp* are also parameterized by the distance between region centers. The angle and distance between two regions are calculated as presented in Figure 4.9. It presents two *disconnected* regions A and B with A being at an angle α and distance d with respect to B .

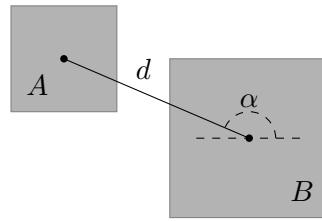


Figure 4.9: Angle and distance between regions

We represent such an example by the formula $A \text{ dcon}(\alpha, d) B$. Angle α can be described in degrees or, if such precision is not necessary, by an (intra)cardinal direction. Thus, the example of Figure 4.9 may also be represented by the formula $A \text{ dcon}(NW, d) B$.

If the direction information is given as a(n) (intra)cardinal point, the angle is given as follows.

$$\begin{aligned} E &= 0, & NE &= 45, & N &= 90, & NW &= 135, \\ W &= 180, & SW &= 225, & S &= 270, & SE &= 315 \end{aligned} \quad (4.1)$$

for each of these angles, a delta of 22.5 is considered. In case the angle is defined as a number, the delta is considered as 0. For each relation using an angle, constraints about the angle among regions are applied. Moreover, if a precise distance is not necessary, d may be omitted from the formula. Such idea is depicted in Figure 4.10.

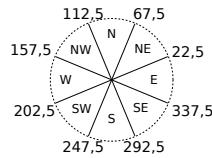


Figure 4.10: (Intra)Cardinal directions

4.1.3 Variable Handling

Multimedia documents often use variables to store information useful for building the spatio-temporal layout of a presentation. For example, in the document presented in Figure 4.6, a variable is used to store the number of interactions performed by the viewer. \mathcal{SHM} represents variables the same way it represents media fragment attributes.

As presented in Section 4.1.1, an attribution state machine holds the state of a variable and its value. Whenever the value of a variable is changed, the state of the attribution state machine associated to it goes from *sleeping* to *occurring* and back to *sleeping* state. Given that changes in the value of variables are usually instantaneous, the state machine does not remain in the *occurring* state as it happens with selection state machines.

During presentation, it is possible for variables to change their value. A common way to achieve it is to declare relations that change the value of a variable in response to an event occurrence. This change can be either *discrete*, or *incremental* over a time interval. In the latter, the relation provides, together with the new values, the duration for the change and the increment by which values have to be changed.

In \mathcal{SHM} there are two relations to define the value of a variable, relations *set* and *animate*. Both relations may be associated to time intervals, so that a variable a has

its value equals to v inside an interval I , similar to what is done in [Belouaer and Maris 2012]. Figure 4.11 depicts this idea.

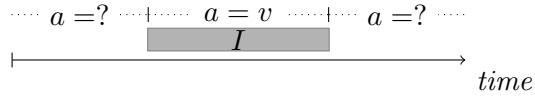


Figure 4.11: Value in relation to time

Relation *set* defines a discrete value change. Relation *animate* defines an incremental change over interval i by defining the value of a variable as a polynomial¹ of order $n > 1$ with the time $t \in [0, i.duration]$ as the polynomial's variable. Time value t is calculated in function of the global presentation time as $t = T - i.init$. Time values t are calculated according to the number of increments for the value change. It is worth noticing that each incremental change acts like a discrete one, i.e., the state of the attribution state machine associated to the given variable goes from *sleeping* to *occurring* and back to *sleeping* state, as discussed before.

In time, intervals represent predicates over the value of variables. An example is presented in Figure 4.6 where two intervals were used to represent that the viewer interacted with the presentation less than three times (p_1) or three or more times (p_2).

In the spatial dimension, variables are used to represent positioning attributes. By using relations *set* and *animate* presented above, \mathcal{SHM} models changes in the spatial layout of a document in relation to time. Moreover, the use of a polynomial for defining the value of a positioning attribute is interesting since it brings the possibility to represent complex animations such as the ones provided by SMIL/SVG animation [W3C 2008b, W3C 2011] or Web Animations [W3C 2014]. In such an approach, animation paths, keyframes and so on, are represented by polynomial interpolation.

4.2 Validation Approach Architecture

As it could be seen previously in this chapter, from model \mathcal{SHM} it is possible not only to reason about document fragments in terms of their state along the presentation, but also about their projections in the temporal axis (as intervals). \mathcal{SHM} allows the description of relations both as causal relations over event occurrences or as constraints over intervals.

¹Polynomials were chosen for relation *animate* so that changes in the value of a given variable due to complex animations (for example moving an item along a path) may be represented (approximated) by polynomial interpolation.

For validation purposes, therefore, we may get from \mathcal{SHM} two representations of a document. One in terms of its state along the presentation, and another in terms of its intervals. We consider that the document state comprises the state of all media fragments and variables declared in a document. Moreover the “state of a media fragment” is considered to be not only its presentation state in time, but also its position in space. Figure 4.12 depicts the general architecture of the proposed validation approach.

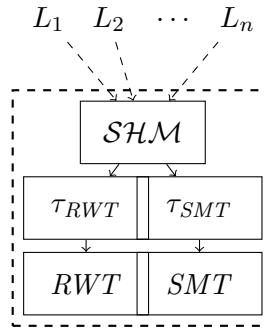


Figure 4.12: Validation approach architecture

Figure 4.12 highlights the role of the \mathcal{SHM} model, as an abstract representation for the purpose of validating documents described with different authoring paradigms (documents specified in languages L_1 to L_n), as discussed in the beginning of this section.

In order to take advantage of the two possible representations for a document, the validation approach we propose in this work is composed of two subparts, as seen in Figure 4.12.

The first part produces from the \mathcal{SHM} model, through transformation τ_{RWT} , a representation of a document in terms of its states. The resulting representation is described into a rewriting theory we call *RWT* [dos Santos et al. 2013b, dos Santos et al. 2015a, dos Santos et al. 2015b]. In theory *RWT*, document fragments are represented as state machines and causal relations are modeled as equations that change their state along document presentation. Moreover, temporal constraints are modeled as linear temporal logic formulas and validation is performed through model-checking.

The second part produces from the \mathcal{SHM} model, through transformation τ_{SMT} , a representation of a document in terms of intervals. The resulting representation is described as a set of SMT formulas we call *SMT*. In *SMT*, document fragments are represented as projections in the temporal and spatial axis along document presentation. Both causal relations and constraints are represented as SMT assertions and validation is performed through SMT solving [Moura and Bjørner 2011].

The \mathcal{SHM} model is built to cover both approaches *RWT* and *SMT*. *RWT* is more suited for expressing documents where events and different occurrences of media fragments are an important feature. On the other hand, *SMT* is more suited for expressing documents where numeric dependencies among media fragment intervals represent an important feature.

Besides its power of expressiveness, the choice for a given approach may depend on its efficiency and the type of validation to be performed. Thus, according to where in the presentation life cycle validation is performed, one approach or the other is chosen. A discussion about the use of each approach is presented in Section 5.6.1.

The following sections present both *RWT* and *SMT* in more details, together with translations τ_{RWT} and τ_{SMT} , respectively.

4.3 Validation Based on Document State

As presented in Section 4.1, from the \mathcal{SHM} model, we can get a representation of a document in terms of its states [dos Santos et al. 2013b,dos Santos et al. 2015a,dos Santos et al. 2015b]. It is done by describing a given document as a rewriting theory [Meseguer 2012]. We call such approach *RWT*. In this section, we present how the representation d_{RWT} is obtained from document d through transformation $\tau_{RWT}(d)$.

The formalization presented in this section relies on the Rewriting Logic calculus and model checker evaluation. On Appendix B, we present an overview of the necessary Rewriting Logic elements to explain our approach (its calculus, rewrite theories modulo associativity, commutativity and identity axioms, and reflective metatheories) and the Maude language, an implementation of Rewriting Logic.

RWT describes the general behavior of multimedia documents as a rewrite theory $\mathcal{R}_{RWT} = (\Sigma, E, R)$ where Σ and E denote the document structure, and R denotes the (possibly) non-deterministic behavior of a document. Intuitively, the deterministic behavior of the application of causal relations is captured by E and the non-determinism induced by viewer interaction and time is captured by R . For a given document d , a theory d_{RWT} extends \mathcal{R}_{RWT} providing document specific information, such as the fragments and causal relations declared by document d . The following section presents theory \mathcal{R}_{RWT} in details.

4.3.1 Rewrite Theory \mathcal{R}_{RWT}

In RWT , document fragments, i.e., compositions, media items, media fragments and variables, are represented by *information units*, or *units* for short. Document relations (*rel* for short) define a temporal order for the presentation of *units*. The representation of a given document in \mathcal{R}_{RWT} is presented in Definition 2.

Definition 2 (\mathcal{R}_{RWT}). The representation of a document through its state is specified in Rewriting Logic as the theory

$$\mathcal{R}_{RWT} = (\Sigma, E, R)$$

where Σ represents the signature of \mathcal{R}_{RWT} , E its equational part and R its rules. Σ declares sorts *MachineType*, *MachineState*, *ClockVal* and sort *UnitAtt*, subsort of *Component*, whose elements are constructed with operators **state**, **occur**, **clock** and **value**. Terms of *Conf*, representing the document state as a whole, are formalized as a set of components, using the set constructor *Set{Component}*, and a number representing a global clock. The set constructor in Maude allows rewriting modulo associative, commutative, idempotence and identity (constant **empty**) as discussed in Section B.4. Sorts *MachineType*, *MachineState*, *UnitAtt*, *Component* and *Conf* are declared in Maude as follows:

```

1   sorts MachineType MachineState ClockVal UnitAtt Component .
2   subsort UnitAtt < Component .
3   subsort InfNat < ClockVal .
4
5   ops pre sel att : -> MachineType [ctor] .
6   ops sleeping occurring paused : -> MachineState [ctor] .
7   op none : -> ClockVal [ctor] .
8
9   op value : Qid InfNat -> UnitAtt [ctor] .
10  op value : Qid String -> UnitAtt [ctor] .
11  op state : Qid MachineType MachineState -> UnitAtt [ctor] .
12  op occur : Qid MachineType InfNat -> UnitAtt [ctor] .
13  op clock : Qid MachineType ClockVal -> UnitAtt [ctor] .
14
15  op <-|-> : InfNat Set{Component} -> Conf [ctor] .
16
17  op dur : Qid MachineType -> InfNat .

```

Σ also declares sorts *EventTransition* and *Action*, both subsort of *Component*. Elements of sort *EventTransition* are constructed with operators **init**, **end**, **hang**, **halt** and **return**. Elements of sort *Action* are constructed with operators **start**, **stop**, **abort**, **pause** and **resume**. Both sorts are declared in Maude as follows:

```

1   sorts EventTransition Action .
2   subsort EventTransition Action < Component .
3
4   ops init end hang halt return : Qid MachineType -> EventTransition [ctor] .
5   op key : Qid String -> EventTransition [ctor] .
6   ops start stop abort pause resume : Qid MachineType -> Action [ctor] .

```

E gives the semantics of elements of sorts *UnitAtt* and *Action* as a set of equations. Equations **natural**, **start**, **stop**, **abort**, **pause** and **resume** are declared as follows:

```

1 vars N M : InfNat . var ID : Qid . var MT : MachineType .
2
3 eq [natural] : state(ID,MT,occurring) occur(ID,MT,N) clock(ID,MT,0) = state(ID,MT,sleeping) occur(
4   ID,MT,s N) clock(ID,MT,none) end(ID,MT) .
5
6 ceq [abort] : abort(ID,MT) state(ID,MT,occurring) clock(ID,MT,M) = state(ID,MT,sleeping) clock(ID,
7   MT,none) hang(ID,MT) if M > 0 .
8 ceq [abort] : abort(ID,MT) state(ID,MT,pause) clock(ID,MT,M) = state(ID,MT,sleeping) clock(ID,MT,
9   none) hang(ID,MT) if M > 0 .
10 eq [abort] : abort(ID,MT) state(ID,MT,sleeping) = state(ID,MT,sleeping) .
11
12 eq [pause] : pause(ID,MT) state(ID,MT,occurring) = state(ID,MT,pause) halt(ID,MT) .
13 eq [pause] : pause(ID,MT) state(ID,MT,pause) = state(ID,MT,pause) .
14 eq [pause] : pause(ID,MT) state(ID,MT,sleeping) = state(ID,MT,sleeping) .
15
16 eq [resume] : resume(ID,MT) state(ID,MT,pause) = state(ID,MT,occurring) return(ID,MT) .
17 eq [resume] : resume(ID,MT) state(ID,MT,occurring) = state(ID,MT,occurring) .
18 eq [resume] : resume(ID,MT) state(ID,MT,sleeping) = state(ID,MT,sleeping) .
19
20 ceq [start] : start(ID,MT) state(ID,MT,sleeping) clock(ID,MT,none) = state(ID,MT,occurring) clock(ID
21   ,MT,dur(ID,MT)) init(ID,MT) .
22 ceq [start] : start(ID,MT) state(ID,MT,occurring) = state(ID,MT,occurring) .
23 ceq [start] : start(ID,MT) state(ID,MT,pause) = state(ID,MT,pause) .
24
25 ceq [stop] : stop(ID,MT) state(ID,MT,occurring) occur(ID,MT,N) clock(ID,MT,M) = state(ID,MT,
26   sleeping) occur(ID,MT,s N) clock(ID,MT,none) end(ID,MT) if M > 0 .
27 ceq [stop] : stop(ID,MT) state(ID,MT,pause) occur(ID,MT,N) clock(ID,MT,M) = state(ID,MT,sleeping)
28   occur(ID,MT,s N) clock(ID,MT,none) end(ID,MT) if M > 0 .
29 eq [stop] : stop(ID,MT) state(ID,MT,sleeping) = state(ID,MT,sleeping) .

```

R declares rule **step** as follows:

```

1 var SC : Set{Component} .
2
3 op dt : Set{Component} -> InfNat .
4 op elapse : Set{Component} InfNat -> Set{Component} .
5 op discard : Set{Component} -> Set{Component} .
6 ops check active : Set{Component} -> Bool .
7
8 crl [step] : < N | SC > => < N + dt(SC) | elapse(discard(SC), dt(SC)) > if check(SC) and active(SC)
9   and min(N,max) == N .

```

it is worth noticing that the document execution is bound to a maximum duration specified (by the user) in constant *max*. Document dependent information is provided by the constructors *max*, *ini*, *doc* declared as follows:

```

1 op max : -> Nat .
2 op ini : -> Action .
3 op doc : -> Set{Component} .
4
5 op run : -> Conf .
6 eq run = < 0 | ini, doc > .

```

constructor *run* is a macro for constructing the document's initial state. \triangle

Lemma 1. *The set of equations E is Church-Rosser and terminating.*

Proof. (Sketch.) By construction, through document transformation, there are no two elements constructed with operators **state**, **occur**, **clock** or **value**, having the same *id*

and *machine type* (when applicable). Moreover, links are built in a way that it is not possible to have two elements of sort *Action* with the same *id*.

Hence, equations in E have disjoint left-hand sides and therefore no critical pairs to be joined arise. \square

Lemma 2. *The set of rules R is coherent with respect to E .*

Proof. (Sketch.) R defines conditional rule **step**, whose condition includes function **check**. Such a function verifies if function **elapse** was already applied (i.e., the document configuration does not contain **elapse** at the top), if the set of components has no element of sort *Action* and no element **clock** with value 0.

Hence, the rule **step** can only be applied after equations in E have been applied. Therefore R is coherent with respect to E . \square

Theorem 1. *The theory \mathcal{R}_{RWT} is preregular, Church-Rosser, sort-decreasing and coherent*

Proof. (Sketch.) \mathcal{R}_{RWT} is Church-Rosser and coherent by Lemmata 1 and 2. The proof of preregularity and sort-decreaseness is done by structural induction over terms in $T_{\Sigma,E}$ by verifying that each term has a least sort. \square

The proof for Theorem 1 was automated by the application of the *Church-Rosser* checker tool provided by the Maude Formal Environment (MFE) [Durán et al. 2011]. The use of MFE together with the validation approach presented in this work is a future work. It is worth noticing that the proof is related to the part of the specification that is document independent.

Theory \mathcal{R}_{RWT} represents the behavior of a document, with Σ providing constructors for representing the state of a document, which is comprised by the states of *units*; E represents the behavior of parts of the document in terms of causal relations, and R represents the behavior of the document presentation as a whole.

Every *unit*, representing a document fragment, is associated with event state machine configurations that represent state information for a given *unit*. Every information about a *unit* is represented in Σ as an element of sort *UnitAtt*. It declares operators **state**, **occur**, **clock** and **value**, to represent respectively the *unit*'s state, occurrences counter, countdown clock (to register the amount of time it will remain in the *occurring* state) and value (in case it represents a document variable). Thus it describes the state machines for representing an element *presentation*, *selection* and *attribution* states. Such state

machines follow the one presented in Figure 4.1. We repeat that figure here to improve this section readability.

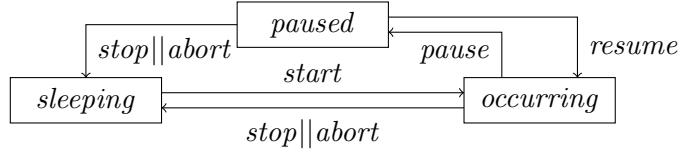


Figure 4.13: State changes

It is worth noticing that every attribute is parameterized by the *unit* identification and the type of the state machine configuration (i.e., presentation, selection or attribution), except for **value**, which is used only for variables. In this case, the (attribution) type is implicit.

A given document declares relations (*rel* for short) to define a temporal order for the execution of its *units*. In *S&HM*, relations are defined through causal relations or constraints. In *RWT*, we represent only causal relations for defining the temporal order for the execution of units. Constraints, as will be discussed later, are used for describing the expected behavior of a document. A *rel* has a condition to be satisfied before the relation is applied. This condition is triggered by event transitions and when the *rel* condition is satisfied, a set of actions is executed. Event transitions and *rel* actions are represented respectively as elements from sorts *EventTransition* and *Action*.

The semantics of *units* and *actions* are defined in *E* as a set of equations. Equation **natural** represents the natural end of *units*, while equations **start**, **stop**, **abort**, **pause** and **resume** declare the semantics of its homonymous state changes as presented in Figure 4.1.

The occurrence of such changes in the state machine of a given unit produces effects, besides in its state, in its countdown clock and occurrences counter. The semantics, therefore, of such state changes are defined as follows.

- whenever a *start* action is fired on a state machine its state changes to *occurring* and its countdown clock is set to the unit duration;
- whenever a *pause* action is fired on a state machine its state changes to *paused* and its countdown clock is paused;
- whenever a *resume* action is fired on a state machine its state changes to *occurring* and its countdown clock is resumed;

- whenever a *stop* action is fired on a state machine its state changes to *sleeping*, its countdown clock is set to zero and its occurrences counter is incremented;
- whenever an *abort* action is fired on a state machine its state changes to *sleeping* and its countdown clock is set to zero;

Rels are represented as equations in E , whose condition is represented in terms of *event transitions* and *unit attributes* in the left-hand side of the equation, while the actions to be performed are declared in the right-hand side of the equation. The *rel* example below starts the presentation of *unit* u_2 and changes the value of *unit* u_3 to “bog”, when the presentation of *unit* u_1 reaches its end and the value of *unit* u_3 is equal to “foo”.

```
1  eq [rel] : end('u1, pre) value('u3, "foo") = start('u2, pre) start('u3, att) value('u3, "bog") .
```

Listing 4.1: *Rel* example

It is worth highlighting that (i) whenever a *rel* is applied, it consumes an event transition, (ii) a *rel* action will be applied only if its target *unit* is in a state where it can be applied (e.g., a *start* action can only be applied to a *unit* in the *sleeping* state), and (iii) the application of a *rel* may trigger the application of another *rel*.

The behavior of the document as a whole in \mathcal{R}_{RWT} is given by R with rules. Rule *step* performs a temporal progression in the document presentation. User interaction, on the other hand, may occur at any moment during the presentation of a *unit*. To simulate user interaction, for each interaction enabled *unit* u auxiliary *units* u_{aux} represent delays from the beginning of the presentation of u to the moment the user interacted with it. The number of auxiliary *units* u_{aux} to be created is defined by the user. Rules for user interaction, therefore, will trigger the selection of u whenever a u_{aux} ends its presentation.

Theory \mathcal{R}_{RWT} is represented as a Maude module **RRWT**, which represents the general behavior for multimedia documents. The complete description of module **RRWT** is presented in Appendix E. For each particular document, we declare another module that extends theory \mathcal{R}_{RWT} defining the document initial configuration in terms of its *units*, its *rels*, interaction rules, an initial action and information about *unit*'s duration. The document initial configuration and *unit*'s duration are declared by equations that use operators **doc** and **dur**, respectively. An initial action for starting the document execution is declared by an equation that uses operator **ini**. Document *rels* are specified as equations as seen in Listing 4.1.

Listing 4.2 presents parts of a Maude module representing a document. The complete example is presented in Appendix F.

```

1 mod 'JOAO' is
2   including 'BOOL'.
3   including 'RNCL'.
4   sorts none .
5   none
6   none
7   none
8   eq 'doc.Set'{Component} = '_,-[_,-[ 'state['animation.Qid,'pre.MachineType,'sleeping.
MachineState],'_,-[_,[occur['animation.Qid,'pre.MachineType,'0.Zero], 'clock['animation.Qid,'pre.
MachineType,'none.ClockVal]]], ... ] [none] .
9
10  eq 'dur['animation.Qid,'pre.MachineType] = 's_-^71['0.Zero] [none] .
11  ...
12
13  eq 'init['body.Qid,'pre.MachineType] = 'start['animation.Qid,'pre.MachineType] [none] .
14  ...
15
16  rl '<_ | ->[T:InfNat,_,-[_,[S:Set{Component}, 'end['icon+selec1.Qid,'pre.MachineType]]] =>
17  '<_ | ->[T:InfNat,'discard[_,-[_,-[_,[S:Set{Component}, 'key['icon.Qid,'"RED".String]], ,
start['icon.Qid,'sel.MachineType]]]] [label('icon+selec:RED')] .
18
19  rl '<_ | ->[T:InfNat,_,-[_,[S:Set{Component}, 'end['icon+selec2.Qid,'pre.MachineType]]] =>
20  '<_ | ->[T:InfNat,'discard[_,-[_,-[_,[S:Set{Component}, 'key['icon.Qid,'"RED".String]], ,
start['icon.Qid,'sel.MachineType]]]] [label('icon+selec:RED')] .
21 endm

```

Listing 4.2: Document example

Let us take as example element '`animation`' in line 8. It declares a unit for representing a video media item. Such unit is built with constructions `state`, `occur` and `clock`².

Equation `dur` in line 10 represents the duration of such a video and the equation in line 13 starts '`animation`'s presentation when the document starts its presentation. Both rules in lines 16 and 19 represent user interaction with *unit* '`icon`'³.

4.3.2 Modeling Spatio-temporal Relations

Variables are used (but are not limited) to store the position of units in *RWT*. A given variable represents the value of a given positioning attribute - *left*, *top*, *width* or *height*. To be able to relate a given variable to a unit, *RWT* provides functions *left*, *top*, *width* and *height*, defined as follows:

$$\text{left}, \text{top}, \text{width}, \text{height} : \text{MedId} \rightarrow \text{VarId} \quad (4.2)$$

where *MedId* and *VarId* are sets of identifiers for media items and variables in *SHM*, respectively. Thus, we can evaluate the value of the *left* attribute of unit *A*, by evaluating the value of the unit whose id is *left(A)*.

²Constructors `state`, `occur` and `clock` are metarepresented by operations '`state[...]`', '`occur[...]`' and '`clock[...]`', respectively. Moreover, operator '`_,-[_,...]`' is the metarepresentation for the set formation constructor '`-,-`'.

³Operator '`<_ | ->[...]`' is the metarepresentation for a configuration in *R_{RWT}*.

Causal relations are used in *RWT* for changing the position and/or size of a unit, by changing the value of its positioning attributes. This change can be either *discrete*, or *incremental* over a time interval as seen in Section 4.1.3.

In case the change is incremental, the relation provides, together with the new values, the duration for the change and the increment by which values have to be changed. Such relations are modeled in *RWT* by adding an auxiliary unit to represent the delay between two incremental changes. For example, suppose the following causal relation.

$$[r] A.begin \rightarrow \left\{ \begin{array}{l} B.left := 400 \\ \text{during : } 4s \text{ by : } 10px \end{array} \right\} \text{ if } B.left == 0$$

It states that whenever the document reaches a configuration where media item *A* begins its presentation and *B*'s left position is equal to 0, *B*'s positioning attribute *left* changes its value by an increment of 10 *pixels* for 4 *seconds*. The *left* value changes 400 *pixels*, with an increment of 10 *pixels*, thus 40 incremental changes are produced over 4 seconds. Therefore each change occurs at each 0.1 seconds.

For representing such an example, a unit *C* will be created to represent the delay between changes with a duration of 0.1 seconds. It will start its presentation at the first time the value is changed and will be restarted as each increment is performed. The following rewrite rules represent such behavior.

$$[r_{init}] A.pre.begin \rightarrow \left\{ \begin{array}{l} left(B).att.start, \\ left(B).value += 10, \\ C.pre.start \end{array} \right\} \text{ if } left(B).value == 0 \quad (4.3)$$

$$[r_inc] C.pre.end \rightarrow \left\{ \begin{array}{l} left(B).att.start, \\ left(B).value += 10, \\ C.pre.start \end{array} \right\} \text{ if } C.pre.occur < 40 \quad (4.4)$$

where operation $+=$ represents an increment operation over the value of variables.

4.3.3 *RWT* Document Validation

Given a document d , according to Section 4.3.1, d_{RWT} is the metarepresentation of a rewriting logic theory representing d , where d_{RWT} includes the rewriting theory \mathcal{R}_{RWT} . Metaterm d_{RWT} represents a Maude module and declares the initial state of document d with an equation for operator *doc* and the initial action for starting the document execution with an equation for operator *ini*. The initial term t_0 , that represents the initial state of d is built as follows

$$t_0 = \langle \text{ 0 } | \text{ doc, ini } \rangle$$

Maude allows us to either *rewrite* or *search* term t_o . The *rewrite* command chooses one of the possible traces of a term rewrite. However, opposite to equations, rules may be non-deterministic, thus producing concurrent rewrites. In Maude we can search in the “space state” of a term rewrite using command *search*.

Therefore, we can validate the behavior of a document in three possible ways. (i) Using the *rewrite* command to simulate document d , presenting one possible presentation for it. (ii) Using the *search* command to present all possible presentations of d or to present traces that leads the document state to the pattern defined in the search command. (iii) Using the *modelCheck* command to verify the existence of a given behavior in every possible execution of d .

Theory d_{RWT} induces a transition system $\mathcal{S}_{RWT} = (S, \rightarrow)$, where the state in S are terms in the equational theory (Σ, E) of \mathcal{R}_{RWT} and the transition relation \rightarrow is captured by the rules in R of \mathcal{R}_{RWT} . Therefore, a given document presentation is captured by computations in \mathcal{S}_{RWT} and therefore as rewrites in \mathcal{R}_{RWT} . A one step rewrite in \mathcal{R}_{RWT} corresponds exactly to a transition in \mathcal{S}_{RWT} .

As seen before, \mathcal{SHM} declares relations either as causal relations or constraints. Causal relations in *RWT* are used for defining the temporal order of units. Constraint relations, on the other hand, are used for describing the expected behavior of the document.

\mathcal{SHM} temporal and spatial relations (see Sections 4.1.1 and 4.1.2) are formalized as Linear Temporal Logic (LTL) [Pnueli 1977] formulas. An LTL formula φ is defined as follows, where X, F, G, U, W and R are called temporal operators.

$$\varphi ::= \top \mid \perp \mid p \mid \neg(\varphi) \mid (\varphi \wedge \varphi) \mid (\varphi \vee \varphi) \mid (\varphi \rightarrow \varphi) \mid (4.5)$$

$$(X\varphi) \mid (F\varphi) \mid (G\varphi) \mid (\varphi U\varphi) \mid (\varphi W\varphi) \mid (\varphi R\varphi) \quad (4.6)$$

$X\varphi$ (*next*) states that a formula φ must be valid for the following state. $F\varphi$ (*future*) states that a formula φ must be valid for some future state. $G\varphi$ (*global*) states that a formula φ must be valid for all states in a path. $\varphi_1 U \varphi_2$ (*until*) states that a formula φ_1 must be valid until a formula φ_2 becomes valid. $\varphi_1 W \varphi_2$ (*weak until*) states that a formula φ_1 must be valid until a formula φ_2 becomes valid or φ_1 must be valid for all states in the path. $\varphi_1 R \varphi_2$ (*release*) states that a formula φ_1 must be valid until a formula φ_2 becomes valid and both φ_1 and φ_2 must be valid at the same time for some state.

It is worth noticing that formulas in the temporal axis describe the evolution of (part of) the document state through several states. For example, formula *A meets B* is described by the following LTL formula.

$$i_1 \text{ meets } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.occuring} \wedge i_2.\text{pre.sleeping}) \wedge \\ X(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occuring}) \end{array} \right) \quad (4.7)$$

The other Allen relations [Allen 1983] are described by the following LTL formulas.

$$i_1 \text{ before } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.occuring} \wedge i_2.\text{pre.sleeping}) \wedge \\ X(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.sleeping}) \wedge \\ F(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occuring}) \end{array} \right) \quad (4.8)$$

$$i_1 \text{ overlaps } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.occuring} \wedge i_2.\text{pre.sleeping}) \wedge \\ X(i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring}) \wedge \\ F(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occuring}) \end{array} \right) \quad (4.9)$$

$$i_1 \text{ starts } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.sleeping} \wedge i_2.\text{pre.sleeping}) \wedge \\ X(i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring}) \wedge \\ F(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occuring}) \end{array} \right) \quad (4.10)$$

$$i_1 \text{ during } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occurring}) \wedge \\ X(i_1.\text{pre.occurring} \wedge i_2.\text{pre.occurring}) \wedge \\ F(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occurring}) \end{array} \right) \quad (4.11)$$

$$i_1 \text{ finishes } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occurring}) \wedge \\ F \left(\begin{array}{l} (i_1.\text{pre.occurring} \wedge i_2.\text{pre.occurring}) \wedge \\ X(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.sleeping}) \end{array} \right) \end{array} \right) \quad (4.12)$$

$$i_1 \text{ equals } i_2 = F \left(\begin{array}{l} (i_1.\text{pre.sleeping} \wedge i_2.\text{pre.sleeping}) \wedge \\ X(i_1.\text{pre.occurring} \wedge i_2.\text{pre.occurring}) \wedge \\ F \left(\begin{array}{l} (i_1.\text{pre.occurring} \wedge i_2.\text{pre.occurring}) \wedge \\ X(i_1.\text{pre.sleeping} \wedge i_2.\text{pre.sleeping}) \end{array} \right) \wedge \\ \neg F \left(\begin{array}{l} (i_1.\text{pre.occurring} \wedge i_2.\text{pre.sleeping}) \vee \\ (i_1.\text{pre.sleeping} \wedge i_2.\text{pre.occurring}) \end{array} \right) \end{array} \right) \quad (4.13)$$

On the other hand, formulas in the spatial axis consider the values of positioning attributes inside a given state. They are combined with a temporal operator for representing a spatio-temporal layout. As a usage example of a spatial formula, suppose we want to verify if, at some time, A and B will overlap in space. Thus we write the following formula

$$F(A \text{ pover } B) \quad (4.14)$$

where we combine spatial relation *pover* with temporal operator F (future).

