N^o d'ordre: 4588

THÈSE

PRÉSENTÉ À

L'UNIVERSITÉ BORDEAUX I

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

Par **Mauricio TORO**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ: INFORMATIQUE

Structured interactive scores: From a structural description of a multimedia scenario to a real-time capable implementation with formal semantics

Soutenue le: 25/09/2012

Rapporteurs:

M. Gérard ASSAYAG	CR CNRS, IRCAM
M. Jean-Michel COUVREUR	Professeur des Universités, Université d'Orléans

Devant la comission d'examen composée de:

M. Gérard ASSAYAG	CR CNRS, IRCAM	Rapporteur
M. Jean-Michel COUVREUR	Professeur, Université d'Orléans	Rapporteur
Mme. Myriam DESAINTE-CATHERINE	Professeur, Université de Bordeaux 1	Directeur de thèse
M. Camilo RUEDA	Professeur, Universidad Javeriana Cali	Co-encadrant
M. Jean-Philippe DOMENGER	Professeur, Université de Bordeaux 1	Président
M. David Janin	MCF, Université de Bordeaux 1	Examinateur
M. Arshia Cont	Chercheur, IRCAM	Examinateur

– 2012 –

Acknowledgments

"Happiness only real when shared"
"Le bonheur n'est réel que lorsqu'il est partagé"
"La felicidad sólo es verdadera si es compartida."
–Christopher McCandless

In first place I would like to thank my beloved parents Lucía and Gabriel, who have always supported me in my choice of an academic life, my Ph.D. and my research.

Afterwards, I want to thank Myriam Desainte-Catherine, my thesis supervisor, who encouraged me to do research in this delightful topic, to collaborate with other scientists from our discipline and other disciplines, who invited me to several concerts of Electroacoustic and Experimental music, and who guided me through my Ph.D. I learned from her how to divide time for research, writing, teaching, supervision, without disregarding family life and life itself! I deeply admire her dedication to research, her team-work skills, her ability to write grants and her capability to motivate people towards research.

I also own a lot to Camilo Rueda! Camilo not only co-supervised my Ph.D thesis, but also supervised my B.Sc thesis and my internship. He started me into the research life path and I will always be in debt with him. I learned from him how to do research, how to collaborate with other researchers, how to write papers, among many other skills. Furthermore, I deeply admire from him his humbleness, how he supports research in my home country, and the effort he has put to train researchers and give value to our research group AVISPA. I also want to thank Camilo for inviting me twice during my Ph.D to Colombia to work on model checking.

In addition to my parents and my supervisors, I also want to thank the people who contributed to this research with comments on my research, papers and the manuscript itself. First, my coauthors Julien Castet, Antoine Allombert, Carlos Olarte, Frank Valencia, Gerardo Sarria, Yann Orlarey, Pascal Baltazar and David Janin. I learned from them a lot about teamwork, how to write papers and how to conduct research in general. In addition, there is people with whom I had the chance to discuss during my doctorate and their comments will be forever part of this manuscript: Andrés Aristizábal, Víctor Rivera, Julián Gutierrez, Mathias Robine, Alexander Heussner, Alain Griffault , Jérôme Leroux, Jean Haury, Joseph Laralde, Grégoire Sutre, Charlotte Truchet, Arshia Cont and Carlos Agón. In addition, special thanks to Matt Wright and Andrew Brown for inviting me to give talks about my research in their universities in summer 2011 and for fruitful discussions, and to Frank Valencia for inviting me to work with him at Lix for one week in November 2010.

I also want to thank my brothers Carlos and Rodrigo, and my friends Sergio Mercado, Juan Felipe Gallego, Juan Carlos Mejía, Julián Valdés, Anaïs Labeyrie, Sophie Cadet, Jesús Pozuelo, Alejandra Muñoz, Estefanía Ruiz, Marcela Ñañez, Andrés Porras, Carmen Giraldo, Claire Jacquout, Andrés Gonzales, Claudia Rincón, Elior Gallon, Iréne Aminata Villaret, Dunia Urrego, Ana Maldonado, Liliam Sandoval, Claudia Valencia, Maïté Abadie, Emmanuelle Lagar, Karina Vargas, Javier Arnaez, Santiago Alfaro, Diego Useche, Diego

Parra, Jorge Rodas, Wilmar Cardona, Luis Giraldo, and many others I may forget at the moment. You all supported me through my Ph.D. We spent very good times during my holidays, evenings and weekends!