The RCC relations [Randell et al. 1992] are described by the following LTL formulas, where $l_j = \text{left}(i_j).\text{value}$, $t_j = \text{top}(i_j).\text{value}$, $w_j = \text{width}(i_j).\text{value}$ and $h_j = \text{height}(i_j).\text{value}$.

$$i_1 \text{ dcon } i_2 = \left(\begin{array}{l} i_1.\text{pre.occurring} \wedge i_2.\text{pre.occurring} \wedge \\ \left(\begin{array}{l} (l_1 > (l_2 + w_2)) \vee ((l_1 + w_1) < l_2) \vee \\ (t_1 > (t_2 + h_2)) \vee ((t_1 + h_1) < t_2) \end{array} \right) \end{array} \right) \quad (4.15)$$

$$i_1 \text{ econ } i_2 = \begin{cases} i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring} \wedge \\ \left(\begin{array}{l} (((l_1 = (l_2 + w_2)) \vee ((l_1 + w_1) = l_2)) \wedge \\ (t_1 \leq (t_2 + h_2)) \wedge (t_2 \leq (t_1 + h_1)) \end{array} \right) \vee \\ \left(\begin{array}{l} (((t_1 = (t_2 + h_2)) \vee ((t_1 + h_1) = t_2)) \wedge \\ (l_1 \leq (l_2 + w_2)) \wedge (l_2 \leq (l_1 + w_1)) \end{array} \right) \end{cases} \quad (4.16)$$

$$i_1 \text{ pover } i_2 = \begin{cases} i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring} \wedge \\ \left(\begin{array}{l} ((l_1 < l_2) \wedge ((l_1 + w_1) > l_2) \wedge ((l_1 + w_1) < (l_2 + w_2)) \wedge \\ (t_1 < (t_2 + h_2)) \wedge (t_2 < (t_1 + h_1)) \end{array} \right) \vee \\ \left(\begin{array}{l} ((l_2 < l_1) \wedge ((l_2 + w_2) > l_1) \wedge ((l_2 + w_2) < (l_1 + w_1)) \wedge \\ (t_1 < (t_2 + h_2)) \wedge (t_2 < (t_1 + h_1)) \end{array} \right) \vee \\ \left(\begin{array}{l} ((t_1 < t_2) \wedge ((t_1 + h_1) > t_2) \wedge ((t_1 + h_1) < (t_2 + h_2)) \wedge \\ (l_1 < (l_2 + w_2)) \wedge (l_2 < (l_1 + w_1)) \end{array} \right) \vee \\ \left(\begin{array}{l} ((t_2 < t_1) \wedge ((t_2 + h_2) > t_1) \wedge ((t_2 + h_2) < (t_1 + h_1)) \wedge \\ (l_1 < (l_2 + w_2)) \wedge (l_2 < (l_1 + w_1)) \end{array} \right) \end{cases} \quad (4.17)$$

$$i_1 \text{ tpp } i_2 = \begin{cases} i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring} \wedge \\ \left(\begin{array}{l} (((w_1 < w_2) \wedge (h_1 \leq h_2)) \vee ((h_1 < h_2) \wedge (w_1 \leq w_2))) \wedge \\ \left(\begin{array}{l} (((l_1 = l_2) \vee ((l_1 + w_1) = (l_2 + w_2))) \wedge \\ (t_1 \geq t_2) \wedge ((t_1 + h_1) \leq (t_2 + h_2)) \end{array} \right) \vee \\ \left(\begin{array}{l} (((t_1 = t_2) \vee ((t_1 + h_1) = (t_2 + h_2))) \wedge \\ (l_1 \geq l_2) \wedge ((l_1 + w_1) \leq (l_2 + w_2)) \end{array} \right) \end{array} \right) \end{cases} \quad (4.18)$$

$$i_1 \text{ ntpp } i_2 = \begin{cases} i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring} \wedge \\ \left(\begin{array}{l} (l_1 > l_2) \wedge ((l_1 + w_1) < (l_2 + w_2)) \wedge \\ (t_1 > t_2) \wedge ((t_1 + h_1) < (t_2 + h_2)) \end{array} \right) \end{cases} \quad (4.19)$$

$$i_1 \text{ equal } i_2 = \begin{pmatrix} i_1.\text{pre.occuring} \wedge i_2.\text{pre.occuring} \wedge \\ (l_1 = l_2) \wedge (w_1 = w_2) \wedge \\ (t_1 = t_2) \wedge (h_1 = h_2) \end{pmatrix} \quad (4.20)$$

In case the formula is parameterized by an angle, function *angle* is called together with the formulas presented previously to determine the angle between region centers. Function *angle* is parameterized by the expected angle in either degrees or one (intra)cardinal direction.

4.4 Validation Based on Constraints

Section 4.1 presented the general model \mathcal{SHM} and how it represents multimedia documents. It also presented the general architecture for the validation approach proposed in this work, where from \mathcal{SHM} we get a representation of a document as a set of SMT formulas. We call such a representation *SMT*. In this section, we present how the representation d_{SMT} is obtained from document d through transformation $\tau_{SMT}(d)$.

SMT [Moura and Bjørner 2011] is a satisfaction problem where formulas combine logical connectives such as conjunction, disjunction and negation, with atomic formulas in the form of linear arithmetic inequalities. An example of SMT formula is presented as follows.

$$(v_1 \leq v_2) \wedge (v_3 \vee (v_1 > v_2))$$

As seen in the example, formulas are composed by variables, representing either boolean or arithmetic values. A solution for such a formula is an assignment, mapping variables v_i to values that make the formula *true*.

An SMT solver builds the formula representing the whole satisfaction problem from *assertions*. Each *assertion* is a formula like the one presented above, and the whole satisfaction problem is given by the conjunction of assertions. In this work, each assertion models a constraint representing either document relations or properties that must hold during document validation.

It is worth noticing that some SMT solvers call variables in a formula *constants*. We shall use the term *constant* to refer to an SMT formula variable in order to avoid conflict with the term variable referring to multimedia document variables.

The preceding paragraphs presented a brief description of SMT, a more complete

description of SMT is found in Appendix C.

4.4.1 Satisfaction Problem d_{SMT}

In *SMT*, document fragments are represented by their projections in the temporal and spatial dimensions. \mathcal{SHM} relations are represented in *SMT* by assertions, which may include inequalities among event occurrences. The representation of a document d as a satisfaction problem d_{SMT} is presented in Definition 3.

Definition 3 (d_{SMT}). The representation of a document through its intervals is specified in SMT as a satisfaction problem

$$d_{SMT} = a_1 \wedge a_2 \wedge \dots \wedge a_n$$

where a_i is an assertion, i.e., a logical formula combining logical connectives with atomic formulas, which can be either boolean formulas or arithmetic inequalities. Assertions are parameterized by constants, which can be either boolean or arithmetic. d_{SMT} defines the following constants for representing a fragment projection.

$$\begin{aligned} f:a^{beg}, f:a^{mid}, f:a^{end}, f:a^{exp}, T, I &\in \mathbb{R}_{\geq 0} \\ e:t^{plc}, e:s^{plc} &\in \text{Bool} \end{aligned}$$

where constants $f:a^{beg}$, $f:a^{mid}$, $f:a^{end}$, $f:a^{exp}$, $e:t^{plc}$ and $e:s^{plc}$ represent a fragment projection (its begin, center, end, expected size and if its part of the temporal/spatial layout) in a given axis $a \in \{t, x, y\}$. Constant T represents the global presentation time and constant I represents an *infinite time*. Constants $f:a^{beg}$, $f:a^{mid}$, $f:a^{end}$, $f:a^{exp}$ are related as follows.

$$f:a^{beg} < f:a^{end} \tag{4.21}$$

$$f:a^{mid} = (f:a^{beg} + f:a^{end}) / 2 \tag{4.22}$$

$$f:a^{end} = f:a^{beg} + f:a^{exp} \tag{4.23}$$

Furthermore, constants $f:s^{plc}$ and $f:t^{plc}$ are related as follows.

$$\begin{aligned} & \{f:t^{plc} \wedge T > f:t^{beg} \wedge T < f:t^{end} \wedge f:s^{plc}\} \quad \vee \\ & \{(\neg f:t^{plc} \vee T < f:t^{beg} \vee T > f:t^{end}) \wedge \neg f:s^{plc}\} \end{aligned} \quad (4.24)$$

Given the existence of several projections of a given fragment, it is assumed that a projection k has to occur after a projection $k - 1$. Moreover, given that a projection k exists, it means that a projection $k - 1$ also exists. This is formalized by the following constraints.

$$f:a_k^{beg} \geq f:a_{k-1}^{end}, \quad k > 1 \quad (4.25)$$

$$f:a_k^{plc} \rightarrow f:a_{k-1}^{plc}, \quad k > 1 \quad (4.26)$$

Moreover, given the existence of a pause projection for a given fragment f , the following constraints apply.

$$f:t^{beg} < f_p:t^{beg} \wedge f_p:t^{end} \leq f:t^{end} \quad (4.27)$$

$$f_p:t^{plc} \rightarrow f:t^{plc} \quad (4.28)$$

$$f_p:t^{plc} \rightarrow f_p:t^{exp} > 0 \quad (4.29)$$

$$\neg f_p:t^{plc} \rightarrow f_p:t^{exp} = 0 \quad (4.30)$$

$$f:t^{exp} = v + f_p:t^{exp} \quad (4.31)$$

Finally, constant I is related to constant T as follows.

$$T \leq I \quad (4.32)$$

Every fragment f projection in a document is inside a *canvas* defined as follows.

$$\text{canvas}:t^{\text{beg}} = 0 \quad (4.33)$$

$$\text{canvas}:t^{\text{beg}} \leq T \leq \text{canvas}:t^{\text{end}} \quad (4.34)$$

$$\text{canvas}:x^{\text{beg}} = 0 \quad (4.35)$$

$$\text{canvas}:y^{\text{beg}} = 0 \quad (4.36)$$

$$\text{canvas}:x^{\text{exp}} = \text{screen}.width \quad (4.37)$$

$$\text{canvas}:y^{\text{exp}} = \text{screen}.height \quad (4.38)$$

Thus the following assertion apply.

$$f:a^{\text{beg}} \geq \text{canvas}:a^{\text{beg}} \wedge f:a^{\text{end}} \leq \text{canvas}:a^{\text{end}} \quad (4.39)$$

for each axis a . Allen's relations are represented by the following assertions.

$$a \text{ before } b \equiv a:t^{\text{end}} < b:t^{\text{beg}} \quad (4.40)$$

$$a \text{ meets } b \equiv a:t^{\text{end}} = b:t^{\text{beg}} \quad (4.41)$$

$$a \text{ overlaps } b \equiv a:t^{\text{beg}} < b:t^{\text{beg}} \wedge b:t^{\text{beg}} < a:t^{\text{end}} \wedge a:t^{\text{end}} < b:t^{\text{end}} \quad (4.42)$$

$$a \text{ starts } b \equiv a:t^{\text{beg}} = b:t^{\text{beg}} \wedge a:t^{\text{end}} < b:t^{\text{end}} \quad (4.43)$$

$$a \text{ during } b \equiv a:t^{\text{beg}} > b:t^{\text{beg}} \wedge a:t^{\text{end}} < b:t^{\text{end}} \quad (4.44)$$

$$a \text{ finishes } b \equiv a:t^{\text{beg}} > b:t^{\text{beg}} \wedge a:t^{\text{end}} = b:t^{\text{end}} \quad (4.45)$$

$$a \text{ equals } b \equiv a:t^{\text{beg}} = b:t^{\text{beg}} \wedge a:t^{\text{end}} = b:t^{\text{end}} \quad (4.46)$$

Similar to Allen's relations, RCC relations are represented by the following assertions.

$$a \text{ dcon } b \equiv a:x^{\text{beg}} > b:x^{\text{end}} \vee a:x^{\text{end}} > b:x^{\text{beg}} \vee a:y^{\text{beg}} > b:y^{\text{end}} \vee a:y^{\text{end}} < b:y^{\text{beg}} \quad (4.47)$$

$$a \text{ econ } b \equiv ((a:x^{\text{beg}} = b:x^{\text{end}} \vee a:x^{\text{end}} = b:x^{\text{beg}}) \wedge a:y^{\text{beg}} \leq b:y^{\text{end}} \wedge a:y^{\text{end}} \geq b:y^{\text{beg}}) \vee ((a:y^{\text{beg}} = b:y^{\text{end}} \vee a:y^{\text{end}} = b:y^{\text{beg}}) \wedge a:x^{\text{beg}} \leq b:x^{\text{end}} \wedge a:x^{\text{end}} \geq b:x^{\text{beg}}) \quad (4.48)$$

$$\begin{aligned}
& (a:x^{beg} < b:x^{beg} \wedge a:x^{end} > b:x^{beg} \wedge a:x^{end} < b:x^{end} \\
& \wedge a:y^{beg} < b:y^{end} \wedge a:y^{end} > b:y^{beg}) \vee \\
& (a:x^{beg} > b:x^{beg} \wedge a:x^{beg} < b:x^{end} \wedge a:x^{end} > b:x^{end} \\
& \wedge a:y^{beg} < b:y^{end} \wedge a:y^{end} > b:y^{beg}) \vee \\
a \text{ pover } b \equiv & (a:y^{beg} < b:y^{beg} \wedge a:y^{end} > b:y^{beg} \wedge a:y^{end} < b:y^{end} \\
& \wedge a:x^{beg} < b:x^{end} \wedge a:x^{end} > b:x^{beg}) \vee \\
& (a:y^{beg} > b:y^{beg} \wedge a:y^{beg} < b:y^{end} \wedge a:y^{end} > b:y^{end} \\
& \wedge a:x^{beg} < b:x^{end} \wedge a:x^{end} > b:x^{beg})
\end{aligned} \tag{4.49}$$

$$\begin{aligned}
& ((a:x^{exp} < b:x^{exp} \wedge a:y^{exp} \leq b:y^{exp}) \vee \\
& (a:x^{exp} \leq b:x^{exp} \wedge a:y^{exp} < b:y^{exp})) \wedge \\
a \text{ tpp } b \equiv & (((a:x^{beg} = b:x^{beg} \vee a:x^{end} = b:x^{end}) \wedge \\
& a:y^{beg} \geq b:y^{beg} \wedge a:y^{end} \leq b:y^{end}) \vee \\
& ((a:y^{beg} = b:y^{beg} \vee a:y^{end} = b:y^{end}) \wedge \\
& a:x^{beg} \geq b:x^{beg} \wedge a:x^{end} \leq b:x^{end}))
\end{aligned} \tag{4.50}$$

$$a \text{ ntpp } b \equiv a:x^{beg} > b:x^{beg} \wedge a:x^{end} < b:x^{end} \wedge \tag{4.51} \\
a:y^{beg} > b:y^{beg} \wedge a:y^{end} < b:y^{end}$$

$$a \text{ equal } b \equiv a:x^{beg} = b:x^{beg} \wedge a:x^{end} = b:x^{end} \wedge \tag{4.52} \\
a:y^{beg} = b:y^{beg} \wedge a:y^{end} = b:y^{end}$$

For each RCC relation defining an angle, the following additional assertions are applied, where a is the angle, d is the distance between the fragment region centers and k is a delta for the angle (see Section 4.1.2).

$$c_{max} = \cos((a + k) * (\pi / 180)) * d$$

$$c_{min} = \cos((a - k) * (\pi / 180)) * d$$

$$s_{max} = \sin((a + k) * (\pi / 180)) * d$$

$$s_{min} = \sin((a - k) * (\pi / 180)) * d$$

$$c_{min} < c_{max} \rightarrow (c_{min} \leq a:x^{mid} - b:x^{mid} \leq c_{max}) \quad (4.53)$$

$$c_{min} > c_{max} \rightarrow (c_{max} \leq a:x^{mid} - b:x^{mid} \leq c_{min}) \quad (4.54)$$

$$c_{min} = c_{max} \wedge c_{max} > 0 \rightarrow (c_{max} \leq a:x^{mid} - b:x^{mid} \leq d) \quad (4.55)$$

$$c_{min} = c_{max} \wedge c_{max} < 0 \rightarrow (-d \leq a:x^{mid} - b:x^{mid} \leq c_{max}) \quad (4.56)$$

$$s_{min} < s_{max} \rightarrow (s_{min} \leq b:y^{mid} - a:y^{mid} \leq s_{max}) \quad (4.57)$$

$$s_{min} > s_{max} \rightarrow (s_{max} \leq b:y^{mid} - a:y^{mid} \leq s_{min}) \quad (4.58)$$

$$s_{min} = s_{max} \wedge s_{max} > 0 \rightarrow (s_{max} \leq b:y^{mid} - a:y^{mid} \leq d) \quad (4.59)$$

$$s_{min} = s_{max} \wedge s_{max} < 0 \rightarrow (-d \leq b:y^{mid} - a:y^{mid} \leq s_{max}) \quad (4.60)$$

In the above assertions, the auxiliary variables c_{min} , c_{max} , s_{min} and s_{max} are used for calculating the angle between two fragments. Causal relations in *SMT* are defined over the set of event occurrences $Evt = \{\triangleright, \square\}$ (\triangleright representing event start and \square representing event stop) and document fragments (represented here by sort *Fragment*). Thus a relation $r : \epsilon \rightarrow A$, where ϵ is an event occurrence and $A = Evt \times Fragment$ is represented in *SMT* as follows.

$$\epsilon \rightarrow \triangleright e \equiv e:t^{beg} = \omega(\epsilon) \wedge e:t^{plc} \text{ if } \neg(e:t^{beg} \leq \omega(\epsilon) < e:t^{end}) \quad (4.61)$$

$$\epsilon \rightarrow \square e \equiv e:t^{end} = \omega(\epsilon) \text{ if } e:t^{plc} \wedge (e:t^{beg} \leq \omega(\epsilon) < e:t^{end} + e:t^{exp}) \quad (4.62)$$

where $\omega(\epsilon)$ denotes the time when event ϵ occurs. Finally, relations *set* and *animate* are defined as follows, where i is the interval for which variable x is set to value v .

$$\{i:t^{plc} \wedge T > i:t^{beg} \wedge T < i:t^{end}\} \rightarrow x = v \quad (4.63)$$

Such that, given that T 's value is between $i:t^{beg}$ and $i:t^{end}$, then variable x 's value is equal to v . \triangle

The \mathcal{SHM} model, as presented in Section 4.1, is based on events, where document fragments (media items, media fragments and compositions) are associated to state machines representing its *presentation*, *selection* and *attribution* states. Event occurrences, as presented in Figure 4.1, represent changes in the state of fragments that occur during document presentation. We repeat that figure once again to improve this section readability.

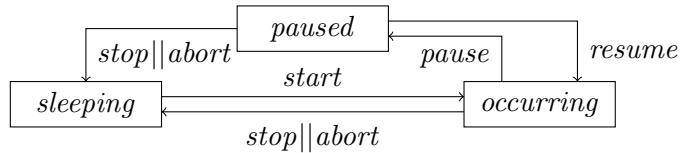


Figure 4.14: State changes

It is possible to project the *presentation*, *selection* and *attribution* of document fragments in time, as presented in Section 4.1.1. Each projection is an interval representing the amount of time a media item is in the *occurring* or *paused* state. Figure 4.3 presented an example where projections represent the *occurring* and *paused* states of a fragment. We repeat Figure 4.3 here to improve this section readability.

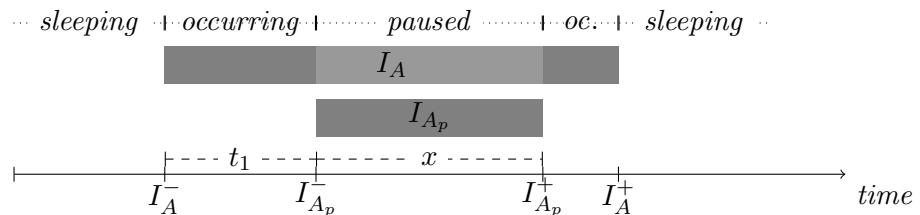


Figure 4.15: Paused element projection

Different from the temporal dimension, in the spatial dimension, \mathcal{SHM} represents document fragments as rectangular regions. Such a representation is presented in Section 4.1.2.

In *SMT*, document fragments are represented by their projections in the temporal dimension and by their rectangular regions in the spatial dimension. Regions in space are represented by their projection in either x and y axes. Projections in time or space are represented using the following constants:

- $f:a^{beg}$ represents the begin of a projection of fragment f in axis a ;
- $f:a^{mid}$ represents the center of a projection of fragment f in axis a ;
- $f:a^{end}$ represents the end of a projection of fragment f in axis a ;

- $f:a^{exp}$ represents the expected size of a projection of fragment f in axis a .

The constraint in Equation 4.21 states that a projection must have a size bigger than zero. The constraint in Equation 4.22 defines the center of a projection in relation to its begin and end. Besides the above constraints, in Equation 4.23 *SMT* relates the end of projection with its expected size. It is important to highlight, however, that such relation is asserted together with other relations that constrain the end of a projection, as will be discussed latter in this section.

It is possible for a given document fragment to be presented more than once during document presentation. In this case, a fragment will enter in the *occurring* state more than once. As a result we have more than one projection for such a fragment in time. Each individual projection of a fragment f in time is called an *occurrence* of f . It is important to notice that, whenever a fragment has a projection in time, it implies that the given fragment will also have a projection in space, i.e., in the x and y axes. Exceptions to this are media items without visual representation, such as an audio object, viewer selection and variable attribution. A subscript $k \in [1, n]$ represents different occurrences of f in a , where n is the total number of occurrences of f in a .

Besides constants for defining the fragment projection, *SMT* defines the boolean constants $f:t^{plc}$ and $f:s^{plc}$ with the following meaning:

- $f:t^{plc}$ indicates if a given projection is considered or not to be part of the temporal layout;
- $f:s^{plc}$ indicates whether or not the media item is being presented on the screen.

As an example of use of constant $f:t^{plc}$, suppose the example presented in Figure 4.6 where either interval I_{m_2} or interval I_{m_3} are part of the resulting temporal layout. In the case where I_{m_2} is presented (Figure 4.6a), we have

$$m_2:t^{plc} = \text{true} \wedge m_3:t^{plc} = \text{false},$$

while in the second case, where I_{m_3} is presented (Figure 4.6b), we have

$$m_2:t^{plc} = \text{false} \wedge m_3:t^{plc} = \text{true}.$$

As an example of use of constant $f:s^{plc}$, suppose the example presented in Figure 4.6a, where intervals I_{m_1} and I_{m_2} are presented at different times on the exhibition device. In

the beginning of the document presentation, only I_{m_1} is presented. At this point we have

$$m_1:s^{plc} = \text{true} \wedge m_2:s^{plc} = \text{false},$$

while after I_{m_1} ends its presentation, we have

$$m_1:s^{plc} = \text{false} \wedge m_2:s^{plc} = \text{true}.$$

The relation between both constants $f:t^{plc}$ and $f:s^{plc}$ is given by Equation 4.24, which has the following meaning. Whenever a fragment is part of the temporal layout (i.e., $f:t^{plc} = \text{true}$) and global time T is inside the fragment's projection in time (i.e., between $f:t^{beg}$ and $f:t^{end}$), variable $f:s^{plc}$ is true and false elsewhere.

From Figure 4.3, it is possible to notice that, given the existence in time of a “pause projection” f_p of fragment f , the resulting size of f 's projection will be its original value plus the size of f_p . Definition 3 states that a given “pause projection” has to be inside the fragment projection (Equation 4.27); if a “pause projection” exists, than the fragment projection also exists (Equation 4.28); the expected size of a “pause projection” is greater than zero if it exists or equal to zero in the opposite case (Equations 4.29 and 4.30); and that the size of a fragment projection is its declared size plus the “pause projection” size (Equation 4.31).

As discussed in Section 4.1.1, it is possible for a fragment to have an infinite duration. This is represented in *SMT* by setting the end of a fragment projection equals to a constant I , representing an *infinite time*.

Every element is represented inside a *canvas*, which represents the document as a whole in both time and space. In the spatial axis, the *canvas* has the size of the exhibition device. In the temporal axis, the *canvas* does not have a predefined size.

It is worth noticing that the constants used in the representation of document fragments in either space and time introduce some redundancy. Some examples of redundant constant are $f:a^{mid}$, $f:a^{exp}$ and $f:s^{plc}$. This redundancy was included in order to ease the definition of expressions over projections.

As seen in Section 4.1.1, *SHM* enables defining both causal relations and constraints. Such relations are represented in *SMT* by assertions, which may include inequalities among the constants representing projections and boolean operations over variables $f:t^{plc}$ and $f:s^{plc}$.

Constraints are expressed in time with Allen's relations [Allen 1983] between time

intervals and in space by RCC relations [Randell et al. 1992]. Definition 3 presents the representation of both Allen and RCC relations in terms of constants $f:a^{beg}$, $f:a^{mid}$, $f:a^{end}$, $f:a^{exp}$, $e:t^{plc}$ and $e:s^{plc}$ between two fragments a and b .

In Section 4.1.2, we presented an extension of RCC relations (except *equal*), so that they are parameterized by the angle and distance between region centers. Moreover, relations *dcon*, *pover* and *ntpp* are also parameterized by the distance between region centers. Definition 3 also defines additional constraints to be applied whenever a given RCC relation defines an angle.

Recalling Section 4.1.1, causal relations are defined over event occurrences in the form

$$\{\epsilon_1, \dots, \epsilon_m\} \rightarrow \epsilon_n$$

where event ϵ_n will occur when the first event in $\{\epsilon_1, \dots, \epsilon_m\}$ occurs. If no event in $\{\epsilon_1, \dots, \epsilon_m\}$ occurs during document execution, event ϵ_n will not occur either.

In this section, state changes depicted in Figure 4.1 are represented with the following graphical notation: \triangleright (*start*), \square (*stop*), \bowtie (*abort*), $\bowtie\bowtie$ (*pause*) and $\bowtie\triangleright$ (*resume*).

Given that a fragment state may be projected in the temporal axis as presented above, we have the following equivalences:

- $\bowtie e \equiv \triangleright e_p$: given that the pause state is represented by a “pause projection”, pausing a fragment e is equivalent to starting a “pause projection” e_p .
- $\bowtie\triangleright e \equiv \square e_p$: given that the pause state is represented by a “pause projection”, resuming a fragment e is equivalent to stopping a “pause projection” e_p .
- $\bowtie\bowtie e \equiv \square e$: aborting or stopping a fragment e will produce the same output in the resulting presentation, i.e., e will end its presentation⁴.

Therefore, it is possible to reduce the set of event occurrences to be used in causal relations to $\{\triangleright, \square\}$. It is worth mentioning that, by treating actions \bowtie and \square as the same, we loose expressiveness in comparison to *SHM*. Such a loss is given by the fact that we can not create properties differentiating both actions.

In Definition 3, Equation 4.61 states that the begin of a fragment projection will happen at the same time of event ϵ , given that it does not occur while the given fragment

⁴In NCL, both actions are used together as a way to provide two different ways of ending an element’s presentation, and thus avoiding triggering links (a \bowtie will not trigger links specified for a \square and vice versa).

is already being presented. Equation 4.62 states that the end of a fragment projection will happen at the same time of event ϵ , given that it occurs while the given fragment is being presented.

It is possible to observe that more than one event may be defined at the left-hand side of a causal relation, i.e., more than one event occurrence may trigger the relation. As stated at the definition of causal relations, whenever more than one event is defined at the left-hand side of a causal relation, the first one to occur will be the one triggering the relation. *SMT* models such a behavior with operator *first*. It receives as parameter a list of events and returns the first one to occur. For example, suppose the following causal relation:

$$\{\epsilon_1, \epsilon_2\} \rightarrow \square e$$

It will be represented by the following constraint:

$$e:t^{end} = \text{first}(\epsilon_1, \epsilon_2)$$

Such that the value of $e:t^{end}$ will be the same of (i) ϵ_1 or ϵ_2 , in case just one of them occurs; (ii) the first to occur between ϵ_1 and ϵ_2 , in case both occur; or (iii) the first argument of function *first*, in case both ϵ_1 and ϵ_2 occur at the same time. For this specific example, function *first* represents the following formula.

$$\begin{aligned} & \{\exists \epsilon_2 \wedge (\omega(\epsilon_2) < \omega(\epsilon_1) \vee \neg \exists \epsilon_1) \wedge e:t^{end} = \omega(\epsilon_2)\} \vee \\ & \{\exists \epsilon_1 \wedge (\omega(\epsilon_1) \leq \omega(\epsilon_2) \vee \neg \exists \epsilon_2) \wedge e:t^{end} = \omega(\epsilon_1)\} \end{aligned}$$

It is worth noticing that, when relations define the end of a projection, operator *first* may define an additional default value to be chosen when none of the events occurs. Such a default value may be the one defined in Equation 4.23 when the fragment has an intrinsic duration or equals to infinite (constant I) otherwise.

Section 4.1.3 presented relations *set* and *animate*, which may be used for defining the value of a given variable instantaneously or incrementally, respectively. Both relations are associated to a given interval such that the value of a given variable x is equal to v inside such interval and some other value (possibly the same) outside. Value v will be either a number or a polynomial in function of constant T .

4.4.2 SMT Document Validation

In the previous sections, we presented how representation d_{SMT} is obtained from document d through transformation $\tau_{SMT}(d)$. In short, document fragments are represented as a set of constants and document relations are represented by constraints, i.e., formulas containing inequalities over those constants.

In order to provide validation for a given document, constraints representing that document's spatio-temporal layout are fed to an SMT solver. The solver then computes the conjunction of all constraints. When the whole formula holds, it presents back a possible valuation for constants. Validation in *SMT*, therefore, can be provided in the following ways.

The first way to provide validation is by checking the possibility of evaluating d_{SMT} . Given that no valuation is provided by the SMT solver, it is possible to infer that the set of relations in d is inconsistent. It is worth mentioning that this approach is commonly used during document creation, as a new relation is included in the document layout. Whenever validation returns an error, it is possible to infer that the new relation made the document layout inconsistent.

The second way to provide validation is by computing the conjunction $d_{SMT} \wedge P$, where P represents a given property or the viewer context. In the first case, given that a valuation is provided by the solver, it is possible to infer that a given property P holds. In the second case, given that a valuation is provided by the solver, it is possible to infer that d is adapted to the viewer context.

Finally, a third way of providing validation for d is to compute the conjunction $d_{SMT} \wedge \neg P$, where P represents an undesirable behavior. In that case, given that no valuation is provided, it is possible to infer that this behavior will never occur during document execution.

Chapter 5

Implementation and Use

This chapter presents the implementation of the validation approach presented in Chapter 4. Both validation techniques, presented in Chapters 4.3 and 4.4 have been implemented into validation tools, presented in Sections 5.1 and 5.3, respectively.

Moreover, this chapter also presents practical results about the use of both validation techniques. Section 5.2 describes the results obtained by using our *RWT* implementation and Section 5.4 does the same for the *SMT* implementation. At the end of this chapter, Section 5.5 presents a use case where both validation techniques may be applied and Section 5.6 provides a discussion about the choice for a given technique and limitations in the tools here presented.

5.1 *RWT* Implementation

This section presents the tools that implement the validation approach proposed in this work. The tools presented in this section are used for the validation of NCL documents. Each tool is described in one of the following sections.

5.1.1 aNaa

API for NCL Authoring and Analysis (aNaa) is an API built for being used by other tools that focus on the authoring of NCL documents. The idea is to improve such tool by providing the validation of a document being created. aNaa partially implements the validation approach proposed in this work, dealing only with the *RWT* validation technique, described in Chapter 4.3.

The process of transforming an NCL document into a rewriting theory in the *RWT* approach and performing its validation is done in the following steps. (i) First aNaa gathers the information contained in an NCL document and represents it as an equational theory $\mathcal{E} = (\Sigma, E)$. (ii) It triggers the application of a set of Maude equations in E to create a Maude module representing the given document in *RWT*. (iii) Finally, it calls Maude to perform the validation using the Maude model-checker tool. Figure 5.1 presents the architecture of aNaa.

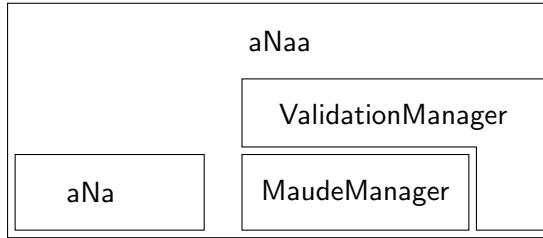


Figure 5.1: aNaa API architecture

In order to gather the information contained in an NCL document necessary for providing its validation, aNaa extends the API for NCL Authoring (aNa) API [dos Santos et al. 2012b]. aNa was created to represent an NCL document. Its structure is optimized so the author of NCL tools that manipulate NCL XML code does not need to worry about the language representation. aNa defines Java classes that represent NCL elements following the characteristics presented in Section A.1.

Every aNa class is extended with new methods to create the representation of a given NCL element into \mathcal{E} . The most important methods are the *getElementId()* and the *createElement()* methods. The former creates a unique identification for a given NCL element. The latter creates a string representing a given NCL element in Maude as specified in theory \mathcal{E} .

Theory $\mathcal{E} = (\Sigma, E)$ is an equational theory that represents the syntax of XML elements with Σ and E represents the mapping of such elements into *RWT*. Σ is defined as follows.

```

1   sorts AttributeName AttributeValue Attribute AttributeSet .
2   sorts ElementName Element ElementSet .
3
4   subsort Attribute < AttributeSet .
5   subsort Element < ElementSet .
6
7   op attr(.,.) : AttributeName AttributeValue -> Attribute [ctor] .
8
9   op empty : -> AttributeSet .
10  op -- : AttributeSet AttributeSet -> AttributeSet [assoc comm id: empty ctor] .
11
12  op {--|-;-} : ElementName Qid Qid AttributeSet -> Element [ctor] .
13
14  op empty : -> ElementSet [ctor] .
15  op -- : ElementSet ElementSet -> ElementSet [assoc comm id: empty ctor] .
  
```

Thus, for each multimedia language to be translated into *RWT*, theory \mathcal{E} is extended. The Σ of the extended theory defines constants of sorts *ElementName* and *AttributeName*, one for each element and attribute in the given language, respectively. Moreover, Σ will also have constructs for representing the possible attribute values as elements of sort *AttributeValue*.

For the validation of NCL documents, we extend theory \mathcal{E} into theory \mathcal{E}_{NCL} as discussed above. For a given document, aNaa represents in theory \mathcal{E}_{NCL} a document by a set of elements, represented by sort *ElementSet*. For each NCL element of a given document, one element of sort *Element* is created. Each element e of sort *Element*, a unique *id* (represented as a quoted identifier, i.e., a String preceded by a quote) is defined. For a given NCL element, all its attributes are represented as elements of sort *Attribute*. The hierarchy of NCL elements is represented by indicating, for each element e , the *id* of its parent element, captured by the second parameter in operator $\{__|_ \}$. When element e is the document root element, it will declare ‘nil’ as its parent element *id*.

Suppose the following example of NCL element presented in Listing 5.1.

```

1 <media id="vdoComercial" src="comercial.mp4" descriptor="dComercial">
2   <area id="areaInter" begin="16s" end="76s"/>
3 <\media>
```

Listing 5.1: NCL element example

Listing 5.2 gives the representation of the NCL element presented in Listing 5.1 as a term in theory \mathcal{E}_{NCL} produced by method *createElement()* from aNaa.

```

1 {media 'ELM142' 'ELM130 | attr(id,str("vdoComercial")), attr(src,str("comercial.mp4")), attr(descriptor,
2   eid('ELM028)), attr(mediaType,str("AUDIO")), attr(duration,num(107.0, S))},
3  {area 'ELM143' 'ELM142 | attr(id,str("areaInter")), attr(begin,num(16.0, S)), attr(end,num(76.0, S))
 },
```

Listing 5.2: NCL element representation in \mathcal{E}_{NCL}

The description of the process implemented by the equations in E_{NCL} for transforming an NCL document into *RWT* is presented in Appendix A.

Although aNaa is implemented for validating NCL documents, we have also implemented an extension of \mathcal{E} for the SMIL language. Theory \mathcal{E}_{SMIL} will define the constants for representing SMIL elements in Σ_{SMIL} and the mapping from SMIL to *RWT* in E_{SMIL} . The description of the process implemented by equations in E_{SMIL} for transforming a SMIL document into *RWT* is also presented in Appendix A.

Finally, after getting the representation of a document in *RWT*, aNaa triggers the document validation using its model-checker tool. The validation process using Maude is

presented in Section 4.3.3.

In order to be able to call Maude for executing the validation, aNaa implements packages *MaudeManager* and *ValidationManager*. Figures 5.2 and 5.3 present the diagrams of both packages.

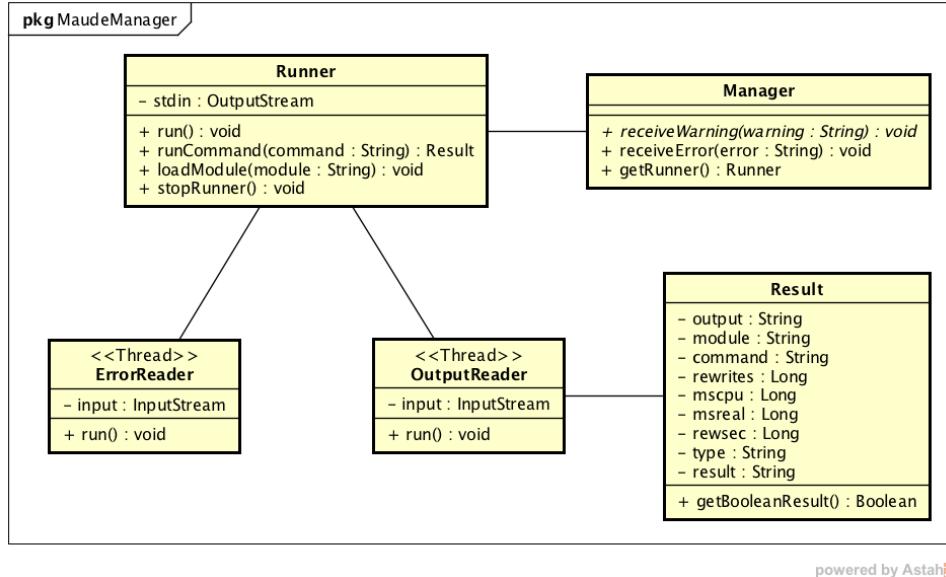
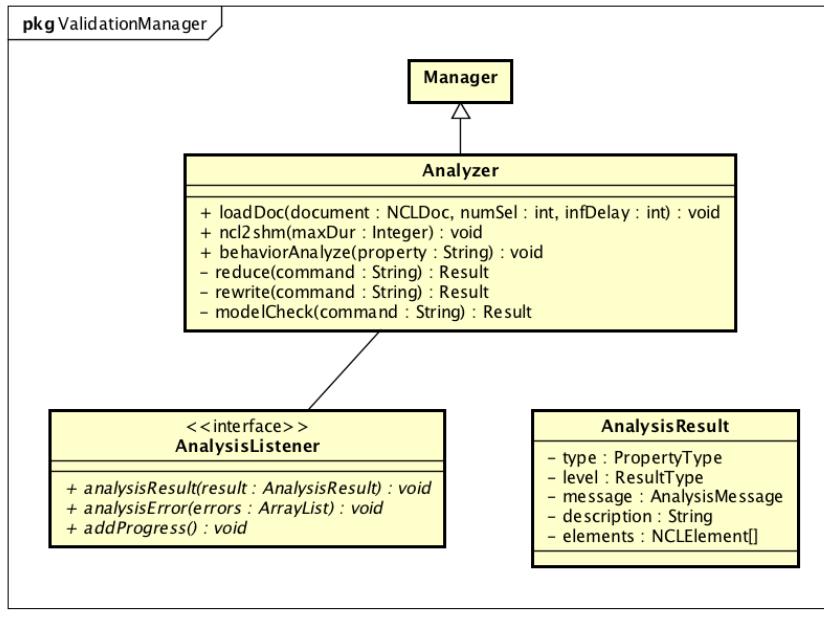


Figure 5.2: *MaudeManager* package

Figure 5.2 presents the *MaudeManager* package. It is used to control the Maude tool and has two main classes: *Runner* and *Manager*. The *Runner* class calls Maude as a system process and creates objects for writing an input into Maude and reading its output. Class *Runner* provides method *runCommand()*, which receives as a parameter a Maude command to be executed and returns an object of class *Result*. Class *Result* is used for storing the information provided by Maude about a given executed command. The *Manager* class defines an interface for receiving errors and warnings from Maude and accessing a *Runner* in order to send a command to Maude.

Figure 5.3 presents the *ValidationManager* package. It is responsible for performing the document validation and has two main classes: *Analyzer* and *AnalysisListener*. The *Analyzer* class extends the *Manager* from the *MaudeManager* package. It receives the document to be validated (as Java objects in aNaa) through method *loadDoc()*. Additionally, this method receives information about how to perform the validation, such as the number of viewer interactions to be performed over a selectable media item.

The transformation from NCL is triggered by method *ncl2shm()*. It uses method *createElement()* of each document element for creating the representation of the document using the equational theory \mathcal{E} . It will then load the Maude module declaring Σ_{NCL} (as

Figure 5.3: *ValidationManager* package

presented above) and the document representation $\mathcal{E}_{NCL}(d)$. It then sends a command to Maude for rewriting $\mathcal{E}_{NCL}(d)$ into its representation in *RWT*.

Once the Maude module representing the document in *RWT* is created, validation is performed by calling method *behaviorAnalyze()* providing a property to be validated. Function *behaviorAnalyze()* executes the Maude model-checker tool whenever it is called in order to perform the validation of a given property.

While executing the transformation and the document validation, class *Analyzer* produces an output that can be presented to the user. The *AnalysisListener* class defines an interface for receiving this output, which can be the result of a property validation (either *true* or a counterexample), a progress indicator or an error in the tool execution.

The aNaa API was tested by validating documents developed by the NCL community available at the NCL Club. A more detailed discussion is presented in Section 5.2.

Two tools were built upon the aNaa API. They are briefly presented in Section 5.1.2 and 5.1.3.

5.1.2 Property editor

In [Batista 2015], the author presents an editor for creating user-defined properties to be validated in a multimedia document. The editor presented in [Batista 2015] is built upon the aNaa API. Figure 5.4 presents the property editor interface.

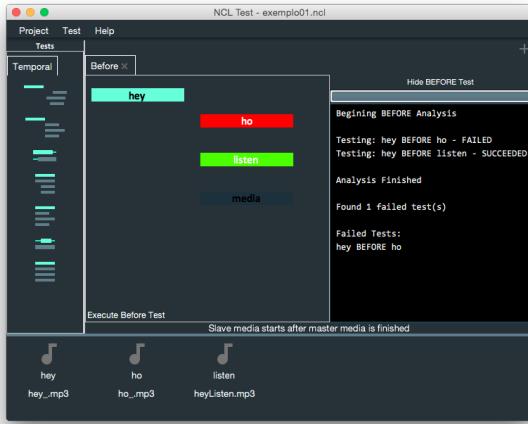


Figure 5.4: Property editor interface

The editor provides to the author a graphical way for defining behavioral properties and perceiving the validation result. In its current implementation, the editor performs only the validation of temporal properties. It represents the Allen's relations presented in Section 4.1.1 as unitary test types. For each created test, the user chooses the media items to be considered. The media items to be used for creating tests are gathered from the document to be validated.

Once the tests are created, the user has the option of validating a single test or the test suite as a whole. The tool shows the result of the test, marking as green and red media items that passed the test or not, respectively.

A usability test [Batista 2015] was conducted with the property editor with test subjects ranging from experts to users without knowledge of NCL. The tool was evaluated using the System Usability Scale (SUS) technique. Test subjects were also able to provide general comments about the tool.

5.1.3 aNaa4Web

We have implemented a validation tool for NCL documents on top of aNaa. It is called aNaa4Web. Figure 5.5 presents the tool interface.

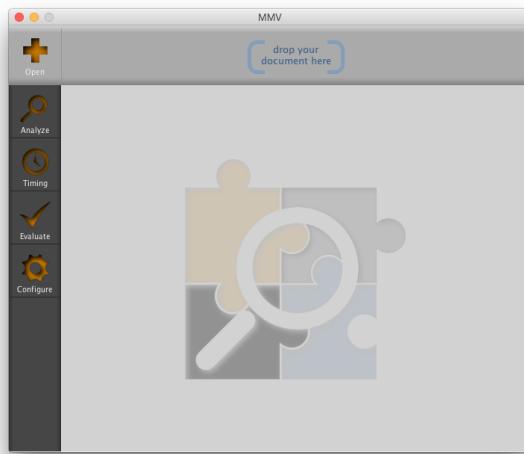


Figure 5.5: aNaa4Web interface

Using aNaa4Web, the author chooses a document to be validated. The information about the document is presented in the upper part of the tool as seen in Figure 5.6.

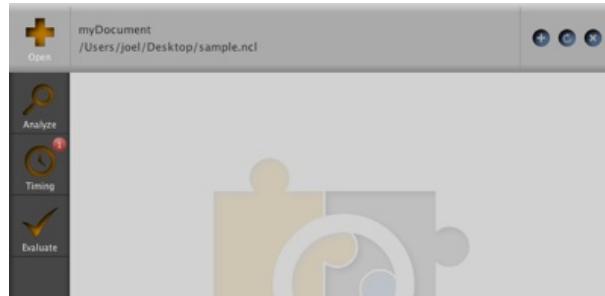


Figure 5.6: Document information in aNaa4Web

Once the document is open, the tool asks the user to provide the duration of document media items. For each media item with a duration (audio and video), the tool presents a field where the user can indicate its value. Figure 5.7 presents the field for indicating media item duration.

Finally, the user can choose to validate the document. *Error* and *warning* messages are presented as indicated in Figure 5.8.

Following the error messages presented by the tool, the user can correct the document accordingly.



Figure 5.7: Media item duration



Figure 5.8: Error and warning messages

The aNaa4Web tool was used by NCL developers and in classes where the NCL language was taught. In order to enable its user evaluation the aNaa4Web tool provides a SUS-like questionnaire.

5.2 *RWT* Implementation in Practice

Since its adoption as a standard for Digital TV, both NCL standard [ABNT 2011] and its reference implementation have evolved in the past few years. We used our *RWT* validation technique to validate real documents created by the NCL community which are available at a code repository called NCL Club (<http://clube.ncl.org.br>). They were created in different moments in time, and therefore, they comply with different versions of the NCL standard and its reference implementation. As a consequence, they may not work as expected or simply do not work at all in the current version of NCL standard (2011) and its reference implementation (version 0.12.4). Most of the validated documents are non-trivial with respect to document structure (many levels of nested document elements), number of declarations (a few thousand lines of declarations) and size of the associated automata (millions of states).

Although it is not the focus of this work, the *RWT* technique also allows for the structural validation of multimedia documents. Therefore, while validating such documents we identified the following structural problems: (i) links instantiating parameterized connectors without defining the value of its parameters; (ii) wrong references among elements, through their identifiers. As we mentioned before, these problems arose as consequence of the evolution of the standard and they are not related to the authoring process *per se*.

However, our tool was able to identify inconsistencies in real NCL documents that arose from the standard evolution.

The behavioral analysis identified documents that could not end (up to a time bound defined by the validation tool used), depending on how the user interacts with them, and also documents that simply did not work at all with respect to the NCL reference implementation. In a careful analysis of the documents that did not work, by comparing their traces in Maude and their execution in the Ginga-NCL reference implementation, we were able to identify errors in the Ginga-NCL middleware reference implementation version 0.12.4¹.

The first problem found regards NCL *context* elements and its inner elements. Intuitively, suppose c a *context* element, E_i the set of its inner elements and $E_p \subseteq E_i$, such that every element $e \in E_p$ is mapped by a port of c . While c is in the *occurring* state, it would keep relaying *start* actions to elements in E_p . This behavior is incorrect: once in the *occurring* state, an NCL node should not accept *start* actions. Furthermore, once c is in the *occurring* state and receives a *stop* action, it should be relayed to every element in E_i . However, the *stop* action is not relayed to the elements in E_i that represent possibly reused nodes.

The second problem found regards NCL *switch* elements. Suppose a *switch* s that define rules $[r_1, \dots, r_n]$. The evaluation of rules follows their order and the first rule r_i whose condition is evaluated as *true* is chosen. However, in the reference implementation, given that two rules r_i and r_j ($i < j$) have conditions evaluated as *true*, the last one (r_j) is chosen.

These problems were reported back to the group that maintains Ginga-NCL to be addressed in future releases of the reference implementation.

As an indication of a reasonable performance of our implementation, we present performance results of the validation of three real documents found in the NCL Club. A brief overview about each document is given as follows.

“First João” presents an animation about a famous Brazilian soccer player named Garrincha. It synchronizes a video and a photo of kids performing the same dribbles as Garrincha with the main video. The user may interact with the application pressing the red key at the moment a soccer shoes icon appears. The animation is resized and a video of a kid thinking about shoes starts playing.

¹Version available at <http://www.ginga.org.br>

“Live More” presents a TV show discussing health and welfare. While the TV show is playing, an interaction icon appears. If the user presses the remote control red key, four different meal options appear. The user chooses a meal by pressing one of the colored keys of the remote control. When a meal is chosen, the user is informed about the quality of his choice, telling whether there are missing nutrients or nutrients in excess.

“Day’s Route” presents a non-linear show where the user can make his own narrative line by choosing the next sight in a city tour around Rio de Janeiro. At the beginning of the show, the user may choose to interact or not with the application. If the user chooses not to interact, a default tour is presented.

Table D.1 summarizes the characteristics of the three documents. As an indication of the document sizes, it presents their number of NCL elements and the number of *fragments* and *causal relations* when represented in \mathcal{SHM} .

Table 5.1: Document sizes and validation time comparison

Document			“First João”	“Live More”	“Day’s Route”
NCL	Elements		160	121	229
	Nodes		9	14	33
	Anchors		11	15	18
	Links		8	9	16
\mathcal{SHM}	Fragments		17	17	26
	Relations		15	13	30
	\mathcal{R}_{RWT}	States	19	23	45
		Transitions	19	26	44
Validation	Structural	Invariants	663	469	850
		Time	1,490ms	856ms	2,197ms
		Average	2.25ms	1.83ms	2.58ms
	Behavioral	Properties	23	31	33
		Time	386ms	936ms	882ms
		Average	16.78ms	30.19ms	26.72ms

The behavioral validation of each document creates the document *RWT* counterpart and model-checks it verifying the behavioral properties. Table D.1 presents the time spent for the behavioral validation and statistics of the \mathcal{SHM} counterpart of each document. It also presents the number of properties tested for each document. The tests presented here were conducted in an Intel i5 computer with 16GB memory.

In order to indicate the reasonable performance of our tool, we compared it with two other tools used for NCL document validation. Both tools were presented in Chapter 3. It is worth noticing that the durations presented for both tools are estimated.

The first one [Neto et al. 2011] is a structural validation tool and part of the Composer² authoring tool. It was used for the analysis of “First João”, “Live More” and “Day’s Route”. The time it spent for the structural validation was around 4s, 3.5s and 9s, respectively.

The second one [Júnior et al. 2014] is a behavioral validation tool. It was used for the analysis of “First João” and “Live More”. The same behavior identified by our tool was identified using this tool. The time it spent for the behavior validation was around 15s and 40s, respectively³.

5.3 SMT Implementation

SMT Validator is a tool implemented for performing the *SMT* validation technique presented in Chapter 4.4. The validator tool is implemented in Lua on top of the Yices2 [Dutertre 2014] solver. Figure 5.9 presents the architecture of the SMT Validator tool.

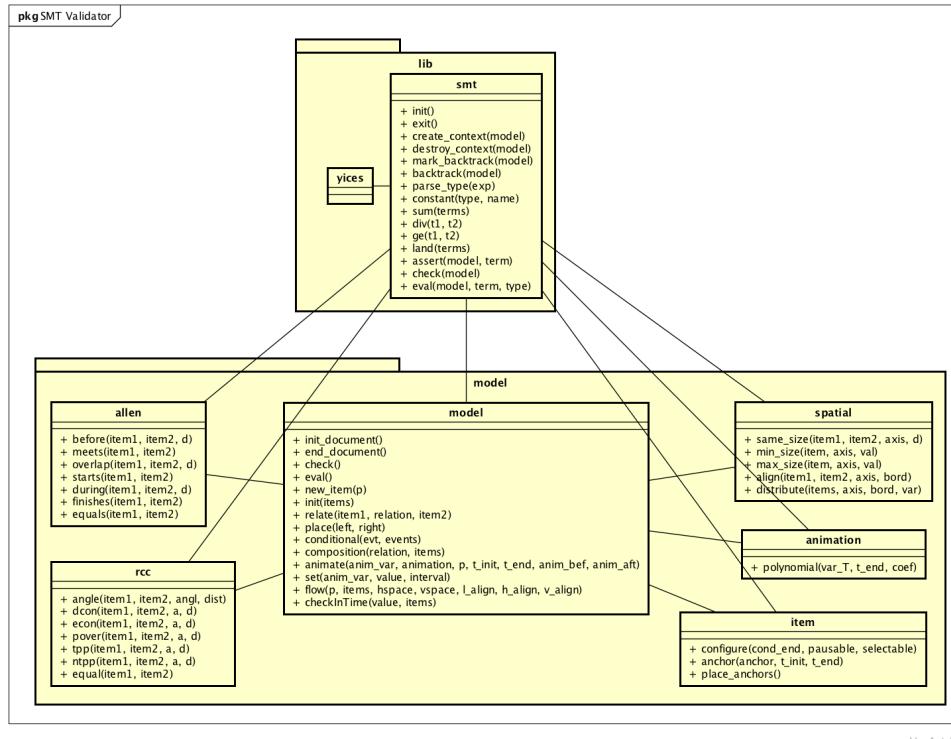


Figure 5.9: SMT Validator architecture

SMT Validator is implemented using object orientation in Lua, as presented in Fig-

²<http://composer.telemidia.puc-rio.br>

³Information kindly given by the authors of [Júnior et al. 2014]

ure 5.9. It is divided in two main packages: *lib* and *model*. Package *lib* defines classes for the interface with the solver. The Yices2 API is implemented in C. Class *yices* uses the Lua C API in order to export all the Yices2 API functions to Lua. Class *smt* creates an abstraction layer on top of *yices*. It provides functions for the operations used for representing document elements and relations. It also provides functions for controlling the Yices2 API, such as creating a context, backtracking, checking the context and evaluating constants.

Package *model* defines classes for representing the document as a whole, media items and spatio-temporal relations. Classes *allen*, *rcc*, *spatial* and *animation* define relations to be used in the document representation in *SMT*. It is worth noticing that, by calling the functions defined in each class, the given relation is automatically translated into constants and formulas in the Yices2 solver.

Classes *item* and *model* are responsible for creating the representation of a media item and the document as a whole, respectively. Listing 5.3 presents an extract of the representation of an NCL document in *SHM* using the SMT Validator tool.

```

1 local model = require('model')
2 local allen = require('model.allen')
3
4 m = model:new{ scenario = SCENARIO.ST, x_size = 640, y_size = 480, num_selec = 1}
5 m:init_document()
6
7 animation = m:new_item{t_size = 71}
8 segIcon = m:new_item()
9 animation:anchor(segIcon, 45, 51)
10 animation:place_anchors()
11
12 icon = m:new_item{t_size = 6, x_init = 560, y_init = 55, x_size = 54, y_size = 32, cond_end = true,
13   selectable = true}
13 shoes = m:new_item{t_size = 13, x_init = 96, y_init = 288, x_size = 160, y_size = 120}
14
15 ----- media items mapped by ports
16 m:init({animation})
17
18 ----- Link lIcon (onBegin segIcon -> start icon)
19 m:place({segIcon}, {icon, shoes})
20 m:conditional(icon:BEG(), {segIcon:BEG()})
21
22 ----- Link lBegingShoes (onSelection[RED] icon -> stop icon; start shoes)
23 m:place({icon.i_selec[1]}, {shoes})
24 m:conditional(icon:END(), {icon.i_selec[1]:BEG()})
25 m:conditional(shoes:BEG(), {icon.i_selec[1]:BEG()})
26
27 ----- Check the document
28 print(m:check())
29 m:end_document()
```

Listing 5.3: Document representation using SMT Validator

As presented in Listing 5.3, the tool represents the concepts presented in Chapter 4 for the *SHM* model in Lua. Each document element is modeled as a Lua table (using Lua's object orientation) and each *SHM* relation is modeled as a function, having as

parameter the fragments that participate in the given relation.

Class *model* is responsible for the creation of the representation of the document as a whole and triggering the solver execution and getting the solver response. As a use case, given that the solver provided a valuation to the constants representing the document elements, this information is presented back to the user in a graphical representation.

It is worth highlighting that the implementation of the SMT Validator in Lua brings the possibility of integrating it with an NCL player (as a module of the NCL document itself), thus providing validation of a document at *execution* time. An application of the validator with such a goal is discussed in Section 5.4.

5.4 SMT Implementation in Practice

In [Amorim et al. 2015, Amorim et al. 2016], they presented an approach for defining the spatial layout of NCL document through templates. It defines the XTemplate 4.0 language and processor for defining and processing NCL spatio-temporal templates. The main idea is enabling the author to indicate media items to be used in a given document, then the template processor automatically creates *regions* and *descriptors* necessary for declaring the position and size of such items.

An evolution of the approach presented in [Amorim et al. 2016] is the definition of the so-called dynamic layouts. In such an approach, the spatial layout of a document is defined through spatial constraints. The development of dynamic layouts is build upon the *SMT* approach presented in Chapter 4.4.

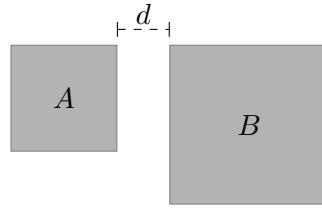
Focusing on improving the declaration of dynamic layouts, we defined additional spatial relations for providing the alignment/distribution of media items. The following relations are defined: *sep-x*, *align-x*, *dist-x* (and respectively *-y*) and *flow*. Suffix *-x* indicates that a given operator is defined in the *x* axis and suffix *-y* for the *y* axis.

Relation *sep* defines a separation among regions in a given axis. An example of relation *sep* is depicted in Figure 5.10. In *SMT*, relation *sep* will define the spacing between the border of two elements with constraints as follows.