I also want to thank my first and second year classmates from room CVT for showing me the laboratory, sharing lunch with me at *Resto U*, inviting me to dinners at *La Raclette*, teaching me LaTex, sharing their experiences about conducting research, living in France and life itself, and, off course, for those good times at appetizers: Jonathan Ouoba, Jigar Solanki, Rémi Laplace, Jérémie Albert, Renaud Tabary, Cyril Cassagnes, Hugo Balacey, Tegawendé Bissyandé and Damien Dubernet. In addition, I want to thank to my young researcher fellows that supported me during my visits to Colombia: Jheyson Vargas, Andrés Barco, Jairo Velazco, Jaime Arias, Laura Pérez, Natalia Villegas, Pablo Muriano and Claudia Oviedo. Furthermore, thanks to my friend Yousouf Oualhadj –with whom I shared an office during third year– for nice discussions about future career, thesis, and life itself.

A special thanks to the reviewers of my thesis Gérard Assayag and Jean-Michel Couvreur for their comments on the first version of this manuscript, to all the members of the jury, and to the reviewers of all our peer-reviewed publications.

Last but not least, the people from the administrative services, I would have never finished this thesis without you: Philippe Bias, Lebna Mizani, Brigitte Cudeville, Auriane Dantes, Cathy Roubineau, Christine Parison, Maïté Labrousse and Sylvie Le Laurain in France, and Monica Posso in Colombia, among others I may forget at this time.

Mauricio Toro-Bermudez
Bordeaux, 19th September, 2012.

Abstract

Technology has shaped the way on which we compose and produce music: Notably, the invention of microphones and computers pushed the development of new music styles in the 20th century. In fact, several artistic domains have been benefiting from such technology developments; for instance, Experimental music, non-linear multimedia, Electroacoustic music, and interactive multimedia. In this dissertation, we focus on interactive multimedia.

Interactive multimedia deals with the design of scenarios where multimedia content and interactive events are handled by computer programs. Examples of such scenarios are multimedia art installations, interactive museum exhibitions, some Electroacoustic music pieces, and some Experimental music pieces. Unfortunately, most interactive multimedia scenarios are based on informal specifications, thus it is not possible to formally verify properties of such systems. We advocate the need of a general and formal model.

Interactive scores is a formalism to describe interactive multimedia scenarios. We propose new semantics for interactive scores based on timed event structures. With such a semantics, we can specify properties for the system, in particular, properties about traces, which are difficult to specify as constraints. In fact, constraints are an important part of the semantic model of interactive scores because the formalism is based on temporal constraints among the objects of the scenario. We also present an operational semantics of interactive scores based on the non-deterministic timed concurrent constraint (ntcc) calculus and we relate such a semantics to the timed event structures semantics. With the operational semantics, we formally describe the behavior of a score whose temporal object durations can be arbitrary integer intervals.

The operational semantics is obtained from the timed event structures semantics of the score. To provide such a translation, we first define the normal form of a timed event structure in which events related with zero-duration delays are collapsed into a single one. We also define the notion of dispatchable timed event structures: Event structures such that its constraint graph can be dispatched by relying only on local propagation.

We believe that operational semantics in ntcc offers some advantages over existing Petri nets semantics for interactive scores; for instance, the duration of the temporal objects can be arbitrary integer intervals, whereas in previous models of interactive scores, such durations can only be intervals to represent equalities and inequalities.

In this dissertation, we also introduce two extensions of the formalism of interactive scores: (1) one to handle audio processing using the Fast AUdio Stream (Faust) language and (2) another one to handle conditional branching, allowing designers to specify choices and loops. For the first extension, we present a timed event structures semantics and ideas on how to define operational semantics. For the second extension, we present an implementation and results comparing the average relative jitter of an implementation of an arpeggio based on Karplus-Strong with respect to existing implementations of Karplus written in Pure Data. We also define a XML file format for interactive scores and for the conditional branching extension. A file format is crucial to assure the persistence of the scores.