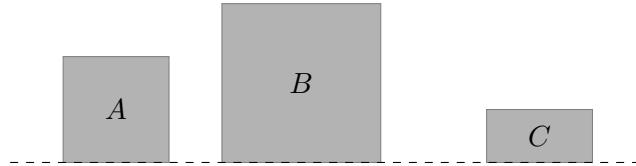
$$a \text{ sep-}x(d) b \equiv b:x^{beg} = a:x^{end} + d \quad (5.1)$$

$$a \text{ sep-}y(d) b \equiv b:y^{beg} = a:y^{end} + d \quad (5.2)$$

Relation *align* aligns regions in a given axis, according to one of three endpoints: *init*,

Figure 5.10: Relation $A \text{ sep-}x(d) B$

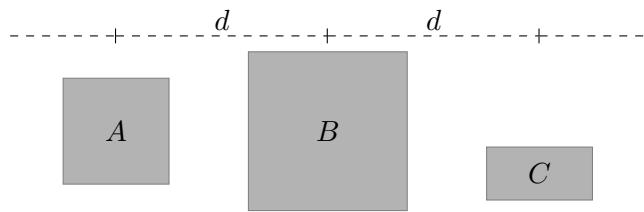
middle or *end*. The endpoint to be used is indicated in the first parameter of the relation. An example of relation *align* is depicted in Figure 5.11. In *SMT*, relation *align* will set

Figure 5.11: Relation $\text{align-}y(\text{end}, A, B, C)$

to equal the constants representing the same endpoint in all of its elements. For example, relation $\text{align-}y(\text{end}, a, b, c)$ is represented as follows.

$$a:y^{\text{end}} = b:y^{\text{end}} \wedge b:y^{\text{end}} = c:y^{\text{end}}$$

Relation *dist* distributes regions in a given axis with a same spacing among them. The spacing is defined related to one of the three endpoints the same way as done for relation *align*. An example of relation *dist* is depicted in Figure 5.12. In *SMT*, relation

Figure 5.12: Relation $\text{dist-}x(\text{middle}, d, A, B, C)$

dist is represented in a similar way to what is done for representing relation *align*. For example, relation $\text{dist-}x(\text{middle}, d, a, b, c)$ is represented as follows.

$$b:x^{\text{mid}} = a:x^{\text{mid}} + d \wedge c:x^{\text{mid}} = b:x^{\text{mid}} + d$$

Finally, relation *flow* defines a region inside which regions are organized in a flow. At a given moment in the document presentation, the regions to be considered inside the flow

are the ones being presented. That means that the arrangement of flow internal regions may change according to the begin/end of fragments execution. Figure 5.13 depicts an example of relation *flow* of media items *A*, *B* and *C*, and the rearrangement of the layout when media item *B* ends its presentation.

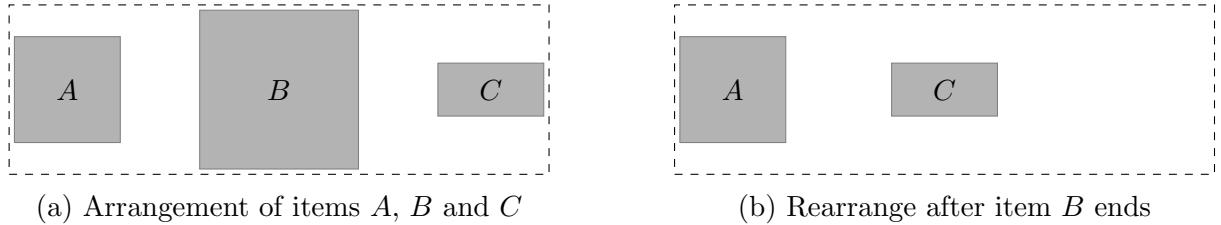


Figure 5.13: Relation $\text{flow}(A, B, C)$

In *SMT*, relation *flow* is defined with constraints specifying that a given element at position l in the flow is either at the right or above its predecessor $l - 1$, given that the latter is being presented. Otherwise the same rule is enforced between the element at position l and $l - 2$.

Although some overlap arises from both RCC and alignment/distribution relations, they are important for enabling a more concise representation of the presentation spatial layout, and thus, making the declaration of dynamic layouts easier.

5.5 Use Case

5.5.1 Multimedia Document Example

We present in this section a multimedia document to exemplify how our validation approach is applied. The example presents touristic information about the state of Rio de Janeiro in Brazil. It presents the state map along with pins in different cities: Rio de Janeiro, Niterói, Teresópolis and Paraty. For each city, a promotional video and additional photos are presented whenever the viewer clicks in the correspondent pin. Figure 5.14 illustrates some screens of the document presentation.

The presentation begins by the presenting Rio de Janeiro state map with a fade animation. After the fading is done, the pins indicating main cities start to appear, one after the other, also with a fade animation (Figures 5.14a and 5.14b). From this point on, the presentation waits for viewer interaction.

Whenever the viewer clicks on a pin, some information about the city is presented,

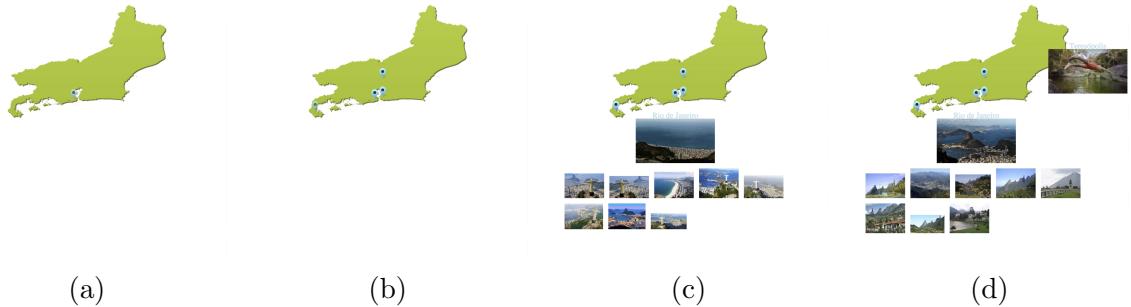


Figure 5.14: Use case document presentation

such as: the city name, its promotional video and additional photos about the city. The city name is presented above the video center-aligned with it and the additional photos are all presented below the video using the whole screen width (Figures 5.14c and 5.14d). The video is aligned with its correspondent pin (the videos for Rio, Niterói and Paraty above the pin and aligned horizontally with it, and the video for Teresópolis on the right side of the pin and aligned vertically with it) unless it would cause an overlapping with another video. When an overlapping occurs, the video will be shifted to the right until no overlapping is done.

The additional photos presented for each city are gathered from Google Images during the presentation. It is done by querying with Google Web Search API the city name and getting the first eight photos returned by the query. Thus, the photos will change as the search in Google yields new results⁴.

It is also possible for the presentation to behave differently according to the viewer context in two ways. (i) If the viewer device is not capable of presenting more than one video at a time, when clicking on a city pin, any other video being presented is stopped before starting the presentation of the current city video. (ii) For smaller device screens, less than 900 pixels width, video objects are all presented in the same location, under the map, and one at a time.

5.5.2 Representation in \mathcal{SHM}

Twenty two fragments are used to represent such a document inside \mathcal{SHM} , five for each city and two for the map. The naming scheme for fragments related to city content is as follows: the first letter of city name ([R]io de Janeiro, [N]iterói, [T]eresópolis and [P]araty) plus a letter representing the content type (p for the pin, v for the video, n for

⁴A version of this presentation using HTML5 and Web Animations is available online at <https://rawgit.com/joeldossantos/usecase/master/tourism.html>

the name and **f** for the additional photos). Moreover, an additional letter **a** indicates a fragment representing the animation of a given content. For example, the map animation is represented by fragment *Ma*, while *Rp* represents the pin of Rio de Janeiro.

Equation 5.3 presents the set of fragments for the use case.

$$\left\{ \begin{array}{l} M, Ma, Rp, Rpa, Rn, Rv, Rf, Np, Npa, Nn, Nv, Nf, \\ Tp, Tpa, Tn, Tv, Tf, Pp, Ppa, Pn, Pv, Pf \end{array} \right\} \quad (5.3)$$

Three properties represent the viewer context: two for the screen size (*SWidth* and *SHeight*) and one for the ability to present multiple videos at a time or not (*Multi*).

Map *M* is presented centralized on the exhibition device and its size is defined according to the screen size. The following rules specify the spatial layout for *M*.

$$\begin{aligned} M.width &= \begin{cases} SWidth * 4/5 & \text{if } SWidth < 900 \\ 600 & \text{if } SWidth \geq 900 \end{cases} \\ M.height &= \begin{cases} SWidth * 43/75 & \text{if } SWidth < 900 \\ 430 & \text{if } SWidth \geq 900 \end{cases} \\ M \ tpp(90) \ canvas \end{aligned} \quad (5.4)$$

Each city pin is placed in relation to *M* as follows. For simplicity, rules that are applied to all pins are defined as **p*.

```
*p.width = 40
*p.height = 40

Rp ntpp(236, 126f) M
Np ntpp(247, 103f) M
Tp ntpp(212, 47f) M
Pp ntpp(206, 330f) M
```

$$f = \begin{cases} SWidth/750 & \text{if } SWidth < 900 \\ 1 & \text{if } SWidth \geq 900 \end{cases} \quad (5.5)$$

Pin positions take into account the map size. Thus, their distance to M are calculated according to factor f . According to Equation 5.5, the pin for the city of Rio de Janeiro, for example, has 40 pixels width and height. It is located inside the map (non-tangential proper part relation), and at an angle of 236 degrees and a distance of $126 \times f$ pixels from the center of the map.

Position for each city video, name and additional photos are defined as follows, where $*v$, $*n$ and $*f$ are used for representing rules that are applied over all video objects, city names and additional photos, respectively.

$$\begin{aligned} *v.width &= \begin{cases} SWidth/2 & \text{if } SWidth < 900 \\ 300 & \text{if } SWidth \geq 900 \end{cases} \\ *n \ econ(90) *v \\ \\ *f.width &= SWidth * 9/10 \\ *f.top &= *v.bottom + 10 \\ *f \ tpp(270) \ canvas \end{aligned} \quad (5.6)$$

According to Equation 5.6, a video width is half of the screen width in case it is smaller than 900 pixels and fixed at 300 pixels otherwise. The name of each city is presented above the video and externally connected to it (*econ* relation). Additional photos have 90% of the screen width and are presented 10 pixels below its video.

Whenever the screen size is smaller than 900 pixels, position for videos is defined as follows.

$$\begin{aligned} *v \ dcon(270) \ M \\ *v \ sep-y(40) \ M \end{aligned} \quad (5.7)$$

where each video is presented 40 pixels below the map.

On the other hand, whenever the screen size is bigger or equal than 900 pixels, positions for videos are defined as follows.

$$\begin{aligned} &Tv \ dcon(0) \ Tp \\ &Tv \ sep-x(10) \ M \quad \vee \quad align-x(end, Tv, canvas) \\ \\ &Pv \ dcon(270) \ Pp \\ &Pv \ sep-y(40) \ M \\ \\ &Rv \ dcon(S) \ Rp \wedge Rv \ dcon(L, 310) \ Pv \\ &Rv \ sep-y(10) \ M \\ \\ &Nv \ dcon(S) \ Np \wedge Nv \ dcon(L, 310) \ Pv \quad \vee \\ &\quad Nv \ dcon(SW) \ Np \wedge Nv \ dcon(L, 310) \ Rv \\ &Nv \ sep-y(10) \ M \end{aligned} \quad (5.8)$$

According to Equation 5.8, the video for the city of Rio de Janeiro, for example, is located 10 pixels below the map and it is presented below the city pin and 10 pixels on the right side of the video for the city of Paraty, in case the latter is presented. It is important to notice, that either videos Rv and Nv may be shifted to the right whenever video Pv or Rv is being played.

In the temporal axis, the presentation of the map and the pins are done in a given order and with a fade animation, as specified as follows.

```

Ma starts M
*pa starts *p
Ma meets Rpa
Rpa meets Npa
Npa meets Tpa
Tpa meets Ppa

```

(5.9)

The map is the first to be presented together with its animation. After the map animation ends the animation for the pins of Rio, Niterói, Teresópolis and Paraty are executed, in this sequence.

Whenever a pin is selected, its correspondent video is played. Such behavior is specified by the following causal relation

$$*ps \rightarrow *v^-$$
(5.10)

where $*ps$ represents the selection of a pin and $*v^-$ the initial endpoint of video $*v$.

Whenever a video is presented, the city name and additional photos are presented as well. The additional photos to be presented in a given instant are the ones related to the last selected pin. Such behavior is specified by the following rules, where superscripts $-$ and $+$ represent the initial and final endpoint of a fragment, respectively.

```

*n equals *v
*f^- \rightarrow *v^-
 $Rf^+ \rightarrow \{Rv^+, Nf^-, Tf^-, Pf^-\}$ 
 $Nf^+ \rightarrow \{Nv^+, Rf^-, Tf^-, Pf^-\}$ 
 $Tf^+ \rightarrow \{Tv^+, Rf^-, Nf^-, Pf^-\}$ 
 $Pf^+ \rightarrow \{Pv^+, Rf^-, Nf^-, Tf^-\}$ 

```

(5.11)

According to Equation 5.11, additional photos are presented one at a time. They start their presentation together with its associated video and can be presented until the end of the video or until additional photos for another video start playing.

Up to now, video objects may be presented at the same time. Whenever the screen size is smaller than 900 pixels or property *Multi* is false, to ensure that videos are not presented at the same time, the following rules are issued, where superscript s represents the duration of a fragment.

$$\begin{aligned} Rv^+ &\rightarrow \{Rv^-, Rv^s, Nv^-, Tv^-, Pv^-\} \\ Nv^+ &\rightarrow \{Nv^-, Nv^s, Rv^-, Tv^-, Pv^-\} \\ Tv^+ &\rightarrow \{Tv^-, Tv^s, Rv^-, Nv^-, Pv^-\} \\ Pv^+ &\rightarrow \{Pv^-, Pv^s, Rv^-, Nv^-, Tv^-\} \end{aligned} \tag{5.12}$$

5.5.3 Example Validation

Analyzing this presentation according to the life cycle presented in Figure 2.2, it is possible to see that, at the *Interpretation* step, the presentation is adapted according to the viewer context, i.e., its screen size and possibility to present multiple videos together. At the *Scheduling* step, the presentation layout is changed due to viewer interaction and also the content retrieved from the web search. The *Dynamic* step represents the search for new images according to the city chosen.

The representation $\mathcal{SHM}(d)$ of the use case document in model \mathcal{SHM} is given by Equations 5.3-5.12. By the composition $\mathcal{SHM}(d) \cap \mathcal{P}$, where \mathcal{P} is a set of properties, we are able to validate d against \mathcal{P} . To illustrate our validation approach we present in the following paragraphs three validation use cases.

Use case 1: At *Interpretation*, the author creates a property stating that no video should be overlapped by additional photos. In this case, \mathcal{P} represents the author-defined property

$$\begin{aligned} (Rv \text{ dcon } Nf \wedge Rv \text{ dcon } Tf \wedge Rv \text{ dcon } Pf) \wedge \\ (Nv \text{ dcon } Rf \wedge Nv \text{ dcon } Tf \wedge Nv \text{ dcon } Pf) \wedge \\ (Tv \text{ dcon } Rf \wedge Tv \text{ dcon } Nf \wedge Tv \text{ dcon } Pf) \wedge \dots \wedge \\ (Pv \text{ dcon } Rf \wedge Pv \text{ dcon } Nf \wedge Pv \text{ dcon } Tf) \end{aligned} \tag{5.13}$$

If the composition of $\mathcal{SHM}(d)$ and \mathcal{P} holds, then the author is able to verify that the presentation layout is as it expects. However, as discussed in Section 2.1.2 validation at this point has a *preventive* role since parts of the document may be edited during

execution. In this case, such a property may be “carried along” the document through its life cycle. At the following steps (i.e., *Interpretation* and *Scheduling*), once the document is edited, such properties may be reenforced in order to maintain the document consistency.

Use case 2: At *Interpretation*, the document is adapted to the viewer context. In this case, \mathcal{P} represents the viewer context, which for the use case would be the triple $\langle SWidth, SHeight, Multi \rangle$. Thus, the document is adapted according to the screen size or the capability of presenting multiple videos. The validation performed at this step will result in a model representing the presentation adapted to the user context. We represent it by $\mathcal{SHM}_C(d)$.

Use case 3: At *Scheduling*, the document layout is (re)generated according to the images retrieved from the web search. In this case, the adapted document is validated against the current presentation time and the events, such as viewer interaction, that occurred up to the current time as follows.

$$\mathcal{SHM}_C(d) \cap T = t \cap \{e_1, \dots, e_n\}$$

where T is the global presentation time (see Section 4.1.3), t is the current presentation time and $\{e_1, \dots, e_n\}$ a list of events that occurred up to time t . By performing validation for a given presentation time, it is possible to (re)calculate the spatial layout of the presentation according to fragments being presented at time t .

5.6 Closing Remarks

Section 5.5 presented a use case document designed to represent real world multimedia documents. It involves presenting different types of media items (image, video, text) representing items either with or without an inherent duration. Media items are presented with or without animation and their spatial layout varies according to the items being presented at a time. Moreover, the use case document declares a temporal layout where media items are presented according to both deterministic and non-deterministic events, such as viewer interaction. Given such characteristics, the use case document represents a meaningful set of multimedia documents.

Equations 5.3-5.12 present the representation of use case document d in model \mathcal{SHM} as $\mathcal{SHM}(d)$. Given that d is a meaningful example of document and that \mathcal{SHM} is expressive enough to represent it, it is possible to infer that \mathcal{SHM} is expressive enough for representing real world multimedia documents.

In the following sections we discuss the choice for a given validation technique in Section 5.6.1 and limitations in the implementations presented in this chapter.

5.6.1 Validation Technique Choice

The validation approach presented in this work provides two techniques for performing document validation (see Figure 4.12), they are *RWT* and *SMT*. Each technique has the ability of representing a given aspect of \mathcal{SHM} . *RWT* focuses on event occurrences and the document state throughout its presentation and *SMT* focuses on numeric dependencies among intervals (or regions).

RWT works upon the complete representation of d , generating all states of a presentation during its execution. Besides, due to its event-based nature, it allows for investigating properties that are related to fragments that can occur several times during the presentation. For example, we could use *RWT* in our use case for investigating if from any given state of the presentation we are able to go back to a state where just the map and the pins are presented. Such a property is represented using *RWT* by the following LTL formula.

$$p_{RWT} : G \left(F \left(\begin{array}{l} M.\text{pre.occuring} \wedge *p.\text{pre.occuring} \wedge \\ *v.\text{pre.sleeping} \wedge *f.\text{pre.sleeping} \wedge *n.\text{pre.sleeping} \end{array} \right) \right) \quad (5.14)$$

By model-checking p_{RWT} over $\mathcal{R}_{RWT}(d)$, we are able to verify that every state of the document has a future state where only the map and pins are presented. This is expected since all items related to a given video (city name and photos) and the video itself are presented for a finite interval.

RWT performs better than *SMT* in such kind of scenario. Since *SMT* represents fragments as intervals, for each occurrence $f_i, i \in [1, n]$ of a given fragment f along the presentation, one interval must be created. In the case where n is not known *a priori*, using *SMT* would be cumbersome. Moreover, representing such kind of property in *SMT* is not an easy task.

Different from *RWT*, *SMT* does not expect the complete representation of d , thus even when d represents a partial document, *SMT* can be used to validate it. Due to its numerical nature, *SMT* allows for investigating properties related to: (i) the duration of fragments, (ii) the relative position of event occurrences in time and (iii) the relative

position of regions in space. For example, we could use *SMT* in our use case for investigating if it is possible for two videos to start their presentation at the same time. Such a property is represented in *SMT*, for videos Rv and Nv , by the following formula.

$$p_{SMT} : Rv:t^{plc} \wedge Nv:t^{plc} \wedge Rv:t^{beg} = Nv:t^{beg} \quad (5.15)$$

By solving $\mathcal{SHM}(d) \cap p_{SMT}$ in the SMT solver, it is possible to see that such situation is only possible when the viewer selects the pin for both cities at the same time. Given that we enforce that two different selection events can not happen at the same time, such situation would not be possible.

SMT performs better than *RWT* in such kind of scenario. Since *RWT* represents the document presentation by its different states along execution, different traces are provided by the interleaving of rules *step* and rules representing viewer interaction as discussed in Section 4.3.3, simulating viewer interaction at different instants of a fragment presentation. Generating every possible combination of viewer interaction for finding a document state where both Rv and Nv start at the same time would lead to an explosion in the number of states created by *RWT*.

5.6.2 Tool Limitations

This chapter presented two tools that partially implement the validation approach proposed in this work. The general architecture of our proposal was depicted in Figure 4.12. We recall it here to improve this section readability.

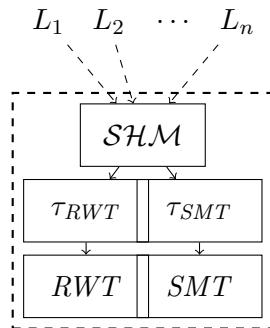


Figure 5.15: Validation approach architecture

The idea depicted in Figure 4.12 is to provide the translation from a given multimedia language L_i to \mathcal{SHM} and from the latter to either *RWT* or *SMT* in order to provide document validation. Currently, our validation tools implement only part of the proposed architecture.

aNaa implements the left-hand side of Figure 4.12. It provides a mapping from NCL to *RWT*. Such mapping was discussed in Chapter 4.3 and Section 5.1.1. SMT Validator, on the other hand, partially implements the right-hand side of Figure 4.12. It provides a mapping from \mathcal{SHM} (or at least from the Lua table representing it) to *SMT*. Although we have already explored mappings from NCL to \mathcal{SHM} , it is not yet automated.

A limitation regarding the implementation, therefore, is the lack of integration of both approaches. A future work in this direction is to provide a representation of \mathcal{SHM} in either Java and/or Lua so that we can translate a given document to it and then to either *RWT* or *SMT*.

Although \mathcal{SHM} is intended to be used for representing different multimedia languages, we currently provide tools that only consider NCL documents. As it was presented in Section 5.1.1, we have extended the equational theory \mathcal{E} for the SMIL language. It is available as a Maude module so that we can provide validation for SMIL documents using *RWT*. However, we have not integrated it with aNaa such that the tool could work with both NCL and SMIL documents. Such integration was also left as future work.

A limitation of the tools that implement the validation approach here proposed is that they consider media fragments only as a subpart of a media item in time. A future work is to refine those tools to consider also fragments as a subpart of a media item in space.

The validation approach presented in this work is intended to be used in different steps of a multimedia document life cycle. However, it is currently implemented for working only at the *authoring/generation* phase. A future work is to provide an implementation of the validation proposed here so that it can be used in different parts of a document life cycle.

Chapter 6

Conclusion

A multimedia presentation may change and evolve during its life cycle. From *authoring/generation* to *execution*, a document is created (or queried/requested), instantiated, adapted and may have its structure changed. Thus, it is important to maintain the consistency of a presentation along its life cycle. The research presented in this work fits in the main idea of maintaining the consistency of a document along its life cycle. It presented an approach for validating documents. Such an approach is intended to be used in different steps of a document life cycle.

The proposed validation relies on a general model \mathcal{SHM} for representing a document d . From \mathcal{SHM} , two validation techniques are possible. They are called *RWT* and *SMT*.

The first captures the behavior of a document d in terms of its state throughout its execution. This is achieved with use of the rewrite theory *RWT*, which induces a transition system $\mathcal{S}_{RWT}(d) = (S, \rightarrow)$, where each state in S represents the state of d as a whole in a given moment of its execution, and each transition in \rightarrow models the user interaction or a time lapse. Validation in *RWT* is performed by means of model-checking.

The second captures the behavior of a document d in terms of intervals and event occurrences. This is achieved with use of an *SMT* solver, where *SMT* formulas are used to represent fragment intervals and regions together with relations among them. Validation in *SMT* is performed by solving the resulting formulas.

Due to the different characteristics of *RWT* and *SMT*, each technique complements the other in terms of the expressiveness of \mathcal{SHM} and the kind of properties to be investigated. A discussion about the choice for each kind of validation is presented in Section 5.6.1.

6.1 Contributions

In this section we recall the main contributions of this thesis. They are presented in the following paragraphs.

One contribution of this work is the definition of the life cycle of multimedia documents. As it was presented in Chapter 2, this life cycle was inspired in related work so that different works about multimedia documents could fit in its steps.

The definition of a life cycle for multimedia documents makes it possible to investigate the validation requirements at different steps. The different requirements are also presented in Chapter 2. The idea is to be able to perform document validation at different steps in order to maintain the document consistent in the whole life cycle.

Targeting on validating multimedia documents at different steps in its life cycle, the main contribution of this work is to propose a validation approach based on a general model called *SHM*. The *SHM* model was presented in Chapter 4. The idea of *SHM* is to provide a formal representation of the behavior of multimedia documents.

Besides the definition of the *SHM* model, the validation approach presented in this work uses two validation techniques for providing document validation. Each technique focuses on a different characteristic and type of properties to be validated. The first technique, presented in Section 4.3 validates a multimedia document based on its state along its execution. The second technique, presented in Section 4.4 validates a multimedia document based on constraints. Additional contributions from using those two techniques is the formalization of a multimedia document as rewriting logic theory and as a set of constraints, respectively.

In order to be able to use the different techniques presented here, both techniques were implemented in validation tools. Each tool partially implements one of the validation technique as discussed in Chapter 5. The same Chapter also presents a discussion about which technique is more suited for different steps of the document life cycle.

Finally, another contribution of this work, related to the implementation of the techniques presented here, is its use for finding and reporting errors in the reference implementation of middleware Ginga-NCL. Thus being able to provide feedback to the group responsible for the middleware implementation.

Those contributions were presented during this thesis. In the following section we discuss some future and ongoing works.

6.2 Future Works

As presented in Section 5.6.1, two complementary techniques (*RWT* and *SMT*) are provided for performing validation over $\mathcal{SHM}(d)$. Currently, at a given step of the document life cycle, the choice for a validation technique to be used is left for the author. A future work is to investigate a way to automatically choose one of them based on the characteristics of the document and the properties to be validated.

As discussed in Chapter 2, validation at a given step should preserve a set of properties representing what we call author guidelines. Our definition of a document life cycle, together with validation requirements for each step lays the base for providing a validation approach with *intention preservation*. A future work is to provide a way for the author to specify such guidelines.

One requirement for providing validation after the *authoring/generation* phase was to have some kind of automatic “error correction” so that it would always yield an executable presentation. The idea behind such automatic correction is to avoid the interruption of the presentation. As presented in Chapter 5 both tools that implement the validation approach proposed here only provide feedback to the author for performing document validation. A future work is to investigate approaches such as the one presented in [Laborie et al. 2011] for providing such automatic correction.

It is worth highlighting that, providing both the possibility of defining author guidelines and an automatic “error correction” we would be able to tackle the main idea of maintaining the consistency of a document along its life cycle.

Given a property to be validated over a document, both tools presented in Chapter 5 present back to the author just if the property is valid or not. A future work is to present back to the author more intuitive messages. One example is to use the information in the trace presented as an answer by the Maude model checker to build an animation or a sequence of NCL link activations up to the point of failure.

As discussed in Chapter 5, both validation tools consider media fragments as a subpart of a media item in time. A future work is to consider also media fragments as a subpart of a media item in space as well. A side effect of such refinement is the possibility of, by the composition of fragment regions, be able to define more complex forms for representing media items.

The validation approach presented here models a few characteristics (such as screen

size) of the execution platform of a multimedia document. A future work is to represent other platform characteristics such as failure rates so that validation may present, for example, a percentage of execution where the document will be valid.

The current implementation of *SMT* uses Lua and the SMT solver Yices2 [Dutertre 2014]. As an ongoing work we are integrating *SHM* with the NCL language to provide dynamic layout adaptation of NCL presentations. A future work is to integrate *RWT* as well so to further investigate its use in such a scenario. Another future work is to update *SMT* for using the Z3 SMT solver [Moura and Bjørner 2008].

In the document life cycle presented in Chapter 2 templates may be used for the creation of multimedia documents. The use of templates can be seen as a way to support authors easing the creation of documents or even as a way for providing the automatic generation of documents. Given the existence of a *Template Base*, it is important to guarantee that templates may not introduce inconsistencies in its instances. A future work is to extend the validation approach proposed here for providing template validation.

References

- [ABNT 2011] ABNT. *Digital terrestrial television - Data coding and transmission specification for digital broadcasting - Part 2: Ginga-NCL for fixed and mobile receivers - XML application language for application coding*. 2011. ABNT NBR 15606-2:2011 standard.
- [Allen 1983] ALLEN, J. F. Maintaining knowledge about temporal intervals. *Communications of the ACM*, ACM, v. 26, n. 11, p. 832–843, 1983.
- [Amorim et al. 2015] AMORIM, G. F.; DOS SANTOS, J. A. F.; MUCHALUAT-SAADE, D. C. Adaptive layouts and nesting templates for hypermedia composite templates. In: *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2015. (WebMedia '15), p. 189–196. ISBN 978-1-4503-3959-9.
- [Amorim et al. 2016] AMORIM, G. F.; DOS SANTOS, J. A. F.; MUCHALUAT-SAADE, D. C. Xtemplate 4.0: Providing adaptive layouts and nested templates for hypermedia documents. In: *Proceedings of the 22nd International Conference on Multimedia Modeling*. [S.l.: s.n.], 2016.
- [Araújo et al. 2008] ARAÚJO, E. C.; AZEVEDO, R. G. A.; NETO, C. S. S. Ncl-validator: um processo para validação sintática e semântica de documentos multimídia ncl. In: *Jornada de Informática do Maranhão*. [S.l.: s.n.], 2008. In portuguese.
- [ARIB 2014] ARIB. *Data Coding and Transmission Specification for Digital Broadcasting*. 2014. Standard B24 v6.1.
- [ATSC 2009] ATSC. *Advanced Common Application Platform (ACAP)*. 2009. Document A/101A.
- [Azevedo et al. 2009] AZEVEDO, R. G. A.; TEIXEIRA, M. M.; NETO, C. S. S. Ncl eclipse: Ambiente integrado para o desenvolvimento de aplicações para tv digital interativa em nested context language. In: *Brazilian Symposium on Computer Networks*. [S.l.]: SBRC, 2009. In portuguese.
- [Baral and Gelfond 1994] BARAL, C.; GELFOND, M. Logic programming and knowledge representation. *The Journal of Logic Programming*, Elsevier, v. 19, p. 73–148, 1994.
- [Barrett et al. 2009] BARRETT, C. W.; SEBASTIANI, R.; SESHIA, S. A.; TINELLI, C. Satisfiability modulo theories. *Handbook of satisfiability*, v. 185, p. 825–885, 2009.
- [Batista 2015] BATISTA, D. T. *Property Editor for the Verification of Temporal Relations in Hypermedia Documents*. Computer Science Bachelor Monography — Universidade Federal Fluminense, 2015. In Portuguese.