Ntcc models of interactive scores are executed using Ntccrt, a real-time capable

interpreter for ntcc. They can also be verified automatically using ntccMC, a bounded-time automata based model checker for ntcc which we introduce in this dissertation. Using ntccMC, we can verify properties expressed on constraint linear-time logic. Ntcc has been used in the past, not only for multimedia interaction models, but also for system biology, security protocols and robots. We argue that simulation and verification tools for ntcc are useful for interactive score and they are also useful for applications in other domains.

KEYWORDS: Multimedia interaction, timed event structures, concurrent constraint programming, ntcc, signal processing, temporal constraint programming.

Résumé

La plupart des scénarios multimédia interactifs sont basés sur des spécifications informelles, il n'est donc pas possible de vérifier formellement des propriétés de ces systèmes. Nous préconisons la nécessité d'un modèle général et formel. Partitions interactives est un formalisme pour décrire des scénarios multimédia interactifs.

Nous proposons une nouvelle sémantique pour les partitions interactives basée sur les structures d'événements temporisés. Avec une telle sémantique, nous pouvons spécifier des propriétés pour le système, en particulier, des propriétés sur les traces, qui sont difficiles à préciser avec la programmation par contraintes. Nous présentons également une sémantique opérationnelle des partitions interactives basée sur le calcul non-déterministe, temporisé, concurrent, par contraintes (ntcc) et nous rapportons la sémantique operationelle à la semantique en structures d'événements temporisés. Avec la sémantique opérationnelle, nous pouvons décrire formellement le comportement d'un scenario dont les durées des objets temporels peuvent être des intervalles d'entiers arbitraires.

La sémantique opérationnelle est obtenue à partir de la sémantique en structures d'événements temporisés de la partition interactive. Pour fournir une telle traduction, nous avons d'abord défini la forme normale d'une structure d'événements temporisés, dans laquel les événements liés avec une durée zéro sont regroupés en un seul. Nous avons également défini la notion de structures d'événements temporisés répartissables, de telle sorte que son graphe de contraintes peut être expédié en se fondant uniquement sur la propagation locale.

Nous croyons que la sémantique opérationnelle basée sur ntcc offre certains avantages par rapport à la sémantique des partitions interactives basée sur des réseaux de Petri; par exemple, les durées des objets temporels peuvent être des intervalles d'entiers arbitraires, tandis que dans la plupart des modèles de partitions interactives, les intervalles ne peut être utilisés que pour représenter les relations telles que l'égalité et les inégalités.

Nos modèles ntcc de partitions interactives sont exécutés en utilisant Ntccrt, un interprète temps réel pour ntcc. Nos modèles peuvent également être vérifiés automatiquement en utilisant ntccMC, un verificateur pour ntcc, de temps borné, basée sur les automates finis, que nous introduisons dans cette thèse. En utilisant ntccMC, nous pouvons vérifier des propriétés de logique de temps linéaire avec des contrantes (CLTL).

Dans cette thèse, nous introduisons deux extensions du formalisme de partitions interactives: (1) l'une pour gérer le traitement audio en utilisant le langage de programmation français Faust et (2) l'autre pour traiter des condition et des branchements, permettant de spécifier des choix et des boucles. Pour la première extension, nous présentons une sémantique basée sur les structures d'événements temporisés et des idées sur la façon de définir une sémantique opérationnelle. Pour la deuxième extension, nous présentons une mise en œuvre et la comparaison des résultats du jitter relative moyenne d'une implémentation d'un arpège base sur l'algorithme de Karplus-Strong par rapport aux implémentations existants écrits dans Pure Data. Nous définissons aussi un format de sauvegarde XML pour les partitions interactives et pour la extension avec branchement

conditionnel. Un format de sauvegarde est crucial pour assurer la persistance des partitions.

MOTS-CLÉS: interaction multimédia, structures d'événements temporisés, programmation concurrente par contraintes, ntcc, CCP, traitement du signal, programmation par contraintes temporelles.

Contents

Part I

Introduction

CHAPTER 1

Motivation

Technology has shaped the way on which we compose and produce music: Notably, the invention of microphones, magnetic tapes, amplifiers and computers pushed the development of new music styles in the 20th century. In fact, several artistic domains have been benefiting from such technology developments; for instance, *Experimental music*, *non-linear multimedia*, *Electroacoustic music*, and *interactive multimedia*.