- [Belouaer and Maris 2012] BELOUAER, L.; MARIS, F. Smt spatio-temporal planning. In: *ICAPS 2012 Workshop on Constraint Satisfaction Techniques for Planning and Scheduling Problems (COPLAS 2012)*. [S.l.: s.n.], 2012. p. 6–15.
- [Bertino et al. 2005] BERTINO, E.; FERRARI, E.; PEREGO, A.; SANTI, D. A constraint-based approach for the authoring of multi-topic multimedia presentations. In: *IEEE International Conference on Multimedia and Expo*. Amsterdam, Netherlands: IEEE Computer Society, 2005. p. 578–581.
- [Blakowski and Steinmetz 1996] BLAKOWSKI, G.; STEINMETZ, R. A media synchronization survey: Reference model, specification and case studies. *Journal on Selected Areas in Communications*, IEEE, v. 14, n. 1, p. 5–35, January 1996.
- [Boll 2001] BOLL, S. *ZYX - Towards flexible multimedia document models for reuse and adaptation*. Ph.D. Thesis — Vienna University of Technology, 2001.
- [Bouyakoub and Belkhir 2008] BOUYAKOUB, S.; BELKHIR, A. H-smil-net: A hierarchical petri net model for smil documents. In: *Proceedings of the Tenth International Conference on Computer Modeling and Simulation*. Washington, DC, USA: IEEE Computer Society, 2008. (UKSIM '08), p. 106–111. ISBN 978-0-7695-3114-4.
- [Bouyakoub and Belkhir 2011] BOUYAKOUB, S.; BELKHIR, A. Smil builder: An incremental authoring tool for smil documents. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, ACM, New York, NY, USA, v. 7, n. 1, p. 2:1–2:30, fev. 2011. ISSN 1551-6857.
- [Bulterman et al. 2013] BULTERMAN, D.; CESAR, P.; GUIMARAES, R. L. Socially-aware multimedia authoring: Past, present, and future. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, ACM, v. 9, n. 1s, p. 35, 2013.
- [Clavel et al. 2007] CLAVEL, M.; EKER, S.; DURÁN, F.; LINCOLN, P.; MARTÍ-OLIET, N.; MESEGUER, J. *All about Maude - A High-performance Logical Framework: how to Specify, Program, and Verify Systems in Rewriting Logic*. [S.l.]: Springer-Verlag New York Inc, 2007.
- [Damasceno et al. 2014] DAMASCENO, J. R.; DOS SANTOS, J. A. F.; MUCHALUAT-SAADE, D. C. Editec - a graphical editor for hypermedia composite templates. *Multimedia Tools and Applications*, Springer US, v. 70, n. 2, p. 1167–1198, 2014. ISSN 1380-7501.
- [dos Santos 2012] DOS SANTOS, J. A. F. *Multimedia and hypermedia document validation and verification using a model-driven approach*. Dissertation (Masters) — UFF, 2012.
- [dos Santos et al. 2012a] DOS SANTOS, J. A. F.; BRAGA, C.; MUCHALUAT-SAADE, D. C. A model-driven approach for the analysis of multimedia document. In: *SLE (Doctoral Symposium)*. [S.l.: s.n.], 2012. p. 37–44.
- [dos Santos et al. 2013a] DOS SANTOS, J. A. F.; BRAGA, C.; MUCHALUAT-SAADE, D. C. Automating the analysis of ncl documents with a model-driven approach. In: *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2013. (WebMedia '13), p. 193–200. ISBN 978-1-4503-2559-2.

- [dos Santos et al. 2013b] DOS SANTOS, J. A. F.; BRAGA, C.; MUCHALUAT-SAADE, D. C. An executable semantics for a multimedia authoring language. In: *Formal Methods: Foundations and Applications*. Brasília, Brazil: Springer, 2013. p. 67–82.
- [dos Santos et al. 2015a] DOS SANTOS, J. A. F.; BRAGA, C.; MUCHALUAT-SAADE, D. C. A rewriting logic semantics for ncl. *Science of Computer Programming*, v. 107–108, n. 0, p. 64 – 92, 2015. ISSN 0167-6423.
- [dos Santos et al. 2015b] DOS SANTOS, J. A. F.; BRAGA, C.; MUCHALUAT-SAADE, D. C.; ROISIN, C.; LAYAÏDA, N. Spatio-temporal validation of multimedia documents. In: *Proceedings of the 2015 ACM Symposium on Document Engineering*. New York, NY, USA: ACM, 2015. (DocEng '15).
- [dos Santos et al. 2012b] DOS SANTOS, J. A. F.; SILVA, J. V.; VASCONCELOS, R. R.; SCHAU, W.; WERNER, C.; MUCHALUAT-SAADE, D. C. ana: Api for ncl authoring. In: *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web - Workshop of Tools and Applications*. [S.l.: s.n.], 2012.
- [Durán et al. 2011] DURÁN, F.; ROCHA, C.; ÁLVAREZ, J. M. Towards a maude formal environment. In: *Formal Modeling: Actors, Open Systems, Biological Systems*. [S.l.]: Springer, 2011. p. 329–351.
- [Dutertre 2014] DUTERTRE, B. Yices 2.2. In: BIERE, A.; BLOEM, R. (Ed.). *Computer-Aided Verification (CAV'2014)*. [S.l.]: Springer, 2014. (Lecture Notes in Computer Science, v. 8559), p. 737–744.
- [Elias et al. 2006] ELIAS, S.; EASWARAKUMAR, K.; CHBEIR, R. Dynamic consistency checking for temporal and spatial relations in multimedia presentations. In: *Proceedings of the 2006 ACM symposium on Applied computing*. Dijon, France: ACM, 2006. p. 1380–1384.
- [ETSI 2008] ETSI. *Digital Video Broadcasting (DVB); Multimedia Home Platform (MHP) Specification 1.0.3*. 2008. ETSI ES 201 812 v1.1.2.
- [Felix 2004] FELIX, M. F. *Formal Analysis of Software Models Oriented by Architectural Abstractions*. Ph.D. Thesis — Pontifícia Universidade Católica do Rio de Janeiro, 2004. In Portuguese.
- [Furuta and Stotts 2001] FURUTA, R.; STOTTS, P. D. Trellis: a formally-defined hyper-textual basis for integrating task and information. *Coordination Theory and Collaboration Technology*, Lawrence Erlbaum Associates, p. 341–367, 2001.
- [Gaggi and Bossi 2011] GAGGI, O.; BOSSI, A. Analysis and verification of smil documents. *Multimedia Systems*, v. 17, n. 6, p. 487–506, April 2011.
- [Guedes et al. 2015] GUEDES, A. L. V.; AZEVEDO, R. G. A.; MORENO, M. F.; SOARES, L. F. G. Specification of multimodal interactions in ncl. In: *Proceedings of the 21st Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2015. (WebMedia '15), p. 181–187. ISBN 978-1-4503-3959-9.
- [Hardman 1998] HARDMAN, H. L. *Modeling and Authoring Hypermedia Documents*. Ph.D. Thesis — Universität Amsterdam, 1998.

- [Honorato 2010] HONORATO, G. S. C. *NCL-Inspector: A tool for the inspection of NCL applications*. Dissertation (Masters) — Pontifícia Universidade Católica do Rio de Janeiro, March 2010. In portuguese.
- [Honorato and Barbosa 2010] HONORATO, G. S. C.; BARBOSA, S. D. J. Ncl-inspector: Towards improving ncl code. In: *Proceedings of the 2010 ACM Symposium on Applied Computing*. [S.l.]: ACM, 2010. p. 1946–1947.
- [ISO/IEC 2001] ISO/IEC. *Multimedia content description interface*. 2001. ISO/IEC 15938.
- [ISO/IEC 2005] ISO/IEC. *Information technology - Coding of audio-visual objects - Part 11: Scene description and application engine*. 2005. ISO/IEC 14496-11:2005.
- [ITU 2009] ITU. *Nested Context Language (NCL) and Ginga-NCL for IPTV services*. 2009. [Http://www.itu.int/rec/T-REC-H.761-200904-S](http://www.itu.int/rec/T-REC-H.761-200904-S). ITU-T Recommendation H.761.
- [Jourdan et al. 1998] JOURDAN, M.; LAYAÍDA, N.; ROISIN, C.; SABRY-ISMAIL, L.; TARDIF, L. Madeus, an authoring environment for interactive multimedia documents. In: *Proceedings of the 6th ACM International Conference on Multimedia*. [S.l.]: ACM, 1998. p. 267–272.
- [Júnior et al. 2012] JÚNIOR, D. P.; FARINES, J.-M.; KOLIVER, C. An approach to verify live ncl applications. In: *Proceedings of the 18th Brazilian Symposium on Multimedia and the Web*. New York, NY, USA: ACM, 2012. (WebMedia '12), p. 223–232. ISBN 978-1-4503-1706-1.
- [Júnior et al. 2014] JÚNIOR, D. P. et al. Verifying hypermedia applications by using an MDE approach. In: *System Analysis and Modeling: Models and Reusability*. [S.l.]: Springer International Publishing, 2014, (Lecture Notes in Computer Science, v. 8769). p. 174–189. ISBN 978-3-319-11742-3.
- [King et al. 2004] KING, P.; SCHMITZ, P.; THOMPSON, S. Behavioral reactivity and real time programming in xml: functional programming meets smil animation. In: ACM. *Proceedings of the 2004 ACM symposium on Document engineering*. [S.l.], 2004. p. 57–66.
- [Kirchner and Moreau 2001] KIRCHNER, H.; MOREAU, P. E. Promoting rewriting to a programming language: A compiler for non-deterministic rewrite programs in associative-commutative theories. *Journal of Functional Programming*, Cambridge University Press, v. 11, n. 2, p. 207–251, 2001.
- [Laborie et al. 2011] LABORIE, S.; EUZENAT, J.; LAYAÍDA, N. Semantic adaptation of multimedia documents. *Multimedia tools and applications*, Springer, v. 55, n. 3, p. 379–398, 2011.
- [Larsen et al. 1997] LARSEN, K. G.; PETTERSSON, P.; YI, W. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, Springer, v. 1, n. 1, p. 134–152, 1997.
- [Lemlouma and Layaïda 2004] LEMLOUMA, T.; LAYAÍDA, N. Context-aware adaptation for mobile devices. In: *IEEE International Conference on Mobile Data Management*. [S.l.: s.n.], 2004. p. 106–111.

- [Lemordant et al. 2013] LEMORDANT, J.; MICHEL, T.; RAZAFINAHAZO, M. Mixed reality browsers and pedestrian navigation in augmented cities. In: *The Graphical Web Conference*. San Francisco, United States: [s.n.], 2013.
- [Lima et al. 2010] LIMA, B.; AZEVEDO, R. G. A.; MORENO, M.; SOARES, L. F. G. Composer 3: Ambiente de autoria extensível, adaptável e multiplataforma. In: *Proceedings of the Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2010. In portuguese.
- [Lima and Soares 2013] LIMA, G. A. F.; SOARES, L. F. G. Two normal forms for link-connector pairs in ncl 3.0. In: *Proceedings of the 19th Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2013.
- [Meseguer 1992] MESEGUER, J. Conditional rewriting logic as a unified model of concurrency. *Theoretical computer science*, Elsevier, v. 96, n. 1, p. 73–155, 1992.
- [Meseguer 2012] MESEGUER, J. Twenty years of rewriting logic. *The Journal of Logic and Algebraic Programming*, Elsevier, v. 81, n. 7, p. 721–781, 2012.
- [Milner 1999] MILNER, R. *Communicating and mobile systems: the pi calculus*. [S.l.]: Cambridge university press, 1999.
- [Moura and Bjørner 2008] MOURA, L. D.; BJØRNER, N. Z3: An efficient smt solver. In: *Tools and Algorithms for the Construction and Analysis of Systems*. [S.l.]: Springer, 2008. p. 337–340.
- [Moura and Bjørner 2011] MOURA, L. D.; BJØRNER, N. Satisfiability modulo theories: Introduction and applications. *Communications of the ACM*, ACM, New York, NY, USA, v. 54, n. 9, p. 69–77, 2011. ISSN 0001-0782.
- [Muchaluat-Saade 2003] MUCHALUAT-SAADE, D. C. *Relations in Hypermedia Authoring Languages: Improving Reuse and Expressiveness*. Ph.D. Thesis — Pontifícia Universidade Católica do Rio de Janeiro, March 2003.
- [Muchaluat-Saade and Soares 2002] MUCHALUAT-SAADE, D. C.; SOARES, L. F. G. Xconnector & xtemplate: Improving the expressiveness and reuse in web authoring languages. *The New Review of Hypermedia and Multimedia Journal*, Taylor Graham, v. 8, n. 1, p. 139–169, 2002.
- [Na and Furuta 2001] NA, J.; FURUTA, R. Dynamic documents: authoring, browsing, and analysis using a high-level petri net-based hypermedia system. In: *Proceedings of the 2001 ACM Symposium on Document engineering*. [S.l.]: ACM, 2001. p. 38–47.
- [Neto et al. 2011] NETO, J. R. C.; SANTOS, R. C. M.; NETO, C. S. S.; TEIXEIRA, M. M. Método de validação estrutural e contextual de documentos ncl. In: *Proceedings of the 17th Brazilian Symposium on Multimedia and the Web*. [S.l.: s.n.], 2011. p. 1–8. In Portuguese.
- [Oliveira et al. 2001] OLIVEIRA, M. de; TURINE, M.; MASIERO, P. A statechart-based model for hypermedia applications. *ACM Transactions on Information Systems (TOIS)*, ACM, v. 19, n. 1, p. 52, 2001.

- [Pérez-Luque and Little 1996] PÉREZ-LUQUE, M. J.; LITTLE, T. D. C. A temporal reference framework for multimedia synchronization. *Journal on Selected Areas in Communications*, IEEE, v. 14, n. 1, p. 36–51, January 1996.
- [Pnueli 1977] PNUELI, A. The temporal logic of programs. In: IEEE. *18th Annual Symposium on Foundations of Computer Science*. Providence, USA, 1977. p. 46–57. ISSN 0272-5428.
- [Randell et al. 1992] RANDELL, D. A.; CUI, Z.; COHN, A. G. A spatial logic based on regions and connection. *Principles of Knowledge Representation and Reasoning*, p. 165–176, 1992.
- [Santos et al. 1998] SANTOS, C. A. S.; SOARES, L. F. G.; DE SOUZA, G. L.; COURTIAT, J. P. Design methodology and formal validation of hypermedia documents. In: *Proceedings of the sixth ACM International Conference on Multimedia*. Bristol, United Kingdom: ACM, 1998. p. 39–48.
- [Sarkis et al. 2014] SARKIS, M.; CONCOLATO, C.; DUFOURD, J.-C. The virtual splitter: Refactoring web applications for themultiscreen environment. In: *Proceedings of the 2014 ACM Symposium on Document Engineering*. New York, NY, USA: ACM, 2014. (DocEng '14), p. 139–142. ISBN 978-1-4503-2949-1.
- [Silva et al. 2013] SILVA, E. C. O.; DOS SANTOS, J. A. F.; MUCHALUAT-SAADE, D. C. Ncl4web: Translating ncl applications to html5 web pages. In: *Proceedings of the 2013 ACM Symposium on Document Engineering*. New York, NY, USA: ACM, 2013. (DocEng '13), p. 253–262. ISBN 978-1-4503-1789-4.
- [Soares and Barbosa 2012] SOARES, L. F. G.; BARBOSA, S. D. J. *Programming in NCL 3.0 - Developing Applications for Middleware Ginga*. 2nd. ed. [S.l.]: PUC-Rio, 2012. In Portuguese.
- [Soares et al. 2012] SOARES, L. F. G.; NETO, C. S. S.; SOUSA, J. G. Architecture for hypermedia dynamic applications with content and behavior constraints. In: ACM. *ACM symposium on Document engineering*. [S.l.], 2012. p. 217–226.
- [Soares and Rodrigues 2005] SOARES, L. F. G.; RODRIGUES, R. F. *Nested Context Model 3.0 Part 1 - NCM Core*. Rio de Janeiro, May 2005.
- [Soares et al. 2006] SOARES, L. F. G.; RODRIGUES, R. F.; COSTA, R. R.; MORENO, M. F. *Nested Context Language 3.0 Part 9 - NCL Live Editing Commands*. Rio de Janeiro, December 2006.
- [Soares et al. 2000] SOARES, L. F. G.; RODRIGUES, R. F.; MUCHALUAT-SAADE, D. C. Modeling, authoring and formatting hypermedia documents in the hyperprop system. *Multimedia Systems*, Springer-Verlag, v. 8, n. 2, p. 118–134, 2000.
- [Sun et al. 1998] SUN, C.; JIA, X.; ZHANG, Y.; YANG, Y.; CHEN, D. Achieving convergence, causality preservation, and intention preservation in real-time cooperative editing systems. *ACM Trans. Comput.-Hum. Interact.*, ACM, New York, NY, USA, v. 5, n. 1, p. 63–108, mar. 1998. ISSN 1073-0516.

- [Teixeira et al. 2012] TEIXEIRA, C. A. C.; MELO, E. L.; FREITAS, G. B.; SANTOS, C. A. S.; PIMENTEL, M. G. C. Discrimination of media moments and media intervals: sticker-based watch-and-comment annotation. *Multimedia Tools and Applications*, Springer, v. 61, n. 3, p. 675–696, 2012.
- [Troncy et al. 2010] TRONCY, R.; BAILER, W.; HÖFFERNIG, M.; HAUSENBLAS, M. Vamp: a service for validating mpeg-7 descriptions w.r.t. to formal profile definitions. *Multimedia Tools and Applications*, Springer, v. 46, n. 2, p. 307–329, 2010.
- [W3C 1999a] W3C. *Synchronized Multimedia Integration Language (SMIL) Document Object Model*. 1999. [Http://www.w3.org/TR/SMIL/smil-DOM.html](http://www.w3.org/TR/SMIL/smil-DOM.html). World-Wide Web Consortium Recommendation.
- [W3C 1999b] W3C. *XML Stylesheet Language Transformations (XSLT) Version 1.0*. 1999. [Http://www.w3.org/TR/xslt](http://www.w3.org/TR/xslt). World-Wide Web Consortium Recommendation.
- [W3C 2000] W3C. *Document Object Model (DOM) Level 2 Core Specification*. 2000. World-Wide Web Consortium Recommendation DOM-Level-2-Core-20001113.
- [W3C 2004a] W3C. *OWL Web ontology language: reference*. 2004.
- [W3C 2004b] W3C. *XML Schema Part 0: Primer Second Edition*. 2004.
- [W3C 2008a] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2008. World-Wide Web Consortium Recommendation.
- [W3C 2008b] W3C. *Synchronized Multimedia Integration Language - SMIL 3.0 Specification*. 2008. [Http://www.w3c.org/TR/SMIL3](http://www.w3c.org/TR/SMIL3). World-Wide Web Consortium Recommendation.
- [W3C 2011] W3C. *Scalable Vector Graphics (SVG) 1.1*. 2011. [Http://www.w3.org/TR/SVG11](http://www.w3.org/TR/SVG11). World-Wide Web Consortium Working Recommendation.
- [W3C 2014] W3C. *Web Animations 1.0*. 2014. [Http://www.w3.org/TR/web-animations/](http://www.w3.org/TR/web-animations/). World-Wide Web Consortium Working Draft.
- [Wang 2005] WANG, B.-Y. μ -calculus model checking in maude. *Electronic Notes in Theoretical Computer Science*, Elsevier, v. 117, p. 135–152, 2005.

APPENDIX A – Multimedia languages mapping

Chapter 5 presented the general approach for mapping multimedia documents to the \mathcal{SHM} implementation in *RWT*. It is performed by representing a given document d in an equation theory $\mathcal{E} = (\Sigma, E)$, where Σ represents the syntax of XML documents and E provides mappings from a given multimedia language into *RWT*.

As presented in the same section, we have extended \mathcal{E} for both NCL and SMIL languages. Those languages were chosen because they are standard declarative languages based on different paradigms for describing multimedia synchronization. NCL uses an event-based synchronization model and SMIL uses a hierarchy of temporal containers to specify synchronization relationships. In this appendix we give an overview of both languages and describe their mappings to *RWT*.

A.1 NCL

Nested Context Language (NCL) [ABNT 2011, ITU 2009] is an XML-based based multimedia language that describes a document in terms of nodes and links. NCL nodes represent media items and the document structure, while NCL links represent synchronization relationships among nodes.

An NCL node has one of two types: *content node* or *composite node*. A content node represents a media item (text, image, video, audio, etc) and a composite node represents a group of nodes, either content or composite. The idea of a composite node is to structure the NCL document providing “black boxes” that can be reused across documents. Following this approach, NCL links are declared inside compositions and can only relate nodes that are children of a given composition. A content node is declared in NCL by element *media*, while a composite node is declared by element *context*.

A special type of composite node is the NCL *switch*. Switches also represents a group of nodes, but it attaches conditions to nodes. The idea behind a switch is to

provide alternative content by following rules, where each rule specifies the condition to be attached to a given node. The condition specified in a switch rule defines a test over the value of a global variable. The NCL standard [ITU 2009] presents a set of predefined global variables representing system attributes. Moreover, an author can also define its own global variables. The set of global variables to be used inside a document is declared in a special type of content node called *settings*. Whenever a switch is presented only one of its inner nodes is presented. The chosen node is the first one whose rule was evaluated to *true*.

NCL defines how content nodes are presented on the screen with *regions* and *descriptors*. Regions define screen regions where content nodes will be presented and descriptors define presentation characteristics of content nodes, for example, duration, volume, a region where a content node using this descriptor will be presented, etc. With that separation, a same region can be reused in several descriptors, and a same descriptor can be reused in several nodes.

Links can relate nodes or subparts of them. Therefore nodes inside a composite node can be made directly accessible by providing mappings to them (in NCL such mappings are called ports). A subpart of a content node can be either a sub-interval of a media item presentation (e.g. a sequence of frames of a video) or a local variable of that node. Subparts of a node content are called *anchors* while local variables are called *properties*. Every content node has a special type of anchor, called *lambda anchor*, which represents the node entire content (i.e. the node itself).

Temporal relations in NCL are specified by hypermedia connectors [Muchaluat-Saade and Soares 2002]. Connectors define generic relations that can be used across the document. A given connector specifies the semantics of a relation without defining the participants of the relationship. NCL connectors define causal relations where a condition has to be satisfied before a set of actions is applied. A connector condition is triggered by an event occurrence in its source node, anchor or property and possibly a test over the value of properties or the state of anchors. A connector action triggers a change in the state of its target anchors. Event occurrences signal a change in the state of a state machine associated to nodes, anchors and properties.

NCL is an *event-based* multimedia language [Blakowski and Steinmetz 1996, Pérez-Luque and Little 1996]. A multimedia event is an occurrence in time with duration that happens during document execution, such as node presentation, viewer interaction or change of variable values. Therefore, each document element has associated to it state

machines representing its *presentation*, *selection* and *attribution* states. A *presentation* state machine represents a node presentation, a *selection* state machine represents a viewer interaction and an *attribution* state machine represents a property value change. The state of a document element is the current state of its associated state machine.

Every state machine starts in the *sleeping* state. As the NCL document executes, it eventually transits to the *occurring* state. If we suppose a state machine associated with a node representing a video object, as the first frame of the video begins its presentation, the node presentation machine transits to the *occurring* state. State machines remain in the *occurring* state for a given period of time. Thus, besides its state, every state machine has a countdown clock that registers the time it remains in the *occurring* state. The presentation state machine of the node will remain in the *occurring* state while its video frames are presented. As soon as the last frame of the video finishes its presentation (i.e. its clock reaches zero), the node's presentation machine goes back to the *sleeping* state. This is what we call the *natural end* of a node presentation. It is important to highlight that not all nodes may have a natural end. One example is a node representing an image, which does not have an inherent duration. In that case the presentation state machine of that node will remain in the *occurring* state indefinitely.

Every time a state changes from *occurring* to *sleeping*, the so-called *occurrences counter* is incremented. Such a counter enables the author to create relationships for a specific occurrence. For example, “whenever video v ends its presentation for the third time, start image i ”.

Links are defined by referencing a connector and defining a set of binds, associating each connector role to a node interface. Therefore, several links can reuse the same connector defining only the participant nodes. In this work, as a simplification, we refer to a link-connector pair just as link.

Figure A.1 presents an example of NCL document. We follow the same symbols used in NCL documentation [Soares and Barbosa 2012]: (i) solid circles represent content nodes, (ii) dashed circles represent composite nodes, (iii) filled squares represent node interfaces, (iv) arrows represent links and (v) dashed lines represent port mappings.

The document in Figure A.1 defines three nodes N_1 , N_2 and N_3 . Both N_2 and N_3 are inside composition C_2 , and both N_1 and C_2 are inside composition C_1 , which represents the document body. Port P_1 maps to node N_1 , which means that node N_1 will be executed when the document (composition C_1) begins. Port P_2 maps to node N_2 , which has anchor A_2 inside it.

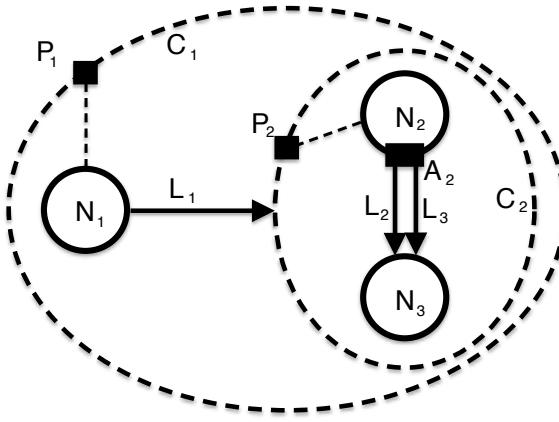


Figure A.1: NCL document example

The document also defines links L_1 , L_2 and L_3 . Link L_1 will start the presentation of node C_2 when node N_1 ends its presentation. Link L_2 will start the presentation of node N_2 when anchor A_2 starts its presentation. Link L_3 will end the presentation of node N_2 when anchor A_2 ends its presentation. An extract of the NCL code of this example presented in Listing A.1.

```

1 <ncl>
2   <head> ... </head>
3   <body id='C1'>
4     <port id='P1' component='N1' />
5     <media id='N1' src='video1.mp4' .../>
6
7     <context id='C2'>
8       <port id='P2' component='N2' />
9       <media id='N2' src='video2.mp4' ...>
10      <area id='A2' begin='5s' end='15s' />
11    </media>
12    <media id='N3' src='image.png' .../>
13
14    <link id='L2' xconnector='onBeginStart'>
15      <bind role='onBegin' component='N2' interface='A2' />
16      <bind role='start' component='N3' />
17    </link>
18    <link id='L3' xconnector='onEndStop'>
19      <bind role='onEnd' component='N2' interface='A2' />
20      <bind role='stop' component='N3' />
21    </link>
22  </context>
23
24  <link id='L1' xconnector='onEndStart'>
25    <bind role='onEnd' component='N1' />
26    <bind role='start' component='C2' />
27  </link>
28 </body>
29 </ncl>
```

Listing A.1: NCL document example

A.2 Mapping NCL to *RWT*

This section discusses the mapping of NCL to *RWT*. This is provided as a set of equations that transforms a term in the equational theory \mathcal{E}_{NCL} into *RWT*. For simplicity, we shall call this process τ_{NCL} .

While translating an NCL document to *RWT*, *units* are produced for representing document nodes, together with equations (using constructor *dur*) for the *unit* duration. For content nodes, τ_{NCL} produces *units* for the node's *lambda* anchor together with all anchors and properties declared for that node. For each composite node (context and the body of an NCL document) it produces a *unit* representing the whole node. Thus, all content information is preserved by this transformation modulo the hierarchical structure, which is not relevant for *RWT*, since we are interested only in the document's behavior.

According to user definition (recall function *ncl2shm()* presented in Section 5.1.1), τ_{NCL} creates *units* (u_{aux}) for each selection enabled anchor inside the document, together with rules to start the selection state machine configuration of a *unit* once one of its u_{aux} finishes its presentation.

For each content node inside the document, we create *rels* (as equations) to preserve the order of anchor presentation. Those *rels* are created as follows:

- For a start action a *rel* starts other anchors defined for the content node, respecting their order with respect to the target anchor of the start action.
- For each action: stop, pause, resume and abort a *rel* relays that action to all *units* representing anchors of the content node.

Moreover, we also create *rels* that end the presentation of the *unit* representing the *lambda* anchor, when *units* representing anchors declared for the content node end their presentation.

For each composite node inside the document, we create *rels* to relay actions performed over the composite node as a whole to inner nodes as follows:

- For a start action a *rel* relays that action to all *units* representing inner nodes of the composite node that are mapped by ports.
- For each action: stop, pause, resume and abort a *rel* relays that action to all *units* representing inner nodes of the composite node.

Moreover, we also create *rels* that end the presentation of the *unit* representing the composite node, when all of its internal *units* have ended their presentation and to start its presentation when at least one internal *unit* has begun its presentation.

For each link, τ_{NCL} produces a *rel* relating *units* representing the nodes (or node interfaces) declared in the link. Once all links in the NCL document are translated into *rels*, we simplify the *rel* set for a given document. This process of *rel* simplification follows the second normal form presented in [Lima and Soares 2013]. We perform this simplification step in the document transformation in order to guarantee that no concurrence appear in the *rel* equations set. Consider the following example:

$$eq [r_1] : end(u_1, pre) = start(u_2, pre) .$$

$$eq [r_2] : end(u_1, pre) = start(u_3, pre) .$$

A *rel*, once applied, removes from the term an event transition. Since both, r_1 and r_2 have the same condition in the NCL document, if we do not join them together in the associated equation (see equation r_1++r_2 below), a non-confluent equational theory would arise.

$$eq [r_1++r_2] : end(u_1, pre) = start(u_2, pre) \ start(u_3, pre) .$$

Listing A.2 presents the Maude specification of the NCL document in Figure A.1. Notice that nodes N_1 , N_2 and N_3 are represented by *units* representing their lambda anchors (u_{N_1} , u_{N_2} and u_{N_3}). *Unit* u_{A_2} represents anchor A_2 and *unit* u_D represents a delay for the presentation of *unit* u_{A_2} with respect to *unit* u_{N_2} . The set of document's anchors is declared by equation *doc*, while the action to be performed in the document begin is declared by equation *ini*. Equation *max* declares the maximum time allowed for this document execution before it is considered not to finish.

```

1 mod EXAMPLE is
2   protecting RRWT .
3
4   eq max = 3600 .
5   eq ini = init('uC1,pre) .
6   eq doc = state('uC1,pre,sleeping) occur('uC1,pre,0) clock('uC1,pre,none)
7     state('uN1,pre,sleeping) occur('uN1,pre,0) clock('uN1,pre,none)
8     state('uC2,pre,sleeping) occur('uC2,pre,0) clock('uC2,pre,none)
9     state('uN2,pre,sleeping) occur('uN2,pre,0) clock('uN2,pre,none)
10    state('uN3,pre,sleeping) occur('uN3,pre,0) clock('uN3,pre,none)
11    state('uA2,pre,sleeping) occur('uA2,pre,0) clock('uA2,pre,none)
12    state('uD,pre,sleeping) occur('uD,pre,0) clock('uD,pre,none) .
13
14   eq dur('uN1,pre) = 30 .
15   eq dur('uN2,pre) = 20 .