Experimental music is composed in such a way that its outcome is often unforeseeable; for instance, it may contain random generated tones, computer-generated content, variable-duration notes and "free" content. It may also include atonal melodies and microtones.

Another domain is *non-linear multimedia*, in which the scenario is divided in parts whose order can be chosen at execution time. We will use the term "non-linear" music in that sense. Non-linear music exists from many centuries ago; for instance, Mozart's minuets in which the order of work's musical material was determined by coin-tosses.

Electroacoustic music was originated by the incorporation of electronic sound production into compositional practice. It subsumes styles such as *musique concrète* (French for *concrete music*), *Acousmatic music*, *musique mixte* (French for "mixed" music) and *Electronic music*. Note that Electroacoustic and Experimental music are not mutually exclusive: a piece can belong to both styles or to a single one, for instance, Experimental music explores composition with microtones which does not incorporate electronic sounds.

Interactive multimedia deals with the design of scenarios where multimedia content and interactive events are handled by computer programs. Examples of such scenarios are multimedia art installations, interactive museum exhibitions, some Electroacoustic music pieces, and some Experimental music pieces.

In what follows we briefly explain Experimental music, non-linear multimedia, Electroacoustic music and interactive multimedia. In this thesis we will focus on interactive multimedia. We are interested in Electroacoustic, Experimental and non-linear music that is interactive. In this chapter, we introduce the problems that arise when designers and composers want to write a score for interactive multimedia, and the problems with existing computer tools to compose and perform interactive multimedia; afterwards, we briefly describe some background concepts and we propose a solution based on the formalism of interactive scores.

1.1 Some Artistic Domains Shaped by Technology

In this section we briefly define Experimental music, non-linear multimedia, Electroacoustic music and interactive multimedia. To clarify the classification of these domains, we present a Venn's diagram in Figure 1.1: The diagram shows the intersection between the

different domains. Figure 1.1 includes *multimedia art installations*, which are an interesting subset of interactive multimedia; and *Tape music*, which is a subset of Electroacoustic music that is linear (i.e., parts have a fixed order) and is not interactive.

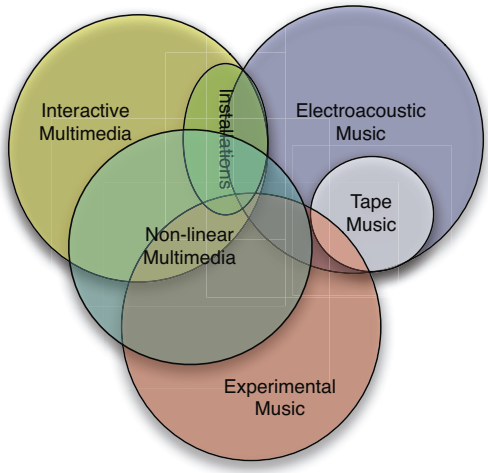

Figure 1.1: Intersection between Electroacoustic music, non-linear multimedia, Experimental music and interactive multimedia.

Experimental music. Nyman argues that, in Experimental music, a score may no longer represent a sound by the means of western music notation [Nyman 1999]: Composers may provide the performer the means of making calculations to determine the nature, timing and spacing of sounds. Composers may indicate temporal areas in which a number of sounds may be placed. Experimental music can span from a minimum of organization to a minimum of arbitrariness. As an example, Christopher Hobb's *voicepiece (1967)* is written for any number of vocalists and any length. Nyman argues that, usually, in Experimental music pieces, certain time frames may be chosen at random and filled with sounds.

Nyman argues that an important feature of Experimental music is the diversity of processes available; processes may be relationships between chance and choice. He argues that there are five types of processes: (1) *change determination processes*; for instance, when Cage used random numbers to choose tones, and also when he wrote pieces in which it was required to take information from the telephone directory during performance; (2) *people processes*, for instance, the eventuality of players getting lost or an unknown number of players; (3) *contextual processes*, such as actions taken on unpredictable conditions within the musicians or the audience; (4) *repetition processes*, such as unbounded loops; and (5) *electronic processes*, difficult to describe because they are not well formalized.

A characteristic of Experimental music is that, often, the starting and ending times of a piece are unknown. As an example, Nyman argues that in Wolff's *duo II for pianists (1958)*, the beginning and the ending times are determined in performance by the circumstances of the concert occasion. As another example, Nyman discussed Reich's *pendulum*

music (1968). In this piece, microphones are suspended from the ceiling. The piece begins when the performers swing the microphones and turn on the amplifiers; the piece ends after all microphones come to rest.

Nyman argues that performing Experimental music goes above and beyond performing of *Western music* because of all the possibilities that can be modeled with the five types of processes, and the unknown starting and ending times of a piece, as explained above.

Non-linear multimedia. Since 1950, computer technology is used to control sound structures; however, there is a long history of non-linear music in western culture. Vickery argues that, in the 20th century, there are examples of non-linear music such as Boulez's *third piano sonata (1958)*, and free improvisation with game strategies such as interactive electronics from Gordon Mum and several Stockhausen's pieces. Nonetheless, such an interest is not new. In fact, Vickery argues that Mozart composed minuets and trios in which the order of work's musical material was determined by coin-tosses, as we stated before.

Vickery has composed some non-linear pieces [Vickery 2003] in the 21st century. As an example, *ladders and snakes (2000)* is a piece in which the *ladder* processes descend to improvise in a later section, and the *snake* processes ascend to an earlier section, as a flash back in a film. As another example, *splice (2002)* is a piece in which the computer performs meta-music shaping of the sound made by the musician. Finally, in Vickery's piece *parallel trajectories (2003)*, performers have a score map with different paths from start to end, and they can also choose to stay silent in some parts. As an example, the score is presented in Figure 1.2.

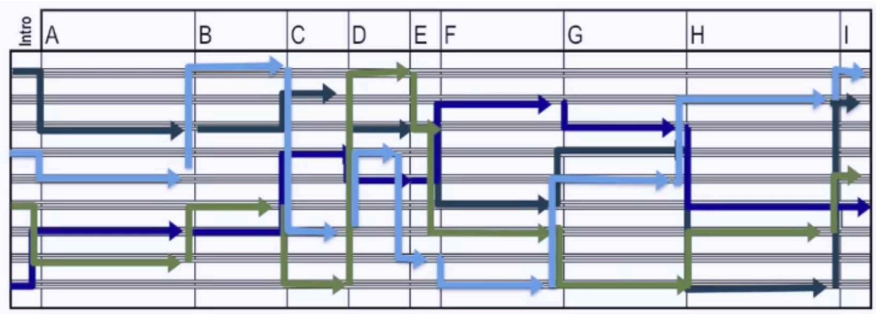

Figure 1.2: Score of Vickery's *parallel trajectories (2003)*. There are 14 lines of musical material and each of the 9 players is provided with four of the lines. There are 9 "modal points" in the score in which the player may choose a different line or choose to remain silent until the next point.

Furthermore, Vickery argues that computer coordination of live musical performance allows for the control and synchronization of the score; for instance, non-linear selection of multimedia material [Vickery 2011]. Music is traditionally linear: left-to-right and top-to-bottom. Computer music offers two main new possibilities according to Vickery: (1) Permutation of large structural blocks of music such as Stockhausen's *momente (1962)*, and (2) interactive generative processes may be used in real-time. There are some other

implications of such a computer-controlled behavior, according to Vickery [Vickery 2011]. As an example, Jason Freeman's *glimmer (2004)* is written for chamber orchestra and audience participation by waving four-inch LED sticks. Vickery's *delicious ironies (2002)* has also an unpredictable environment for the solo improviser with sample choice, playback speed, duration, volume and pan. As another examples, Vickery recalls Stockhausen's spectral analysis used in *zyklus (1959)* and *regrain (1959)*.

According to Vickery, non-linearity allows pieces to have *openness of interpretation* and *openness of content* [Vickery 2004]. Vickery cites some interesting examples. Game based analysis first used by Xenakis in *duel (1959)* and *strategies (1962)*, then used by John Zorn in *cobra (1984)*, allows the musician to give commands to games. Richard Teitelbaum, creator of *automata (1978)*, presents an analogy to finite state automata in which a system responds to user actions. The californian group *The* HUB is a computer network band in which the musicians and sounds communicate through a network.