```

```

16  eq dur('uN3,pre) = inf .
17  eq dur('uA2,pre) = 10 .
18  eq dur('uD,pre) = 5 .
19
20  eq [P1] : init('uC1,pre) = start('uN1,pre) .
21  eq [L1] : end('uN1,pre) = start('uC2,pre) .
22  ...
23
24  eq [P2] : init('uC2,pre) = start('uN2,pre) start('uD,pre) .
25  eq [rel] : end('uD,pre) = start('uA2,pre) .
26  eq [L2] : init('uA2,pre) = start('uN3,pre) .
27  eq [L3] : end('uA2,pre) = stop('uN3,pre) .
28  ...
29 endm

```

Listing A.2: NCL document example in *RWT*

Links L_1 , L_2 and L_3 are represented by equations whose left side represents the link condition and the right side represents the link action. Moreover, ports P_1 and P_2 are also represented as equations, that starts the presentation of the nodes mapped by them. Each port equation has as condition the begin of its parent context presentation.

Given that anchor A_2 is presented during N_2 's presentation, the additional *unit* u_D is started whenever node N_2 is. An additional equation starts the presentation of anchor A_2 after the end of the auxiliar *unit*.

A.3 SMIL

Differently from NCL, Synchronized Multimedia Integration Language (SMIL) [W3C 2008b] describes a multimedia document in terms of nodes and containers with temporal semantics. SMIL nodes represent media items and the document structure given by the temporal containers represent synchronization relationships among nodes.

A SMIL node can be declared by different elements (*ref*, *text*, *img*, *audio*, *video*, *animation* and *textstream*). In practice, each different element name is a synonym for *ref* and is used to help the author infer the content of a given document. A node may declare all the content of a given media item or just a subpart of it. In the second case, a SMIL author may *crop* the begin and end of a given media item presentation by selecting just the fragment of interest. SMIL defines how nodes are presented on the screen with *regions*. Regions define screen regions where nodes using them will be presented.

SMIL containers have a temporal semantics embedded to it. There are three containers in SMIL: *par*, *seq* and *excl*. Each temporal container defines a reference point for the presentation of an inner component (node or other container) called *syncbase*. Whenever a component declares a delay for its presentation, this delay is considered with respect to

its *syncbase*. The *syncbase* of a component is defined as follows:

- The *par* container presents its inner components in parallel. It means that the *syncbase* of every inner component is the beginning of the presentation of the container.
- The *seq* container presents its inner components in sequence, following the order they are declared in the document. It means that the *syncbase* of a component is the end of its predecessor. The *syncbase* for the first component is the beginning of the presentation of the container.
- The *excl* container presents only one of its inner components at a time. This means that once a component starts its presentation, another (possible) component being presented is ended. For this container the default *syncbase* is set to *indefinite*. Therefore, the start time of a given component inside the *excl* container must be explicitly related to an event occurrence inside the document (e.g. viewer interaction or the begin or end of another node or container).

Listing A.3 presents an example where the *syncbase* of components inside a *par* container are overwritten.

```

1 <par id='comp'>
2   <text id='title' ... begin='2s' dur='5s' />
3   <video id='video' ... begin='3s' />
4   <audio id='audio' ... begin='video.begin+5s' />
5 </par>

```

Listing A.3: *Syncbase* overwrite

In the example presented in Listing A.3, node *text* will begin its presentation two seconds after the beginning of the presentation of the *par* container, followed by node *video* which will start three seconds after the *par* container. Node *audio* will begin its presentation five seconds after the beginning of the presentation of node *video*. Notice that the *syncbase* for node *audio* is modified for the beginning of another node and a delay is also added.

The *par* container ends its presentation when its last inner component ends its presentation. This behavior can be overwritten allowing the container to end with the first inner component to finish or together with a chosen component.

SMIL elements may relate their start time to an event occurrence inside the document as seen in Listing A.3. The same can be done for their end time. SMIL provides a set of

event occurrences that can be used for determining an element start or end time. They are:

- beginEvent**: occurs when an element starts its presentation.
- endEvent**: occurs when an element ends its presentation.
- pause**: occurs when an element is paused.
- resume**: occurs when an element resumes its presentation.
- mediacomplete**: occurs when an element finishes loading.
- timeerror**: occurs when an invalid time value is set to a time attribute.
- activateEvent** or **click**: occurs when the user clicks over an element.
- inBoundsEvent** or **mouseover**: occurs when the mouse pointer enters the element.
- outOfBoundsEvent** or **mouseout**: occurs when the mouse pointer exits the element.
- focusInEvent**: occurs when an element receives focus through the keyboard.
- focusOutEvent**: occurs when an element loses focus through the keyboard.

When using events, the author must indicate the element whose event is expected in the format “element.id.event”. It is also possible to add a delay with the format “element.id.event+delay”. Events are particularly useful together with *excl* containers, since its inner components do not have a default start time.

SMIL also allows the alternative presentation of content through element *switch*. It delimits a set of alternative content and each element inside of the *switch* defines a condition for its presentation. The conditions are evaluated in the order the elements are declared inside the document. The first element whose condition is evaluated to *true* is presented. If an element does not have a condition, it is considered to be *true*. Switch tests are done over global variables representing system attributes. The set of predefined global variables is defined in the SMIL standard [W3C 2008b].

Figure A.2 presents a SMIL document in pictorial form. We follow the same symbols used in Figure A.1.

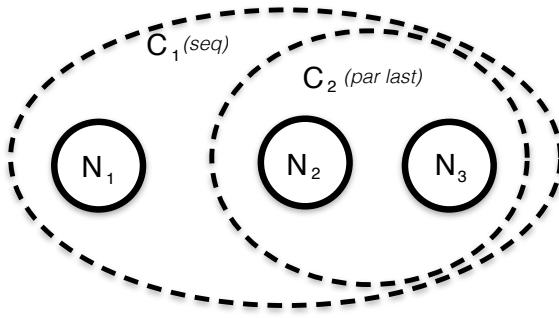


Figure A.2: SMIL document example

The document in Figure A.2 defines three nodes N_1 , N_2 and N_3 . Both N_2 and N_3 are inside container C_2 , a *par* container with its end given by the last element to end among N_2 and N_3 (*endsync* = “*last*”). Both N_1 and C_2 are inside container C_1 , a *seq* container, which represents the document body. An extract of the SMIL code of this example presented in Listing A.4.

```

1 <smil>
2   <head> ... </head>
3   <body id='C1'>
4     <video id='N1' src='video1.mp4' .../>
5
6     <par id='C2' endsync='last'>
7       <video id='N2' src='video2.mp4' .../>
8       <video id='N3' src='video3.png' .../>
9     </par>
10   </body>
11 </smil>
```

Listing A.4: SMIL document example

A.4 Mapping SMIL to *RWT*

This section discusses the mapping of SMIL to *RWT*. This is provided as a set of equations that transforms a term in the equational theory \mathcal{E}_{SMIL} into *RWT*. For simplicity, we shall call this process τ_{SMIL} .

While translating a SMIL document to *RWT*, *units* are produced for representing document nodes, together with equations (using constructor `dur`) for the *unit* duration. Since every node represents a media item as a whole or a subpart of its content, τ_{SMIL} produces one *units* for each node. For each temporal container (par, seq, excl and the body of a SMIL document) it produces a *unit* representing the whole container together with *rels* for representing its behavior.

The same way it is done for NCL, according to user definition (recall function `ncl2shm()`

presented in Section 5.1.1), τ_{SMIL} creates *units* (u_{aux}) for each selection enabled anchor inside the document, together with rules to start the selection state machine configuration of a *unit* once one of its u_{aux} finishes its presentation.

For each temporal container inside the document, we create *rels* that start the container inner *unit* following their *syncbase*, as described in Section A.3, and *rels* that end the presentation of the *unit* representing the container, when internal *units* end their presentation. Those *rels* are created as follows:

- For a *par* container, when the *unit* representing the container starts its presentation, a *rel* will start the presentation of all inner elements, considering delays declared by inner elements. If a given element overrides its *syncbase* with respect to an event occurrence, a different *rel* is created having the declared event as condition.
- For a *par* container, a *rel* will end the presentation of the *unit* representing the container when one of its inner *units* ends (i.e. the first one ends); when one of its inner *units* ends and all other units are in the sleeping state (i.e. the last one ends); or when a given inner *unit* ends (i.e. the chosen one ends). The choice is made depending on the SMIL *par* container attribute *endsync*.
- For a *seq* container, when the *unit* representing the container starts its presentation, a *rel* will start the presentation of the first inner element. Other *rels* will start the presentation of the i-th inner element when the previous element ends its presentation. Finally, a *rel* ends the presentation of the *unit* representing the container when the last inner element ends its presentation. Every *rel* must consider eventual delays for the presentation of inner elements.
- For an *excl* container, when a container inner *unit* starts its presentation, a *rel* will start the presentation of the *unit* representing the container. Besides, for each inner element, a *rel* will end the presentation of every other inner *unit* when it starts its presentation. For each *unit* inside the container that overrides its *syncbase*, with respect to an event occurrence, a different *rel* is created having the declared event as condition.

For each switch element inside the SMIL document, we create *rels* (one for each inner element) that start the presentation of a given inner *unit* when the *unit* representing the switch starts its presentation and the condition attached to the *unit* is evaluated to *true*.

Finally, for each node whose *syncbase* was overwritten by the use of events, τ_{SMIL}

produces a *rel* relating the event defined in the node *syncbase* with the *unit* representing itself. The same way it is done for NCL, once all the document behavior is mapped into *rels*, we simplify the *rel* set for a given document.

Listing A.5 presents the Maude specification of the document in Figure A.2. Notice that nodes N_1 , N_2 and N_3 are represented by *units* u_{N_1} , u_{N_2} and u_{N_3} . *Unit* u_{C_1} represents container C_1 and *unit* u_{C_2} represents container C_2 .

```

1 mod EXAMPLE is
2   protecting RRWT .
3
4   eq max = 3600 .
5   eq ini = init('uC1,pre) .
6   eq doc = state('uN1,pre,sleeping) occur('uN1,pre,0) clock('uN1,pre,none)
7     state('uN2,pre,sleeping) occur('uN2,pre,0) clock('uN2,pre,none)
8     state('uN3,pre,sleeping) occur('uN3,pre,0) clock('uN3,pre,none)
9     state('uC1,pre,sleeping) occur('uC1,pre,0) clock('uC1,pre,none)
10    state('uC2,pre,sleeping) occur('uC2,pre,0) clock('uC2,pre,none) .
11
12  eq dur('uN1,pre) = 25 .
13  eq dur('uN2,pre) = 20 .
14  eq dur('uN3,pre) = 25 .
15  eq dur('uC1,pre) = inf .
16  eq dur('uC2,pre) = inf .
17
18  eq [C1] : init('uC1,pre) = start('uN1,pre) .
19  eq [C1] : end('uN1,pre) = start('uC2,pre) .
20  eq [C1] : end('uC2,pre) = stop('uC1,pre) .
21
22  eq [C2] : init('uC2,pre) = start('uN2,pre) start('uN3,pre) .
23  eq [C2] : end('uN2,pre) state('uN3,pre,sleeping) = stop('uC2,pre) .
24  eq [C2] : end('uN3,pre) state('uN2,pre,sleeping) = stop('uC2,pre) .
25 endm

```

Listing A.5: SMIL document example in *RWT*

The synchronization relation of container C_1 is represented by three relationships. The first one starts N_1 when C_1 begins its presentation. The second one starts C_2 when N_1 ends its presentation. The third one, ends C_1 when C_2 ends its presentation.

The synchronization relation of container C_2 is also represented by three relationships. The first one starts N_2 and N_3 when C_2 starts its presentation. The second one ends C_2 when N_2 ends its presentation and N_3 is in the *sleeping* state. The third one ends C_2 when N_3 ends its presentation and N_2 is in the *sleeping* state.

APPENDIX B - Rewriting logic

This work proposes a validation approach based on model checking. We chose rewriting logic theory for implementing our validation approach. One motivation for the use of rewriting theory is the presence of available tools that already implements the rewriting theory calculus and model checker evaluation strategies. One of such tools is Maude [Clavel et al. 2007], which implements it with a very competitive rewriting performance as indicated by [Kirchner and Moreau 2001]. In this appendix, we give a brief overview of rewriting logic, presenting the key concepts for this work.

B.1 Rewriting logic calculus

In rewriting logic¹, an *equational theory* is a pair (Σ, E) with Σ a ranked alphabet of function symbols and E a set of Σ -equations. Rewriting operates on equivalence classes of terms modulo a given set of equations E , denoted by $[t]_E$. In this way, rewriting is free from the syntactic constraints of a term representation thanks to the “structural axioms” E , in deciding what counts as a *data structure*; for example, string rewriting is obtained by imposing an associativity axiom, and multiset rewriting by imposing associativity and commutativity. Standard term rewriting is obtained as the particular case in which the set E of equations is empty.

Given a equational theory (Σ, E) , its *sentences* are sequents of the form $[t]_E \rightarrow [t']_E$ with t, t' Σ -terms, where t and t' possibly involve some variables from the countably infinite set $X = \{x_1, \dots, x_n, \dots\}$.

The notion of rewrite theory presented below is very general and expressive. In the first place, as already mentioned, it allows for rewriting modulo “structural axioms” E , thus increasing the expressive power. In addition, it allows for *conditional* rules of a very general form, where the conditions do not require equalities to hold but only the

¹This Section borrows and adapts prose from [Meseguer 1992, Section 2.2].

existence of rewritings among pairs of terms in the condition, which further increases the expressive power. Finally, it allows *labeling* of the rewrite rules; this is quite natural for many applications, and customary for automata (viewed as labelled transition systems) and for Petri nets, which are both particular instances of this definition.

Definition 4. A (*labelled*) *rewrite theory* \mathcal{R} is a 4-tuple $\mathcal{R} = (\Sigma, E, L, R)$ where Σ is a ranked alphabet of function symbols, E is a set of Σ -equations, L is a set called the set of *labels*, and R is a set of pairs $R \subseteq L \times (T_{\Sigma, E}(X)^2)^+$ whose first component is a label and whose second component is a nonempty sequence of pairs of E -equivalence classes of terms, with $X = \{x_1, \dots, x_n, \dots\}$ a countably infinite set of variables. Elements of R are called *rewrite rules*. For a rewrite rule $(r, ([t], [t']) ([u_1], [v_1]) \dots ([u_k], [v_k]))$ we use the notation

$$r : [t] \rightarrow [t'] \text{ if } [u_1] \rightarrow [V_1] \wedge \dots \wedge [u_k] \rightarrow [v_k].$$

The part $[u_1] \rightarrow [V_1] \wedge \dots \wedge [u_k] \rightarrow [v_k]$ is called the *condition* of the rule, and may abbreviate it with the letter C . To indicate that $\{x_1, \dots, x_n\}$ is a set of variables occurring in either t , t' , or C , we write $r : [t(x_1, \dots, x_n)] \rightarrow [t'(x_1, \dots, x_n)]$ if $C(x_1, \dots, x_n)$, or in abbreviated notation $r : [t(\bar{x}^n)] \rightarrow [t'(\bar{x}^n)]$ if $C(\bar{x}^n)$. \triangle

Given a rewrite theory \mathcal{R} , we say that \mathcal{R} *entails* a sequent $[t] \rightarrow [t']$ and write

$$\mathcal{R} \vdash [t] \rightarrow [t']$$

if and only if $[t] \rightarrow [t']$ can be obtained by finite application of the following *rules of deduction*:

(1) **Reflexivity.** For each $[t] \in T_{\Sigma, E}(X)$,

$$\overline{[t] \rightarrow [t]}.$$

(2) **Congruence.** For each $f \in \Sigma_n$, $n \in \mathbb{N}$,

$$\frac{[t_1] \rightarrow [t'_1] \quad \dots \quad [t_n] \rightarrow [t'_n]}{[f(t_1, \dots, t_n)] \rightarrow [f(t'_1, \dots, t'_n)]}.$$

(3) **Replacement.** For each rule

$$r : [t(\bar{x})] \rightarrow [t'(\bar{x})] \text{ if } [u_1(\bar{x})] \rightarrow [v_1(\bar{x})] \wedge \dots \wedge [u_k(\bar{x})] \rightarrow [v_k(\bar{x})]$$

in R ,

$$\frac{[w_1] \rightarrow [w'_1] \quad \dots \quad [w_n] \rightarrow [w'_n]}{\frac{[u_1(\bar{w}/\bar{x})] \rightarrow [v_1(\bar{w}/\bar{x})] \quad \dots \quad [u_k(\bar{w}/\bar{x})] \rightarrow [v_k(\bar{w}/\bar{x})]}{[t(\bar{w}/\bar{x})] \rightarrow [t'(\bar{w}'/\bar{x})]}}.$$

That is, for a substitution $x_i \mapsto w_i$, $1 \leq i \leq n$, we can deduce sequents

$$[u_j(\bar{w}/\bar{x})] \rightarrow [v_j(\bar{w}/\bar{x})], \quad 1 \leq j \leq k,$$

then, if in addition we can deduce $[w_i] \rightarrow [w'_i]$, $1 \leq i \leq n$, we are then allowed to deduce $[t(\bar{w}/\bar{x})] \rightarrow [t'(\bar{w}'/\bar{x})]$.

(4) **Transitivity.**

$$\frac{[t_1] \rightarrow [t_2] \quad [t_2] \rightarrow [t_3]}{[t_1] \rightarrow [t_3]}$$

Equational logic (modulo a set of axioms E) is obtained from rewriting logic by adding the following rule:

(5) **Symmetry.**

$$\frac{[t_1] \rightarrow [t_2]}{[t_2] \rightarrow [t_1]}$$

B.2 Rewrite metatheories

Rewriting logic is reflective in a precise mathematical way, namely, there is a finitely presented rewrite theory \mathcal{U} that is universal in the sense that we can represent in \mathcal{U} any finitely presented rewrite theory \mathcal{R} (including \mathcal{U} itself) as a term $\bar{\mathcal{R}}$, any terms $t, t' \in \mathcal{R}$ as terms \bar{t}, \bar{t}' , and any pair (\mathcal{R}, t) as a term $\langle \bar{\mathcal{R}}, \bar{t} \rangle$, in such a way that we have the following equivalence

$$\mathcal{R} \vdash [t] \rightarrow [t'] \leftrightarrow \mathcal{U} \vdash [\langle \bar{\mathcal{R}}, \bar{t} \rangle] \rightarrow [\langle \bar{\mathcal{R}}, \bar{t}' \rangle].$$

Since \mathcal{U} is representable in itself, we can achieve a “reflective tower” with an arbitrary number of levels of reflection:

$$\mathcal{R} \vdash [t] \rightarrow [t'] \Leftrightarrow \mathcal{U} \vdash [\langle \bar{\mathcal{R}}, \bar{t} \rangle] \rightarrow [\langle \bar{\mathcal{R}}, \bar{t}' \rangle] \Leftrightarrow \mathcal{U} \vdash [\langle \bar{\mathcal{U}}, \langle \bar{\mathcal{R}}, \bar{t} \rangle \rangle] \rightarrow [\langle \bar{\mathcal{U}}, \langle \bar{\mathcal{R}}, \bar{t}' \rangle \rangle] \dots$$

B.3 Maude

B.3.1 Syntax

This section presents a subset of Maude's syntax large enough to properly formalize NCL semantics. (We refer the reader to [Clavel et al. 2007] for a full account of Maude's syntax, semantics and applications.)

Maude modules have a mathematical meaning that precisely captures the semantics of Rewriting Logic theories. A Maude module is a formal specification that is also executable under certain conditions, as recalled later in this section and thoroughly addressed in many places such as [Clavel et al. 2007]. For this reason, one may write a Rewriting Logic specification directly in Maude syntax, which is quite close to many declarative languages from functional or logic programming realms.

Given a Rewriting Logic theory $\mathcal{R} = (\Sigma, E, R)$, one may declare a Maude system module (or simply module, as it will be called throughout this paper) for \mathcal{R} . Modules are declared using syntax

```
mod <math>\langle \mathcal{R} \rangle</math> is ... endm
```

where $\langle \mathcal{R} \rangle$ is the module's name.

The signature Σ of \mathcal{R} is declared in a Maude module using sort declarations and operation declarations. A sort is declared as follows:

```
sort <math>\langle Sort \rangle</math> .
```

where $\langle Sort \rangle$ means the name of the sort. Sort inclusion can be specified by a *subsort* declaration

```
subsort <math>\langle Sort_1 \rangle</math> < <math>\langle Sort_2 \rangle</math> .
```

meaning that $\langle Sort_1 \rangle$ is a subsort of $\langle Sort_2 \rangle$.

Maude operators are declared using the *op* keyword with syntax

$$op \langle OpName \rangle : \langle Sort_1 \rangle \dots \langle Sort_k \rangle \rightarrow \langle Sort \rangle [\langle OperAtt \rangle] .$$

where $\langle OpName \rangle$ is the name of the operation, sorts $\langle Sort_1 \rangle$ to $\langle Sort_k \rangle$ are the list of arguments sorts and sort $\langle Sort \rangle$ is the result sort. The so-called structural axioms, such as, associativity, commutativity, idempotence and identity, are declared using $\langle OperAtt \rangle$ with syntax such as *assoc* for associative and *comm* for commutative properties. Operators can be declared using mixfix syntax, where the symbol $_$ identifies the place of an argument.

Equations are declared in Maude using keyword *eq* but may be conditional with general form

$$ceq l = r \text{ if } u_1 = v_1 \wedge \dots \wedge u_n = v_n .$$

where $l = r$ and $u_i = v_i$ are Σ -equations. Equations may be annotated with *label* or *owise* attributes. The former names an equation and the latter specifies that the given equation should be applied otherwise, when no other equation can, for a given pattern l .

Given a Rewriting Logic theory \mathcal{R} , a labeled rewrite rule $r : [t]_E \Rightarrow [t']_E$ in R is specified in Maude using the keyword *rl*, where r is a label and $[t]_E$, $[t']_E$ are congruence classes of terms in $T_{\Sigma,E}(X)$. The general form of a rewrite rule is conditional, declared as follows

$$crl [r] : t \Rightarrow t' \text{ if } (\bigwedge_i u_i = v_i) \wedge (\bigwedge_k p_k \Rightarrow q_k) .$$

Other modules may be imported by a given module. Importation can be done in different modes: *protecting*, *extending* or *including*. Importation in protecting mode preserves the algebraic semantics of imported module in the importing module by not adding either “junk” or “confusion”². Importation in extending mode is looser in the precise sense of allowing “junk” but no “confusion” to the semantics of the included module. Finally, module inclusion specifies that there are no guarantees that the algebraic semantics of the included module will be preserved as both “junk” and “confusion” may be added to the included module. The importation mode is *not* enforced by the Maude interpreter.

²No “junk” means that every data item can be constructed using only the constants and operations in the signature. A data item that cannot be so constructed is junk. No “confusion” means that two data items are equivalent if and only if they can be proved equal using the equations.

B.3.2 Executability constraints

For a given Maude module to be executable, a few constraints must be fulfilled. The equational part of a given module, denoted (Σ, E) , must be *confluent*, *terminating*, *preregular* and *sort-decreasing*. For the remainder of this Section, let $t \in T_\Sigma$. Confluence can be presented in a diagrammatic form as in Figure B.1.

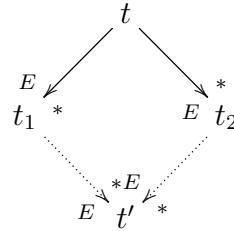


Figure B.1: Confluence property

Definition 5 (Confluence). A set of equations E is *confluent* when any two rewritings t_1 and t_2 , by equational simplification, of a given term t , can always be unified by further rewriting. If $t \rightarrow_E^* t_1$ and $t \rightarrow_E^* t_2$, then there is a term t' such that $t_1 \rightarrow_E^* t'$ and $t_2 \rightarrow_E^* t'$.

Definition 6 (Termination). A set of equations E is *terminating* when, for any given term t , there is no infinite sequence of E -rewriting steps from t .

If a set of equations E is both confluent and terminating, a term t can be reduced to a *unique canonical form* $t \downarrow_E$, that is, to a term that can no longer be rewritten.

Definition 7 (Preregularity). *Preregularity* requires each term t to have a least sort that can be assigned to it.

Definition 8 (Sort-decreasingness). Assuming a set of equations E confluent and terminating, the canonical form $t \downarrow_E$ of a term t by the equations E should have the least sort possible among the sorts of all the terms equivalent to it by the equations E . Moreover, it should be possible to compute this least sort from $t \downarrow_E$ itself, using only the operator declarations in Σ .

Maude assumes that modules are preregular and generates warnings, while loading a module, otherwise. The equational part of a module (Σ, E) is said Church-Rosser and terminating when it is confluent, terminating, and sort-decreasing.

A Maude module (Σ, E, R) has both rules R and equations E . Rule rewriting is thus performed modulo such equations. Equations are divided into a set A of *structural* axioms

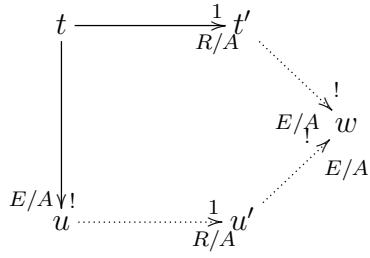


Figure B.2: Coherence property

(associativity, commutativity, idempotence and identity), for which matching algorithms are implemented in Maude, and a set E of equations that are assumed Church-Rosser and terminating as discussed before.

The rules R in the module must be *coherent* with the equations E modulo A , to allow proper shuffle of rewriting with rules (or simply rewriting) and rewriting with equations (or simply simplification). Without coherence, rewrites could be lost by not performing a possible rewrite before a simplification. Coherence can be presented in diagrammatic form as in Figure B.2, where E/A denotes equational reduction modulo A , and R/A denotes rewriting modulo A .

Definition 9 (Coherence). Coherence means that, given a term t , for each one-step rewrite of t with some rule in R modulo the axioms A to some term t' , denoted $t \rightarrow_{R/A}^1 t'$, if u is the canonical term we obtain by rewriting t with the equations and memberships in E to canonical form modulo A , denoted $t \rightarrow_{E/A}^! u$, then there is a one-step rewrite of u with some rule in R modulo A , $u \rightarrow_{R/A}^1 u'$, such that $t' =_{E \cup A} u'$, which by the Church-Rosser and termination properties of E modulo A is equivalent to t' and u' having the same canonical form modulo A by E .

Maude *rewrites modulo $E \cup A$* in a clever way: a term is reduced to canonical form using E before applying any rule in R . Thus, the effect of rewriting modulo $E \cup A$ is achieved with just a matching algorithm for A .

B.4 Set-rewriting theories and Reflection in Maude

We call a set-rewriting theory a subclass of rewriting logic theories that allows for term rewriting *module* associative, commutative, idempotence and identity axioms, together with the rewriting logic calculus. A set-rewriting theory provides atomic concurrent transitions, which corresponds to applying rewrite rules to a sub-term of a term while the rest

of the term is not affected by this atomic transition. Thus, rewrites can take place at the rest of the term at the same time the atomic transition happens.

Listing B.1 presents an example of a set-rewriting theory in Maude, where terms of sort *Conf* form “a soup” of terms of sort *Elem*. Operation `--` (line 9) represents the juxtaposition of elements that are rewritten modulo associative, commutative, idempotence and identity axioms. (Idempotence has to be explicitly specified as there is not an efficient algorithm for rewriting modulo associative, commutative, idempotence and identity axioms and therefore there is no implementation for such theories in Maude.) The identity of `--` is the constant *none*. The operator attribute *ctor* is used to determine operations that appear when a term is reduced to its canonical form.

```

1 mod FOO is
2   protecting NAT .
3
4   sort Elem Conf .
5   subsort Elem < Conf .
6
7   op none : -> Conf [ctor] .
8   ops e f : Nat -> Elem [ctor].
9   op -- : Conf Conf -> Conf [assoc comm ctor id: none] .
10
11  var E : Elem . vars n m : Nat .
12
13  eq [idem] : E E = E .
14  eq [e2f] : e(n) = f(n) .
15
16  crl [fsum] : f(n) f(m) => f(n + m) if m == n + 1 .
17 endm

```

Listing B.1: Set-rewriting example

The rule in line 16 declares that whenever we find two elements with number parameters differing by 1 in a term of sort *Conf* “they may react” producing a new element whose number parameter is the sum of the parameter of the original elements. Notice that the use of associative, commutative and identity axioms for the construction of terms of sort *Conf*, together with the congruence rule of the rewriting logic calculus, allows for very expressive patterns such as idempotence of terms of sort *Elem* as subterms of a *Conf* term declared in Equation *idem*. Similarly, equation *e2f* that replaces terms constructed with operator *e* by terms constructed with operator *f* can be applied anywhere in a term of sort *Conf*. Moreover, the application of the rule that identifies two terms constructed with operator *f*, with one term parameterized by the successor of the parameter of other term, in any order, and replaces them by a term constructed by operator *f* parameterized by the sum of the parameters, can occur anywhere inside a term of sort *Conf*.

On a modeling note, set-rewriting enables us to specify simple and yet expressive theories as the semantics of a specification language. Moreover, from an executability

perspective, it takes advantage of the term rewriting algorithms implemented in the Maude system specially designed to perform matching modulo such axioms.

Maude allows modules to be metarepresented as terms in a data type `Module` of modules. In such a representation, constants, variables and the name of modules, sorts and operations are represented as quoted identifiers (a *string* preceded by '`'`).

```

1 mod 'FOO is
2   protecting 'NAT .
3
4   sorts 'Conf ; 'Elem .
5   subsort 'Elem < 'Conf .
6
7   op 'none : nil -> 'Conf [ ctor ] .
8   op 'e : 'Nat -> 'Elem [ ctor ] .
9   op 'f : 'Nat -> 'Elem [ ctor ] .
10  op '_+_- : 'Conf 'Conf -> 'Conf [ assoc comm ctor id('none.Conf) ] .
11
12  none
13
14  eq '_- [ 'E:Elem , 'E:Elem ] = 'E:Elem [ label('idem) ] .
15  eq 'e[ 'n:Nat ] = 'f[ 'n:Nat ] [ label('e2f) ] .
16
17  crl '_- [ 'f[ 'n:Nat ] , 'f[ 'm:Nat ] ] => 'f[ '_+- [ 'n:Nat , 'm:Nat ] ] if '_==_ [ 'm:Nat , '_+- [ 'n:Nat , 's_- [ '0 . Zero
     ]] ] = 'true . Bool [ label('fsum) ] .
18 endm

```

Listing B.2: Meta representation of set-rewriting example

Terms constructed by a operation is metarepresented using constructor `_[-]` with the first parameter being an identifier and the second parameter a metarepresentation of a term. For example, as can be seen in line 15 of Listing B.2, term `e(n)` is metarepresented as `'e['n:Nat]`. As another example, as seen in line 17, term `n + m` is metarepresented as `'_+- ['n:Nat , 'm:Nat]`.

B.5 Representing finite transition systems in rewriting logic

A transition system $\mathcal{M} = (\Gamma, L, \rightarrow)$, where Γ is the set of configurations of \mathcal{M} , L is a set of labels, and $\rightarrow \subseteq \Gamma \times L \times \Gamma$, is represented by a term rewriting system $\mathcal{R}_{\mathcal{M}}$ as follows,

$$(\Gamma, \rightarrow) \mapsto \mathcal{T}((\Sigma_{\Gamma}, E_{\rightarrow}), R_{\rightarrow})$$

where, $\mathcal{T}(\Sigma_{\Gamma}, E_{\rightarrow})$ is the initial algebra of the rewriting logic theory $(\Sigma_{\Gamma}, E_{\rightarrow})$, Σ_{Γ} represents the signature of states in Γ , the equations in E_{\rightarrow} induce rewritings representing unobservable transitions in \rightarrow , in an observation theory [Milner 1999] sense, and the rules in R_{\rightarrow} induce rewritings representing observable transitions in \rightarrow .

This idea is depicted in Figure B.3, where a circle represents a transition system state and arrows represent transitions between states.



Figure B.3: Transition systems and rewriting logic

B.6 Searching and model checking

In Listing B.1 we presented an example of a Maude module. We are able to rewrite a term built from the operations contained in this module by using the *rewrite* command. An example of term rewrite in the module in Listing B.1 is presented in Listing B.3.

```

1 rewrite in FOO : e(0) e(1) e(2) e(3) e(0) e(2) .
2 rewrites: 15 in 0ms cpu (0ms real) (~ rewrites/second)
3 result Elem: f(3)
```

Listing B.3: Set-rewriting example

The rewrite command chooses one of the possible traces of a term rewrite. However, as presented in Section B.5 in opposite to equations, rules may be non-deterministic, thus producing concurrent rewrites. In Maude we can search in the “space state” of a term rewrite using the command *search*. In Listing B.4 we present the result of a search considering the same term used in the example in Listing B.1. In the example we search for all states with type *Conf*.

```

1 search in FOO : e(0) e(1) e(2) e(3) e(0) e(2) =>* c:Conf .
2
3 Solution 1 (state 0)
4 states: 1 rewrites: 4 in 0ms cpu (0ms real) (100000 rewrites/second)
5 c:Conf --> f(0) f(0) f(1) f(2) f(2) f(3)
6
7 Solution 2 (state 1)
8 states: 2 rewrites: 6 in 0ms cpu (0ms real) (32967 rewrites/second)
9 c:Conf --> f(1) f(1) f(2) f(2) f(3)
10
11 Solution 3 (state 2)
12 states: 3 rewrites: 8 in 0ms cpu (0ms real) (33195 rewrites/second)
13 c:Conf --> f(0) f(0) f(1) f(3) f(3)
14
15 Solution 4 (state 3)
16 states: 4 rewrites: 10 in 0ms cpu (0ms real) (30303 rewrites/second)
17 c:Conf --> f(2) f(2) f(2) f(3)
18
19 Solution 5 (state 4)
20 states: 5 rewrites: 12 in 0ms cpu (0ms real) (31496 rewrites/second)
21 c:Conf --> f(1) f(1) f(3) f(3)
22
23 Solution 6 (state 5)
24 states: 6 rewrites: 16 in 0ms cpu (0ms real) (35087 rewrites/second)
```

```

25 c:Conf --> f(0) f(0) f(1) f(4)
26
27 Solution 7 (state 6)
28 states: 7   rewrites: 18 in 0ms cpu (0ms real) (34749 rewrites/second)
29 c:Conf --> f(2) f(3) f(3)
30
31 Solution 8 (state 7)
32 states: 8   rewrites: 22 in 0ms cpu (0ms real) (37671 rewrites/second)
33 c:Conf --> f(1) f(1) f(4)
34
35 Solution 9 (state 8)
36 states: 9   rewrites: 26 in 0ms cpu (0ms real) (39393 rewrites/second)
37 c:Conf --> f(2) f(4)
38
39 No more solutions.
40 states: 9   rewrites: 28 in 0ms cpu (0ms real) (39215 rewrites/second)

```

Listing B.4: Example of term search

The Maude model checker is executed through command *modelCheck*. This command receives a term and an LTL formula to be evaluated. It takes into consideration the same “state space” of the *search* command. Whenever a formula does not hold, the model checker produces a counterexample presenting a trace where the given formula does not hold.

In order to use the Maude model checker with the example in Listing B.1, we have to extend it by defining sort *Conf* as a subsort of sort *State* and declare atomic properties as presented in Listing B.5.

```

1 mod EXPL is
2   protecting MODEL-CHECKER .
3   protecting FOO .
4
5   subsort Conf < State .
6
7   op has : Nat -> Prop .
8
9   var n : Nat .
10  var c : Conf .
11
12  eq f(n) c |= has(n) = true .
13  eq c |= has(n) = false [owise] .
14 endm

```

Listing B.5: LTL Property definition

In Listing B.5 we define property *has* to test if there exists an element f_i in the term for a given value of i . In Listing B.6 we present the result of two *modelCheck* commands considering the same term used in the examples in Listings B.1 and B.4.

```

1 reduce in EXPL : modelCheck(e(0) e(1) e(2) e(3) e(0) e(2), <> has(1)) .
2 rewrites: 9 in 0ms cpu (0ms real) (34351 rewrites/second)
3 result Bool: true
4
5 reduce in EXPL : modelCheck(e(0) e(1) e(2) e(3) e(0) e(2), <> has(5)) .
6 rewrites: 21 in 0ms cpu (0ms real) (77777 rewrites/second)
7 result ModelCheckResult: counterexample({f(0) f(0) f(1) f(2) f(2) f(3),unlabeled} {f(1) f(1) f(2) f(2) f(3),unlabeled} {f(2) f(2) f(3) f(3),unlabeled}, {f(2) f(4),deadlock})

```

Listing B.6: *modelCheck* result

The first *modelCheck* command evaluates if there is a future state (from the initial state) where we have element f_1 inside the term. The second *modelCheck* command evaluates if there is a future state where we have element f_5 . The second command produces a counterexample with a trace where we can not find a state with element f_5 .

APPENDIX C – Satisfiability Modulo Theories

Part of this work proposal is a validation approach based on Satisfiability Modulo Theories (SMT). In such approach, multimedia document fragments are represented by intervals, in either time and space, and constraints between such intervals. One motivation for choosing SMT is the possibility of defining such intervals and constraints in a direct way, by using variables and inequalities in SMT. Moreover, by solving the resulting formula that represents a given document it is possible to define a valuation for fragment intervals. For the implementation of the SMT validation approach, we use the SMT solver Yices2 [Dutertre 2014].

This appendix is structured as follows. In Section C.1 we present an overview of SMT and Section C.2 it is presented an overview of the SMT solver Yices2.

C.1 SMT Overview

In this section, we give a brief overview of SMT, presenting the key concepts for this work. The text here presented borrows and adapts from [Moura and Bjørner 2011].

The propositional satisfiability, or SAT, problem is a well-known constraint satisfaction problem, where a solver decides whether or not a given boolean formula over boolean variables, called atoms, and composed using logical connectives (such as disjunction, conjunction, negation, implication and etc) can be made *true* or not by choosing *true/false* values for each atom in the formula. A formulas is, therefore, said to be satisfiable when it can be made *true* and unsatisfiable otherwise.

An approach for choosing values to atoms is through *systematic search*. In such case, the search space is a tree with each vertex representing an atom and edges representing the choice of *true* or *false* for a given atom. Each possible path from the root node to a leaf corresponds to an assignment of boolean values for atoms. An assignment that makes the formula *true* is called a *model*.

In order to build a model, a given search-based solver performs the following actions: *decide*, *propagate* and *backtrack*. For *decide*, it chooses an unassigned atom and assign to it *true* or *false*. In *propagate*, it deduces the consequences of such assignment in other parts of the formula. Given that a partial assignment leads the formula to *false*, it means that some earlier decision must be changed to a different value, than the solver *backtracks*. If *backtracking* is not possible, than it means that the formula is *unsatisfiable*.

Satisfiability Modulo Theories (SMT) extends the SAT problem, by making it possible to use in a given formula, besides boolean variables, predicates expressed in a given theory. An example is the theory of linear arithmetic, which is used in this work. In such a case, predicates represent linear arithmetic inequalities. An example of SMT formula is presented as follows.

$$(v_1 \leq v_2) \wedge (v_3 \vee (v_1 > v_2)) \quad (\text{C.1})$$

In such a case, variables represent either boolean or arithmetic values.

Solving an SMT formula is performed by integrating difference arithmetic solvers with SAT solvers. The key idea is to abstract the atoms in an SMT formula as boolean variables. Taking for example the formula presented in Equation C.1 it can be abstracted into the following formula.

$$p_1 \wedge (v_3 \vee p_2) \quad (\text{C.2})$$

where atoms $(v_1 \leq v_2)$ and $(v_1 > v_2)$ are replaced by boolean variables p_1 and p_2 , respectively.

The abstracted formula is treated as a regular SAT formula. If a solver finds it satisfiable, the difference arithmetic solver is used to check the model produced by the SAT solver. Given that the atoms are also found to be satisfiable by the arithmetic solver, the formula as a whole is satisfiable, otherwise the SAT solver needs to backtrack. Such process continues until a model is found or no backtrack is possible (thus the formula will be satisfiable or unsatisfiable, respectively). The resulting model, in this case will be a mapping from boolean and arithmetic values to the boolean and arithmetic variables contained in the formula.

C.2 Yices2

This section presents an overview of the SMT solver Yices2. The text here presented borrows and adapts from [Dutertre 2014].

Yices2 logic supports the primitive types `int` and `real` for arithmetic types, `bool` for boolean type among others¹. Terms in Yices may represent constants or formulas. The primitive terms supported are the arithmetic and boolean ones.

Yices maintains a global database of types and terms. Its API provides functions for constructing types and terms, pretty printing terms and etc. The central structure of Yices is the *context*. A context stores assertions to be checked for satisfiability. The same way as for types and terms, its API provides functions for creating, destroying and checking contexts, besides adding and removing assertions to contexts.

If the assertions in a context is satisfiable, then the solver can build a model of the context. Such model maps constants in assertions to concrete values of the same type of the constant. Once built, a model can be queried and examined independently from the context. Moreover, further changes in the context do not affect an already created model.

Yices also makes it possible for creating a backtracking point and backtracking to a previously created backtracking point. Such feature is interesting when some formulas in the context can be removed if the latter is unsatisfiable.

The declaration of types and terms in Yices2 can be performed using the SMT-LIB 2.0 syntax. Further information about the tool is available in its technical manual².

¹We shall focus here on the types and operations used in our *SMT* validation implementation.

²<http://yices.cs1.sri.com/papers/manual.pdf>

APPENDIX D - *RWT* Refinements

The *RWT* theory was first proposed in [dos Santos 2012, dos Santos et al. 2013a] for the representation of the behavior of multimedia documents. In its first version, *RWT* was able to represent the temporal layout of a given document through causal relations among event occurrences in document fragments.

This work [dos Santos et al. 2012a, dos Santos et al. 2013b, dos Santos et al. 2015a] presents a refinement of theory *RWT* mainly by better exploiting the Rewriting logic calculus and using set-rewriting theories (see Appendix B).

This appendix discusses those refinements made in theory *RWT*. Before presenting the refinements, Section D.1 briefly presents the first version of theory *RWT*, as defined in [dos Santos 2012, dos Santos et al. 2013a]. Following, Section D.2 discusses the refinements and Section D.3 presents test results that indicate performance improvements.

D.1 Previous *RWT* version

This section is based on the definition for the *RWT* theory presented in [dos Santos 2012]. There, a document d represented in theory *RWT* as d_{RWT} is given by a set of fragments (A), causal relations (L) and actions (Ia) that are executed as the document begins.

The reader should notice that, in the definition of theory *RWT* presented in [dos Santos 2012], fragments are called *anchors* and causal relations are called *links*. We have chosen not to change *anchor* and *link* for fragments and causal relations, respectively, to maintain the consistency of the text.

Theory *RWT* defined sort *anchor*, where an element of such sort is formed by its identification and a set of attributes that represent its state information as follows.

```

1  sort AnchorIdentInfo Attribute AttributeSet Anchor .
2  subsort Attribute < AttributeSet .
3  op <-|-> : AnchorIdentInfo AttributeSet -> Anchor [ctor] .
4

```

```

5  op att-value _ : String -> Attribute [ctor] .
6  op att-state _ : EventState -> Attribute [ctor] .
7  op pre-state _ : EventState -> Attribute [ctor] .
8  op pre-duration _ : Nat -> Attribute [ctor] .
9  op sel-state _ : EventState -> Attribute [ctor] .
10 op sel-pressedKey _ : EventKey -> Attribute [ctor] .

```

The state of an anchor presentation (prefix *pre*), selection (prefix *sel*) and attribution (prefix *att*) follows the state machine presented in Figure 4.1, which is presented again in Figure D.1.

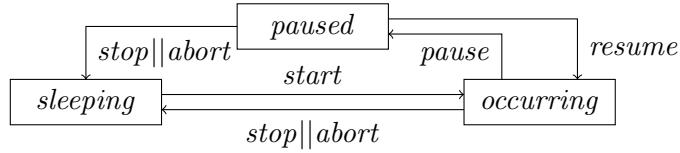


Figure D.1: State changes

In order to be able to model and reason over the occurrence of transitions in the state machine of Figure 4.1, transitions are represented as states in the first version of *RWT*. Thus, the set of states is extended with new states representing those transitions. Figure D.2 presents *RWT*'s extended state machine. The original states are represented in green, while the new “transient” states represented in yellow.

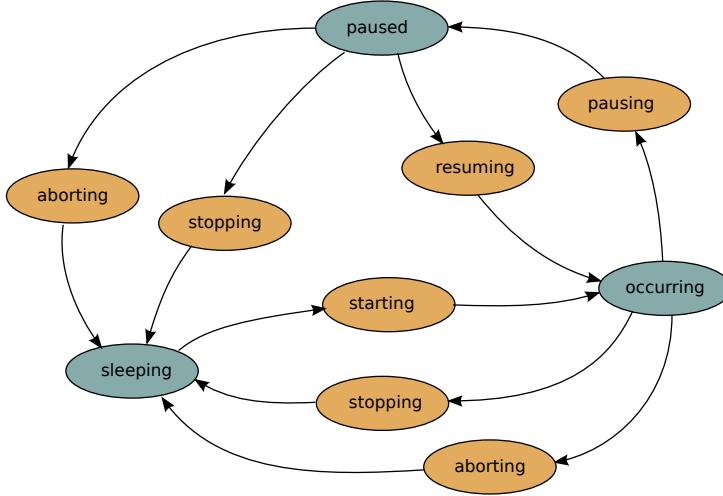


Figure D.2: Extended state machine

Sort *EventState* represents the set of states for the extended state machine. Sorts *EventType* and *EventKey* represent the types of state machines and a set of keys related to viewer selection, respectively. They are defined in *RWT* as follows.

```

1  sort EventState EventType EventKey .
2
3  ops presentation selection attribution : -> EventType [ctor] .
4  ops sleeping occurring paused stopping aborting starting pausing resuming : -> EventState [ctor] .
5  ops noKey RED GREEN BLUE YELLOW OK : -> EventKey .

```

The occurrence of an event, i.e., a transition in the state of a fragment presentation, selection or attribution is represented by sort *EventTransition* as follows.

```

1  sort StateTransition EventTransition .
2
3  ops start stop pause resume abort : -> StateTransition [ctor] .
4  op <--|-|-> : StateTransition EventType AnchorIdentInfo AttributeSet -> EventTransition [ctor] .
5  op value =_ : EventValue -> Attribute [ctor] .
6  op key =_ : EventKey -> Attribute [ctor] .

```

where the *value* attribute represents the value to be set in a document variable and the *key* attribute represents the key pressed in a viewer selection. The logic of the application of a transition over an anchor is represented by equations. For example, the equations for a *start* transition are defined as follows.

```

1  var D : DocContent .
2  var I : AnchorIdentInfo .
3  var A : AttributeSet .
4  var Es : EventState .
5  vars Ev Ev' : String .
6  vars Ek Ek' : EventKey .
7
8  ceq < start presentation | I | none > < I | pre-state = Es, A > D =
9    < I | pre-state = starting , A > D if Es == sleeping or Es == stopping or Es == aborting .
10 ceq < start attribution | I | value = Ev > < I | att-value = Ev', att-state = Es, A > D =
11   < I | att-value = Ev, att-state = starting , A > D if Es == sleeping or Es == stopping or Es
12   == aborting .
13 ceq < start selection | I | key = Ek > < I | sel-state = Es, sel-pressedKey = Ek', A > D =
14   < I | sel-state = starting , sel-pressedKey = Ek, A > D if Es == sleeping or Es == stopping
15   or Es == aborting .

```

A document, in Maude, is represented by the set of fragments that compose it, which is represented by sort *DocContent* as follows.

```

1  sorts DocContent .
2  subsort Anchor EventTransition < DocContent .
3
4  op none : -> DocContent [ctor] .
5  op -- : DocContent DocContent -> DocContent [ctor assoc comm id: none] .

```

A step in the document execution is represented in *RWT* with four steps. They are: (1) the increment of anchor clocks, (2) the evaluation of natural event occurrences, (3) the evaluation of links and (4) the evolving of states that represent transitions. Those steps application continues until the document ends.

$$\frac{a_i \in D \quad a'_i \in D' \quad a''_i \in D'' \quad a'''_i \in D'''}{\begin{array}{c} a_i \xrightarrow{\text{increment}}_1 a'_i \quad a'_i \xrightarrow{\text{natural}}_1 a''_i \quad a''_i \xrightarrow{\text{links}}_1 a'''_i \quad a'''_i \xrightarrow{\text{evolving}}_1 a'''_i \\ D \quad \xrightarrow{\text{SHM}}_1 D''' \end{array}} \neg ended(D)$$
(D.1)

The system configuration, represented by sort *DocState*, is given by *DocContent* plus a token to indicate the current document evolution step as follows.

```

1   sort EvolvingStep EvolvingToken DocContent DocState .
2   subsort EvolvingToken < DocContent .
3
4   ops formatter implicit explicit : -> EvolvingStep [ctor] .
5   op [-:-] : EvolvingStep Nat -> EvolvingToken [ctor] .
6
7   op _[_] : EvolvingInfo DocContent -> DocState .

```

Notice that the previous version of *RWT* defines three evolving steps: *formatter*, *implicit* and *explicit*. During the *formatter* step, it implements the evolving of states that represent transitions and the increment of anchor durations (steps *evolving* and *increment* in Equation D.1). During the *implicit* step, it evaluates natural event occurrences (step *natural* in Equation D.1). And during the *explicit* step, it evaluates links (step *links* in Equation D.1). The rules that model the change from one step to another are defined as follows.

```

1  var S : EvolvingStep .
2  var T : Nat .
3  var ET : EvolvingToken .
4
5  --- start the document simulation
6  eq run = [ explicit : 0 ] [(InitActions Document) links] .
7
8  crl [ explicit : T ] [D] => [ formatter : inc(T) ] [formatter-evolve-doc(D)] if not(ended(D)) and
9    docWillFormatterEvolve(D) .
10  ceq [ explicit : T ] [D] = [ formatter : inc(T) ] [formatter-evolve-doc(D)] if not(docIsEvolving(D))
11    and not(ended(D)) and not(docWillFormatterEvolve(D)) .
12
13  crl [ formatter : T ] [D] => [ implicit : T ] [mountEvolvingStep(T, D, EvolvingOccurSelection) D]
14    if not(ended(D)) and docWillImplicitEvolve(selectKey(T), D) .
15  ceq [ formatter : T ] [D] = [ implicit : T ] [mountEvolvingStep(T, D, EvolvingOccurSelection) D] if
16    not(docIsEvolving(D)) and not(ended(D)) and not(docWillImplicitEvolve(selectKey(T), D)) .
17
18  crl [ implicit : T ] [D] => [ explicit : T ] [links D] if not(ended(D)) and docWillExplicitEvolve
19    (D) .
20  ceq [ implicit : T ] [D] = [ explicit : T ] [links D] if not(docIsEvolving(D)) and not(ended(D))
21    and not(docWillExplicitEvolve(D)) .

```

At each step, *RWT* verifies if the document is still executing. If so, it applies the equations that describe the document behavior inside a step.

After all equations are applied, Maude tests if any modification in the document state will occur in the next step. If the document state will change, a rule is used to change the evolving step, and consequently, creating a new state in the transition system. If the document state will not change, an equation is used, so no new state is created.

Whenever an anchor is in a state representing a transition and no link may be applied, the *formatter* step evolves that state to one of the original states in Figure D.1. Besides, anchors in the *occurring* state have its clock incremented. The following functions are used to implement the *formatter* step.

```

1  op formatter-evolve-doc : DocContent -> DocContent .
2  op formatter-evolve-anchor : Anchor -> EvolvingToken .
3  op evolve-state : EventState -> EventState .

```

The natural event occurrences are evaluated at the same time, by equations that end an anchor presentation if the anchor duration was reached, end an anchor attribution once it is occurring and start an anchor selection once it is sleeping. The following functions are used to implement the *implicit* step.

```

1  op implicit-evolve-doc : DocContent -> DocContent .
2  op implicit-evolve-doc : DocContent EventKey -> DocContent .
3  op implicit-evolve : AnchorIdentInfo -> EvolvingToken .
4  op implicit-evolve : AnchorIdentInfo EventKey -> EvolvingToken .

```

Finally, a viewer selection defines the key that was pressed. This information is important since links may define a different document behavior depending on the key pressed. When a selection may occur, *RWT* chooses one element of set *EventKey* to represent the key pressed as follows.

```

1  op mountEvolvingStep : Nat DocContent Bool -> DocContent .
2  op selectKey : Nat -> EventKey .
3
4  eq mountEvolvingStep(T, D, true) = implicit-evolve-doc(D, selectKey(T)) .
5  eq mountEvolvingStep(T, D, false) = implicit-evolve-doc(D) .

```

Links, in *RWT*, are represented by equations that are applied over anchors whose state represents a transition, inducing the modification of the state of other anchors. Since equations are also used in a transition system state definition, a modification of the document state will be given by the application of all enabled links. Once no other link can be applied, the states that represent transitions are evolved to the ones in the original state machine.

Once the equations that represent document links depend on the document that will be validated, *RWT* does not define any behavior for the *explicit* step a priori and it shall be defined in a separate module providing document-specific information. Fragments of a module specifying document-specific information are presented as follows.

```

1  eq Document = < I1 | pre-state = sleeping, pre-duration = 0, sel-state = sleeping, sel-pressedKey =
   noKey > ...
2
3  eq links = explicit-evolve("L1") ...
4
5  eq InitActions = < start presentation | I1 > .
6
7  eq ET [ explicit-evolve("L1") < I1 | pre-state = starting, A > D ] =
   ET [ < start attribution | I2 | value = "yes" > < I1 | pre-state = starting, A > D ] .
8
9 ...
10
11 eq implicit-evolve(I1) < I1 | pre-state = occurring, pre-duration = 900, A > D =
   < stop presentation | I1 > < I1 | pre-state = occurring, pre-duration = 900, A > D .
12
13 ...
14
15 eq docWillExplicitEvolve(< I1 | pre-state = starting, A > D) = true .
16 ...
17
18 eq anchorWillImplicitEvolve(< I1 | pre-state = occurring, pre-duration = 900, A > ) = true .
19 ...

```

D.2 Theory refinements

Section D.1 briefly presented the previous version of theory *RWT*. The reader should notice four key aspects in such a specification, which are:

1. Each step of a document presentation is divided in three evolving steps and, for each evolving step, *RWT* first evaluates if some event will occur or not. If it will, then a rule is used, otherwise an equation is used for changing from one evolving step to another.
2. A document presentation evolves by incrementing anchor clocks by one.
3. Each evolving step is implemented by functions that traverse all anchors in a document changing their attributes.
4. After each increment in anchor clocks, selection may occur.

The first three characteristics have a side effect of diminishing *RWT*'s efficiency since it has to compute the document state for every increment in anchors clocks, and test if something will happen. Besides every possible modification in the document state is controlled by a set of functions. The fourth characteristic has the side effect of augmenting the state explosion problem, since several viewer selections may occur while an anchor is being presented.

In our current version of theory *RWT*, we tackle such problems mainly by adopting three solutions, which are:

1. Avoid calculating the document state for each time instant and doing so only when some event may occur.
2. Better exploiting the Rewriting logic calculus passing the control of the application of equations and rules that model the document execution to the Maude system.
3. Restrict the number of viewer interactions to a user-defined number.

The first solution is achieved by changing anchor clocks to countdown clocks. Thus, instead of starting it at zero and incrementing it at each document execution step, we start it with the anchor duration and decrement it at each document execution step. This simple change enables us to determine the next instant when an event will occur by

checking the first clock that will reach zero. Therefore, the presentation can jump to this next point and then calculate the document state after the occurrence of such event.

The second solution is achieved by redefining RWT as a set-rewrite theory. By doing so we are able to simplify the equations defining RWT 's behavior (see Section 4.3.1). Moreover, functions *discard* and *elapse* and rule *step* are constructed in a way that induces an order for the application of equations as discussed in the proof of Theorem 1. Therefore we do not need to use functions to control each step of the document execution.

The third solution is achieved by defining rules in the module containing document-specific information that represent viewer interaction. The amount of rules created will depend of the number of possible interactions defined by the user. Moreover, each interaction is defined to occur at a fraction of the anchor duration.

It is worth noticing that no information about the document execution is lost with the adoption of solutions 1 and 2. Jumping the document execution to the next instant when an event occurrence will occur do not represent a lost in information, since no change in the state of anchors shall occur during that jump, only the increment(or decrement) of anchor clocks. Some information may be lost by restricting the number of viewer interactions of an anchor, but we believe it is a tradeoff to be considered by the user. A more refined modeling of viewer interaction, by augmenting the number of possible interactions with an anchor, tends to augment the number of states and consequently the response time of validation or a more simple modeling of viewer interaction, by diminishing the number of possible interactions, with a faster validation.

D.3 Practical results

In Section D.2 we discussed the changes made in RWT to improve its efficiency and avoid the state explosion problem. Tests were conducted in order to investigate the gain obtained with such refinements. This section discusses the conducted tests together with their results.

Section 5.2 presented test results using RWT for validating three multimedia documents created by the NCL community. Test results are summarized in Table D.1, which is presented here to facilitate this section reading.

Table D.1 presents the number of properties validated for a given document, together with the whole time spent in the validation and the average time spent for validating each

Table D.1: Document sizes and validation time comparison

Document	“First João”	“Live More”	“Day’s Route”
Properties	23	31	33
	386ms	936ms	882ms
	16.78ms	30.19ms	26.72ms

property. It is worth noticing that, the bigger the document, in terms of fragments and causal relations, the bigger the time spent validating it.

The same tests were conducted with the “First João” and “Live More” documents for the previous version of *RWT*. The results are presented in Table D.2.

Table D.2: Document sizes and validation time comparison

Document	“First João”	“Live More”
Properties	23	31
	2.5s	5.5s
	108.69ms	177.42ms

Besides the refinements presented in this appendix, in our current version of *RWT*’s implementation we use additional fragments for representing media item position in space as discussed in Section 4.3.2. Thus, an increase in the number of fragments is expected for representing a given media item. We performed two tests to evaluate our approach. Test results are presented in Figure D.4.

In the first test we have a document d with two media items A and B . Both A and B start their presentation as d ’s presentation begins. Media A changes its position and size until it has the same position and size of B and then it remains that way. Document d ’s presentation ends when both A and B finish their presentation. Figure D.3 presents the spatio-temporal layout for this test.

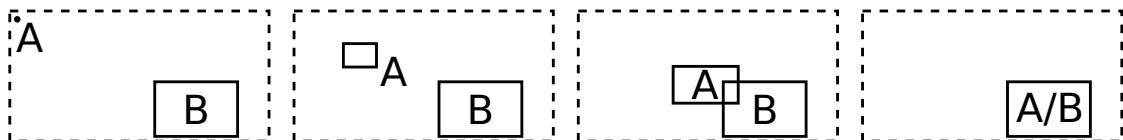


Figure D.3: Spatio-temporal layout for the first test

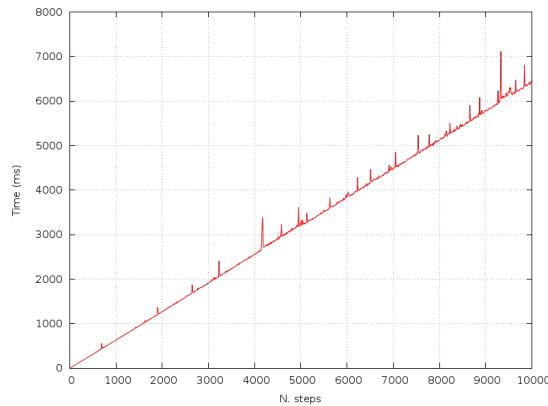
The first test consists in incrementing the number n of steps for changing A ’s position and size until it reaches the same position and size of B . For each value of n we ran the following Maude command and gathered the statistics provided by Maude.

```
1 red modelCheck(run , <>('A equal 'B)) .
```

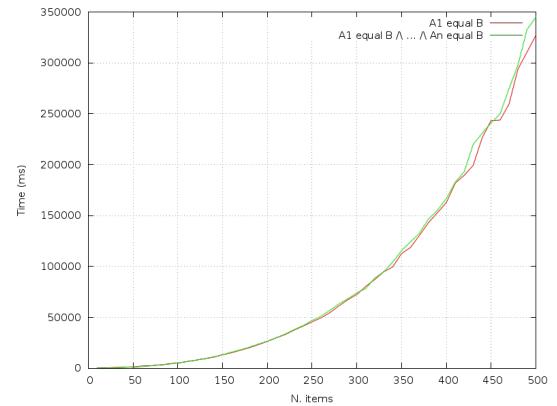
As the number of steps grows, the number of states in \mathcal{S}_{RWT} grows linearly with it. The impact in increasing the number of steps in the time Maude takes to perform the above command is presented in Figure D.4a. As it can be seen, time also increases linearly with the number of steps.

In the second test, we fixed the number of steps to 10 and increased the number n of media items inside the document changing their position. For each value of n we ran the following two Maude commands and gathered the statistics provided by Maude.

```
1 red modelCheck(run , <>('A1 equal 'B)) .
2 red modelCheck(run , <>('A1 equal 'B /\ ... /\ 'An equal 'B)) .
```



(a) Number of steps X time (ms)



(b) Number of items X time (ms)

Figure D.4: Test results

The impact in increasing the number of media items in the time Maude takes to perform the two commands above is presented in Figure D.4b. As it can be seen, time grows exponentially with the number of items and the size of the formula to be tested has almost no impact in time. Testing the execution of each test document alone indicated that approximately all the time spent by validating the above formula was spent by Maude in building the transition system where the temporal formulas were verified.

Comparing our approach presented in this thesis to our previous works indicates that time increases mostly because of the growth in the number of state machines required for representing the document's spatial layout. In our tests, for each state machine representing a media item, five more (four for the positioning attributes and one for the increment delay) were created. Comparing the time spent to run each document with a similar one, regarding the number of state machines, but without spatial information, indicates a small increase in time.

From the graph in Figure D.4a we see a linear growth of time related to the number of steps for changing the position and size of a media item. For a document with 10000

steps, the validation is performed in about 6.5 seconds. It is worth mentioning that, in common multimedia documents, movements of media items take a few seconds, thus, even with a precision of milliseconds we are still able to give an answer to the author in a reasonable time. In general, such a huge number of steps is not necessary and can be abstracted to decrease the overall duration of the validation.

APPENDIX E - Module RRWT

```

1 fmod INFNAT is
2   including NAT .
3
4   sort InfNat .
5   subsort Nat < InfNat .
6
7   op inf : -> InfNat [ctor] .
8
9   eq inf + I:InfNat = inf .
10  eq inf * I:InfNat = inf .
11  eq min(inf, I:InfNat) = I:InfNat .
12  eq sd(inf, I:InfNat) = inf .
13 endfm
14
15
16
17
18 fmod RNCLSORTS is
19   sorts MachineType MachineState UnitAtt EventTransition Action Component Conf .
20   subsort UnitAtt EventTransition Action < Component .
21 endfm
22
23
24
25
26 view Component from TRIV to RNCLSORTS is
27   sort Elt to Component .
28 endv
29
30
31
32
33 fmod RNCLSIG is
34   including INFNAT .
35   including STRING .
36   including QID .
37   including RNCLSORTS .
38   including SET{Component} .
39
40   sort ClockVal .
41   subsort InfNat < ClockVal .
42
43
44   ops pre sel att : -> MachineType [ctor] .
45   ops sleeping occurring paused : -> MachineState [ctor] .
46   op none : -> ClockVal [ctor] .
47
48   op value : Qid InfNat -> UnitAtt [ctor format(sg o)] .
49   op value : Qid String -> UnitAtt [ctor format(sg o)] .
50   op state : Qid MachineType MachineState -> UnitAtt [ctor format(sg o)] .
51   op occur : Qid MachineType InfNat -> UnitAtt [ctor format(sg o)] .
52   op clock : Qid MachineType ClockVal -> UnitAtt [ctor format(sg o)] .
53
54   ops init end hang halt return : Qid MachineType -> EventTransition [ctor format(sb! o)] .
55   op key : Qid String -> EventTransition [ctor format(sb o)] .
56   ops start stop abort pause resume : Qid MachineType -> Action [ctor format(sr! o)] .
57

```

```

58   op <-|-> : InfNat Set{Component} -> Conf [ctor] .
59
60   op dur : Qid MachineType -> InfNat .
61
62   op max : -> Nat .
63   op ini : -> Action .
64   op doc : -> Set{Component} .
65   op run : -> Conf .
66 endfm
67
68
69
70
71 mod RNCL is
72   including RNCLSIG .
73
74
75   vars n m : InfNat .
76   var id : Qid .
77   var mt : MachineType .
78
79   eq [action] : abort(id,mt), state(id,mt,occurring), clock(id,mt,m) =
80     state(id,mt,sleeping), clock(id,mt,none), hang(id,mt) .
81   eq [action] : abort(id,mt), state(id,mt,pause), clock(id,mt,m) =
82     state(id,mt,sleeping), clock(id,mt,none), hang(id,mt) .
83   eq [action] : abort(id,mt), state(id,mt,sleeping) = state(id,mt,sleeping) .
84
85   eq [action] : pause(id,mt), state(id,mt,occurring) =
86     state(id,mt,pause), halt(id,mt) .
87   eq [action] : pause(id,mt), state(id,mt,pause) = state(id,mt,pause) .
88   eq [action] : pause(id,mt), state(id,mt,sleeping) = state(id,mt,sleeping) .
89
90   eq [action] : resume(id,mt), state(id,mt,pause) =
91     state(id,mt,occurring), return(id,mt) .
92   eq [action] : resume(id,mt), state(id,mt,occurring) = state(id,mt,occurring) .
93   eq [action] : resume(id,mt), state(id,mt,sleeping) = state(id,mt,sleeping) .
94
95   eq [action] : start(id,mt), state(id,mt,sleeping), clock(id,mt,none) =
96     state(id,mt,occurring), clock(id,mt,dur(id,mt)), init(id,mt) .
97   eq [action] : start(id,mt), state(id,mt,occurring) = state(id,mt,occurring) .
98   eq [action] : start(id,mt), state(id,mt,pause) = state(id,mt,pause) .
99
100  eq [action] : stop(id,mt), state(id,mt,occurring), occur(id,mt,n), clock(id,mt,m) =
101    state(id,mt,sleeping), occur(id,mt,s n), clock(id,mt,none), end(id,mt) .
102  eq [action] : stop(id,mt), state(id,mt,pause), occur(id,mt,n), clock(id,mt,m) =
103    state(id,mt,sleeping), occur(id,mt,s n), clock(id,mt,none), end(id,mt) .
104  eq [action] : stop(id,mt), state(id,mt,sleeping) = state(id,mt,sleeping) .
105
106
107
108  eq [natural] : state(id,mt,occurring), occur(id,mt,n), clock(id,mt,0) =
109    state(id,mt,sleeping), occur(id,mt,s n), clock(id,mt,none), end(id,mt) .
110
111
112
113   var sc : Set{Component} .
114
115   op dt : Set{Component} -> InfNat .
116   eq dt( (clock(id,mt,m), state(id,mt,occurring), sc) ) = min(m, dt(sc)) .
117   eq dt(sc) = inf [owise] .
118
119   op elapse : Set{Component} InfNat -> Set{Component} .
120   eq elapse( (clock(id,mt,m), state(id,mt,occurring), sc), n ) =
121     clock(id,mt,sd(m,n)), elapse( (state(id,mt,occurring), sc), n) .
122   eq elapse(sc, n) = sc [owise] .
123
124
125   var cf : Conf .
126   var ev : EventTransition .
127   var ac : Action .
128
129   op discard : Set{Component} -> Set{Component} .
130   eq discard((ev, sc)) = discard(sc) .

```

```
131   eq discard((ac, sc)) = discard(sc) .
132   eq discard(sc) = sc [owise] .
133
134   op active : Set{Component} -> Bool .
135   eq active( (state(id,mt,occurring), sc) ) = true .
136   eq active(sc) = false [owise] .
137
138   op check : Set{Component} -> Bool .
139   eq check((ac, sc)) = false .
140   eq check((clock(id,mt,0), sc)) = false .
141   eq check(elapse(sc,n)) = false .
142   eq check(sc) = true [owise] .
143
144   crl [step] : < n | sc > => < n + dt(sc) | elapse(discard(sc), dt(sc)) > if check(sc) and active(sc)
145   and min(n,max) == n .
146
147   eq run = < 0 | ini , doc > .
148 endm
```

APPENDIX F - Document example

The appendix presents an example of multimedia document specified with the NCL language. This example is a real document gathered from the repository of NCL documents called NCL Club.

The document we shall use as an example here is a fragment of a document called “**First João**”. It presents an animation inspired in a chronicle about a famous Brazilian soccer player named *Garrincha*. At some point during the animation presentation the soccer shoes icon appear (Figure F.1a). If the viewer presses the red key of the remote control, a video of a kid thinking about shoes starts playing (Figure F.1b).



Figure F.1: First João spatio-temporal layout

Figure F.1 presented the spatial layout for a few relevant instants of the document execution. Listing F.1 presents the complete NCL code for the *First João* document.

```

1 <ncl id="JOAO">
2   <head>
3     <regionBase >
4       <region id="r_animation" left="0" top="0" width="1920" height="1080"/>
5       <region id="r_icon" left="1680" top="126" width="162" height="72"/>
6       <region id="r_shoes" left="288" top="648" width="480" height="270"/>
7     </regionBase>
8
9     <descriptorBase >
10    <descriptor id="d_animation" region="r_animation"/>
11    <descriptor id="d_icon" region="r_icon"/>
12    <descriptor id="d_shoes" region="r_shoes"/>
13  </descriptorBase>
14
15  <connectorBase >
16    <causalConnector id="onKeySelectionStopStart">
17      <connectorParam name="keyCode"/>
18      <simpleCondition role="onSelection" key="$keyCode"/>
19      <compoundAction operator="seq">
20        <simpleAction role="stop" qualifier="par"/>

```

```

21         <simpleAction role="start" qualifier="par"/>
22     </compoundAction>
23   </causalConnector>
24   <causalConnector id="onBeginStart">
25     <simpleCondition role="onBegin"/>
26     <simpleAction role="start"/>
27   </causalConnector>
28   <causalConnector id="onEndStop">
29     <simpleCondition role="onEnd"/>
30     <simpleAction role="stop"/>
31   </causalConnector>
32 </connectorBase>
33 </head>
34
35 <body>
36   <port id="entry" component="animation"/>
37
38   <media id="animation" descriptor="d_animation" src="animGar.avi">
39     <area id="segIcon" begin="45s" end="51s"/>
40   </media>
41
42   <context id="advert">
43     <port id="pIcon" component="icon"/>
44
45     <media id="icon" descriptor="d_icon" src="icon.png"/>
46     <media id="shoes" descriptor="d_shoes" src="shoes.avi"/>
47
48     <link id="lBegingShoes" xconnector="onKeySelectionStopStart">
49       <bind role="onSelection" component="icon">
50         <bindParam name="keyCode" value="RED"/>
51       </bind>
52       <bind role="set" component="animation" interface="bounds">
53         <bindParam name="var" value="5%,6.67%,45%,45%"/>
54       </bind>
55       <bind role="start" component="shoes"/>
56       <bind role="stop" component="icon"/>
57     </link>
58   </context>
59
60   <link id="lShowIcon" xconnector="onBeginStart">
61     <bind role="onBegin" component="animation" interface="segIcon"/>
62     <bind role="start" component="advert" interface="pIcon"/>
63   </link>
64   <link id="lHideIcon" xconnector="onEndStop">
65     <bind role="onEnd" component="animation" interface="segIcon"/>
66     <bind role="stop" component="advert" interface="pIcon"/>
67   </link>
68 </body>
69 </ncl>
```

Listing F.1: *First João NCL code*

The *First João* document defines seven media items, they are: (i) *animation* for the Garrincha animation video; (ii) *icon* for the shoes image; and (iii) *shoes* for the video of a kid thinking about shoes.

Link *lShowIcon* (line 58) starts the *icon* image and link *lHideIcon* (line 62) ends the *icon* image presentation, both synchronized with the *animation* video. Link *lBegingShoes* (line 46) describes the causal relation that starts the presentation of the *shoes* video and stop the presentation of the *icon* when it is selected by the viewer.

The *First João* document is represented in *RWT* by the Maude module presented in Listing F.2.

```

1 mod JOAO is
2   including RNCL .
3
4   eq doc = state('body,pre,sleeping), occur('body,pre,0), clock('body,pre,none),
5
6     state('animation,pre,sleeping), occur('animation,pre,0), clock('animation,pre,none),
7     state('animation_l,att,sleeping), occur('animation_l,att,0), clock('animation_l,att,none),
8     value('animation_l,0),
9     state('animation_t,att,sleeping), occur('animation_t,att,0), clock('animation_t,att,none),
10    value('animation_t,0),
11    state('animation_w,att,sleeping), occur('animation_w,att,0), clock('animation_w,att,none),
12    value('animation_w,1920),
13    state('animation_h,att,sleeping), occur('animation_h,att,0), clock('animation_h,att,none),
14    value('animation_h,1080),
15
16    state('segIcon,pre,sleeping), occur('segIcon,pre,0), clock('segIcon,pre,none),
17
18    state('advert,pre,sleeping), occur('advert,pre,0), clock('advert,pre,none),
19
20    state('icon,pre,sleeping), occur('icon,pre,0), clock('icon,pre,none),
21    state('icon_sel,sleeping), occur('icon_sel,0), clock('icon_sel,none),
22    state('icon_l,att,sleeping), occur('icon_l,att,0), clock('icon_l,att,none), value('icon_l,
23    ,1680),
24    state('icon_t,att,sleeping), occur('icon_t,att,0), clock('icon_t,att,none), value('icon_t,
25    ,126),
26    state('icon_w,att,sleeping), occur('icon_w,att,0), clock('icon_w,att,none), value('icon_w,
27    ,162),
28    state('icon_h,att,sleeping), occur('icon_h,att,0), clock('icon_h,att,none), value('icon_h,
29    ,72),
30
31    state('shoes,pre,sleeping), occur('shoes,pre,0), clock('shoes,pre,none),
32    state('shoes_l,att,sleeping), occur('shoes_l,att,0), clock('shoes_l,att,none), value('
33    shoes_l,288),
34    state('shoes_t,att,sleeping), occur('shoes_t,att,0), clock('shoes_t,att,none), value('
35    shoes_t,648),
36    state('shoes_w,att,sleeping), occur('shoes_w,att,0), clock('shoes_w,att,none), value('
37    shoes_w,480),
38    state('shoes_h,att,sleeping), occur('shoes_h,att,0), clock('shoes_h,att,none), value('
39    shoes_h,270),
40
41    state('segIcon+delay,pre,sleeping), occur('segIcon+delay,pre,0), clock('segIcon+delay,pre,
42    none),
43    state('icon+selec1,pre,sleeping), occur('icon+selec1,pre,0), clock('icon+selec1,pre,none),
44    state('icon+selec2,pre,sleeping), occur('icon+selec2,pre,0), clock('icon+selec2,pre,none),
45
46 eq ini = start('body,pre) .
47 eq max = 3600 .
48
49 eq dur('body,pre) = inf .
50
51 eq dur('animation,pre) = 71 .
52 eq dur('animation_t,att) = 0 .
53 eq dur('animation_l,att) = 0 .
54 eq dur('animation_w,att) = 0 .
55 eq dur('animation_h,att) = 0 .
56
57 eq dur('shoes,pre) = 13 .
58 eq dur('shoes_t,att) = 0 .
59 eq dur('shoes_l,att) = 0 .

```

```

60   eq dur('shoes_w,att) = 0 .
61   eq dur('shoes_h,att) = 0 .
62
63   eq dur('segIcon+delay,pre) = 45 .
64   eq dur('icon+selec1,pre) = 10 .
65   eq dur('icon+selec2,pre) = 20 .
66
67
68   eq left('animation) = 'animation_l .
69   eq left('icon) = 'icon_l .
70   eq left('shoes) = 'shoes_l .
71
72   eq top('animation) = 'animation_t .
73   eq top('icon) = 'icon_t .
74   eq top('shoes) = 'shoes_t .
75
76   eq width('animation) = 'animation_w .
77   eq width('icon) = 'icon_w .
78   eq width('shoes) = 'shoes_w .
79
80   eq height('animation) = 'animation_h .
81   eq height('icon) = 'icon_h .
82   eq height('shoes) = 'shoes_h .
83
84
85   eq [lBegingShoes] : end('icon,sel), key('icon,"RED"), value('animation_l,L:InfNat), value('
     animation_t,T:InfNat), value('animation_w,W:InfNat), value('animation_h,H:InfNat) = stop('icon,pre
     ), start('shoes,pre), start('animation_l,att), value('animation_l,96), start('animation_t,att),
     value('animation_t,72), start('animation_w,att), value('animation_w,864), start('animation_h,att),
     value('animation_h,486) .
86
87   eq [lShowIcon] : init('segIcon,pre) = start('icon,pre) .
88   eq [lHideIcon] : end('segIcon,pre) = stop('icon,pre) .
89
90   eq init('animation,pre) = start('segIcon+delay,pre) .
91   eq end('segIcon+delay,pre) = start('segIcon,pre) .
92
93   eq init('icon,pre) = start('icon+selec1,pre), start('icon+selec2,pre) .
94   eq end('icon,pre) = stop('icon+selec1,pre), stop('icon+selec2,pre) .
95
96   eq init('body,pre) = start('animation,pre) .
97   eq end('body,pre) = stop('advert,pre), stop('animation,pre) .
98   ceq < T:InfNat | state('body,pre,occurring), occur('body,pre,O:InfNat), clock('body,pre,C:InfNat), S
     :Set{Component} > =
     < T:InfNat | state('body,pre,sleeping), occur('body,pre,s O:InfNat), clock('body,pre,none), end
     ('body,pre), S:Set{Component} >
     if state('advert,pre,sleeping), state('animation,pre,sleeping), S1:Set{Component} := S:Set{
     Component} .
99
100  ceq < T:InfNat | state('body,pre,sleeping), clock('body,pre,C:InfNat), S:Set{Component} > =
     < T:InfNat | state('body,pre,occurring), clock('body,pre,dur('body,pre)), init('body,pre), S:
     Set{Component} >
     if state(Q:Qid,pre,occurring), S1:Set{Component} := S:Set{Component} /\ (Q:Qid == 'advert) or (Q:
     :Qid == 'animation) .
101
102  eq init('advert,pre) = start('icon,pre) .
103  eq end('advert,pre) = stop('icon,pre), stop('shoes,pre) .
104  ceq < T:InfNat | state('advert,pre,occurring), occur('advert,pre,O:InfNat), clock('advert,pre,C:
     InfNat), S:Set{Component} > =
     < T:InfNat | state('advert,pre,sleeping), occur('advert,pre,s O:InfNat), clock('advert,pre,none)
     , end('advert,pre), S:Set{Component} >
     if state('icon,pre,sleeping), state('shoes,pre,sleeping), S1:Set{Component} := S:Set{Component}
105
106  ceq < T:InfNat | state('advert,pre,sleeping), clock('advert,pre,C:InfNat), S:Set{Component} > =
     < T:InfNat | state('advert,pre,occurring), clock('advert,pre,dur('advert,pre)), init('advert,pre
     ), S:Set{Component} >
     if state(Q:Qid,pre,occurring), S1:Set{Component} := S:Set{Component} /\ (Q:Qid == 'icon) or (Q:
     :Qid == 'shoes) .
107
108
109  r1 [icon+selec:RED] : < T:InfNat | end('icon+selec1,pre), S:Set{Component} > =>
     < T:InfNat | discard((start('icon,sel), key('icon,"RED"), S:Set{Component})) >
     .
110
111
112
113
114
115
116
117

```

```

118   r1 [icon+selec:RED] : < T:InfNat | end('icon+selec2',pre), S:Set{Component} > =>
119           < T:InfNat | discard((start('icon',sel), key('icon',"RED")), S:Set{Component})) >
120
121 endm

```

Listing F.2: *First João RWT code*

The Maude module presented in Listing F.2, is used for the document validation in its metarepresented version metarepresented as presented in Listing F.3.


```

72   ceq '<-|->[T:InfNat ,_-,-[S:Set{'Component'},'state['body.Qid,'pre.MachineType,'occurring.
    MachineState], 'occur['body.Qid,'pre.MachineType,'O:InfNat], 'clock['body.Qid,'pre.MachineType,'C:InfNat]]] =
73     '<-|->[T:InfNat ,_-,-[_-,-[_-,-[S:Set{'Component'},'end['body.Qid,'pre.
    MachineType]], 'clock['body.Qid,'pre.MachineType,'none.ClockVal]], 'occur['body.Qid,'pre.
    MachineType,'O:InfNat]]], 'state['body.Qid,'pre.MachineType,'sleeping.MachineState]]]
74   if '_,-[S1:Set{'Component'},'state['advert.Qid,'pre.MachineType,'sleeping.MachineState],
    'state['animation.Qid,'pre.MachineType,'sleeping.MachineState]] := 'S:Set{'Component'} [none].
75   ceq '<-|->[T:InfNat ,_-,-[S:Set{'Component'},'state['body.Qid,'pre.MachineType,'sleeping.
    MachineState], 'clock['body.Qid,'pre.MachineType,'C:InfNat]]] =
76     '<-|->[T:InfNat ,_-,-[_-,-[S:Set{'Component'},'init['body.Qid,'pre.MachineType]],
    'clock['body.Qid,'pre.MachineType,'dur['body.Qid,'pre.MachineType]]], 'state['body.Qid,'pre.
    MachineType,'occurring.MachineState]]
77   if '_,-[S1:Set{'Component'},'state['Q:Qid,'pre.MachineType,'occurring.MachineState]] := 'S:Set{'Component'} /\_.
    or_['==-[Q:Qid,'advert.Qid], '==-[Q:Qid,'animation.Qid]] = 'true.Bool [none].
78
79   eq 'init['advert.Qid,'pre.MachineType] = 'start['icon.Qid,'pre.MachineType] [none].
80   eq 'end['advert.Qid,'pre.MachineType] = '_,-[stop['icon.Qid,'pre.MachineType], 'stop['shoes.
    Qid,'pre.MachineType]] [none].
81   ceq '<-|->[T:InfNat ,_-,-[S:Set{'Component'},'state['advert.Qid,'pre.MachineType,'occurring.
    MachineState], 'occur['advert.Qid,'pre.MachineType,'O:InfNat], 'clock['advert.Qid,'pre.
    MachineType,'C:InfNat]]] =
82     '<-|->[T:InfNat ,_-,-[_-,-[_-,-[S:Set{'Component'},'end['advert.Qid,'pre.
    MachineType]], 'clock['advert.Qid,'pre.MachineType,'none.ClockVal]], 'occur['advert.Qid,'pre.
    MachineType,'O:InfNat]]], 'state['advert.Qid,'pre.MachineType,'sleeping.MachineState]]
83   if '_,-[S1:Set{'Component'},'state['icon.Qid,'pre.MachineType,'sleeping.MachineState],
    'state['shoes.Qid,'pre.MachineType,'sleeping.MachineState]] := 'S:Set{'Component'} [none].
84   ceq '<-|->[T:InfNat ,_-,-[S:Set{'Component'},'state['advert.Qid,'pre.MachineType,'sleeping.
    MachineState], 'clock['advert.Qid,'pre.MachineType,'C:InfNat]]] =
85     '<-|->[T:InfNat ,_-,-[_-,-[S:Set{'Component'},'init['advert.Qid,'pre.MachineType]],
    'clock['advert.Qid,'pre.MachineType,'dur['advert.Qid,'pre.MachineType]]], 'state['advert.
    Qid,'pre.MachineType,'occurring.MachineState]]
86   if '_,-[S1:Set{'Component'},'state['Q:Qid,'pre.MachineType,'occurring.MachineState]] := 'S.
    :Set{'Component'} /\_.
    or_['==-[Q:Qid,'icon.Qid], '==-[Q:Qid,'shoes.Qid]] = 'true.Bool [none].
87
88
89   rl '<-|->[T:InfNat ,_-,-[S:Set{'Component'},'end['icon+selec1.Qid,'pre.MachineType]]] =>
90     '<-|->[T:InfNat ,discard[_,-[_,-,S:Set{'Component'},'key['icon.Qid,"RED".String]], ,
    start['icon.Qid,'sel.MachineType]]] [label('icon+selec:RED')].
91
92   rl '<-|->[T:InfNat ,_-,-[S:Set{'Component'},'end['icon+selec2.Qid,'pre.MachineType]]] =>
93     '<-|->[T:InfNat ,discard[_,-[_,-,S:Set{'Component'},'key['icon.Qid,"RED".String]], ,
    start['icon.Qid,'sel.MachineType]]] [label('icon+selec:RED')].
94 endm

```

Listing F.3: *First João module metarepresentation*

As for the *SMT* validation, the *First João* document shall be represented as an *SMT* formula. The formula representing it is constructed from the conjunction of all subformulas sent to the *SMT* solver. The set of *SMT* subformulas representing the *First João* document is presented in Listing F.4.

```

1   I > 0
2   T ≤ I
3   canvas:tbeg ≤ T ≤ canvas:tend
4
5   --- Canvas definition
6   canvas:tmid = (canvas:tbeg + canvas:tend)/2
7   canvas:tend = canvas:tbeg + canvas:texp
8   canvas:tbeg = 0
9   canvas:texp > 0
10  canvas:tend ≤ I
11  canvas:tplc
12  canvas:xplc
13  canvas:xmid = (canvas:xbeg + canvas:xend)/2
14  canvas:xend = canvas:xbeg + canvas:xexp

```

```

15   canvas:ymid = (canvas:ybeg + canvas:yend)/2
16   canvas:yend = canvas:ybeg + canvas:yexp
17
18   -- Animation definition
19   animation:tmid = (animation:tbeg + animation:tend)/2
20   animation:tend = animation:tbeg + animation:texp
21   animation:tbeg ≥ canvas:tbeg
22   animation:tend ≤ canvas:tend
23   {(animation:tplc ∧ (T ≥ animation:tbeg) ∧ (T ≤ animation:tend) ∧ animation:splc)
24     ∨ ((¬animation:tplc ∨ (T < animation:tbeg) ∨ (T > animation:tend)) ∧ ¬animation:splc)}
25   animation:xmid = (animation:xbeg + animation:xend)/2
26   animation:xend = animation:xbeg + animation:xexp
27   animation:ymid = (animation:ybeg + animation:yend)/2
28   animation:yend = animation:ybeg + animation:yexp
29   segIcon:tmid = (segIcon:tbeg + segIcon:tend)/2
30
31   -- Icon definition
32   icon:tmid = (icon:tbeg + icon:tend)/2
33   icon:tbeg < icon:tend
34   icon:tbeg ≥ canvas:tbeg
35   icon:tend ≤ canvas:tend
36   {(icon:tplc ∧ (T ≥ icon:tbeg) ∧ (T ≤ icon:tend) ∧ icon:splc)
37     ∨ ((¬icon:tplc ∨ (T < icon:tbeg) ∨ (T > icon:tend)) ∧ ¬icon:splc)}
38   icon:xmid = (icon:xbeg + icon:xend)/2
39   icon:xend = icon:xbeg + icon:xexp
40   icon:ymid = (icon:ybeg + icon:yend)/2
41   icon:yend = icon:ybeg + icon:yexp
42   icons:tmid = (icons:tbeg + icons:tend)/2
43   icons:tend = icons:tbeg + icons:texp
44
45   -- Shoes definition
46   shoes:tmid = (shoes:tbeg + shoes:tend)/2
47   shoes:tend = shoes:tbeg + shoes:texp
48   shoes:tbeg ≥ canvas:tbeg
49   shoes:tend ≤ canvas:tend
50   {(shoes:tplc ∧ (T ≥ shoes:tbeg) ∧ (T ≤ shoes:tend) ∧ shoes:splc)
51     ∨ ((¬shoes:tplc ∨ (T < shoes:tbeg) ∨ (T > shoes:tend)) ∧ ¬shoes:splc)}
52   shoes:xmid = ((shoes:xbeg + shoes:xend)/2
53   shoes:xend = (shoes:xbeg + shoes:xexp
54   shoes:ymid = ((shoes:ybeg + shoes:yend)/2
55   shoes:yend = (shoes:ybeg + shoes:yexp
56
57   -- Canvas size
58   (canvas:xbeg = 0) ∧ (canvas:xexp = 1920) ∧ (canvas:ybeg = 0) ∧ (canvas:yexp = 1080)
59
60   -- Animation / SegIcon relation
61   animation:texp = 71
62   segIcon:tbeg = animation:tbeg + 45
63   segIcon:tend = animation:tbeg + 51
64   ((animation:tplc ∧ segIcon:tplc) ∨ (¬animation:tplc ∧ ¬segIcon:tplc))
65
66   -- Icon size
67   icon:texp = 6
68   (icon:xbeg = 560) ∧ (icon:xexp = 54) ∧ (icon:ybeg = 55) ∧ (icon:yexp = 32)
69
70   -- Icon selection
71   icons:tplc → ((icon:tbeg ≤ icons:tbeg) ∧ (icons:tend < icon:tend))
72
73   -- Shoes size
74   (shoes:xbeg = 96) ∧ (shoes:xexp = 160) ∧ (shoes:ybeg = 288) ∧ (shoes:yexp = 120)
75   shoes:texp = 13
76
77   -- Port ‘‘entry’’,
78   animation:tplc ∧ (animation:tbeg = canvas:tbeg)
79
80   -- Link ‘‘lShowIcon’’
81   segIcon:tplc ↔ icon:tplc
82   icon:tplc → (icon:tbeg = segIcon:tbeg)
83
84   -- Link ‘‘lBegingShoes’’
85   icons:tplc ↔ shoes:tplc
86   icons:tplc → ((icon:tend = icons:tend) ∧ (shoes:tbeg = icons:tbeg))
87   {(shoes:splc ∧ (animation:xbeg = 96) ∧ (animation:xexp = 864) ∧ (animation:ybeg = 72) ∧ (animation:yexp = 486)) ∨

```

```
88 ( $\neg shoes:s^{plc} \wedge (animation:x^{beg} = 0) \wedge (animation:x^{exp} = 1920) \wedge (animation:y^{beg} = 0) \wedge (animation:y^{exp} = 1080))\}$ 
89
90 --- Link ``lHideIcon''
91  $\neg icons:t^{plc} \rightarrow (icon:t^{end} = segIcon:t^{end})$ 
```

Listing F.4: *First João SMT code*