Although the many examples that Vickery explained in his articles, he argued towards the urgent need of symbiotic human-machine interactive software to compose non-linear music [Vickery 2004]. In fact, we argue in this chapter why Vickery's preoccupation can be extended to non-linear multimedia in general, for instance, in multimedia art installations.

Electroacoustic music. All Electroacoustic music is made with electronic technology. Some electroacoustic compositions make use of sounds not available in typical acoustic instruments, such as those used in a traditional orchestra. Some Electroacoustic music can be created using non-acoustic technology that exists only in a recorded format (as a fixed medium), and is composed for reception via loudspeakers. The compositional material is not restricted to the inclusion of sonorities derived from musical instruments or voices, nor to elements traditionally thought of as "musical" (e.g., melody, harmony, rhythm and meter), but rather admits any sound, acoustic or synthetic. With the aid of various technologies, such as tape recorders and digital signal processing tools, this material can then be combined, juxtaposed, and transformed, in any conceivable manner [1].

A form of Electroacoustic music, specifically composed for loudspeaker presentation, is *Acousmatic music*. Unlike scored music, compositions that are purely acousmatic exist solely as audio recordings. The term *acousmatic* was introduced by Pierre Schaeffer and refers to the listening experience of *concrete music* in which the audience hears the music from the loudspeakers, without seeing the source of the sound[2]. In an acousmatic concert, the sound component is produced using pre-recorder media, or generated in real-time using a computer. The work is often *diffused* by the composer (if present), but the role of the interpreter can also be assumed by another musician. The main role of musician is to control *spatialisation*. As an example, consider one of Schaeffer's earliest work *five studies of noises (1948)* made without a computer.

The term *concrete music* is defined by Schaeffer as an opposition with way musical work usually goes. Instead of notating musical ideas on a paper with the symbols of solfège and entrusting their realization to well-known instruments, the question is to collect con-

[1] http://en.wikipedia.org/wiki/Electroacoustic_music

[2] en.wikipedia.org/wiki/Acousmatic_music.

crete sounds, wherever they came from, and to abstract the music values they were potentially containing. According to Pierre Henry, another well-known composer of this style, concrete music was not a study of timbre, it is focused on envelopes and forms[3].

A subtype of concrete music, in which sound was registered in magnetic tapes, is called *Tape music*[4]. In such a style, the starting and ending times of all the sounds remain fixed once the composition is over; as opposed, to some pieces of *acousmatic music* in which there is real-time sound generated by computer which order may change.

There is another style subsumed by Electroacoustic music: *"Mixed" music*, which merges acoustic sounds from traditional instruments played by musicians with electroacoustic sounds (diffused by loudspeakers). As an example, in Manoury's *partita I (2006)* for solo viola and live electronic effects, in Section VIIC, the composer wrote a note indicating that the all parts have to be played but in any order. The order is chosen by the musician. This is an example of non-linearity in Electroacoustic music. Another well-known example of "mixed" music is Manoury's *pluton (1988)* for piano and live electronics, and Stockhausen's *mikrophonie I (1964)* for tam-tam, microphone and filters.

1.1.1 Interactive multimedia

Interactive multimedia deals with the design of scenarios where multimedia content and interactive events can be handled by computer programs. Designers usually create multimedia content for their scenarios, and then bind them to external interactive events controlled by *Max/*MSP or *Pure Data (Pd)* programs [Puckette 1998, Puckette 1996]. We recall that examples of interactive multimedia are interactive museum exhibitions and multimedia installations.

Multimedia art installations are an artistic genre of three-dimensional works that are often site-specific and designed to transform the perception of a space. Installations evolved through the use of new and ever-changing technologies: from simple video installations, they expanded to include complex interactive, multimedia and virtual reality environments. Interactive installations were most frequently created and exhibited after 1990s, when artists were particularly interested in using the participation of the audiences to co-author the meaning of the installation[5]. As an example, there is an interactive installation based on spatial sensing written in Max [Yamauchi 2007]. Another example is an interactive installation based on probabilistic control [Baltera 2007]. Both installations are non-linear in the sense that the order in which they diffuse video and sound is unforeseen and depends on user interactions.

In addition to Max, interactive multimedia scenarios are also designed with commercial sequencers. Commercial sequencers for interactive multimedia are based on a fixed timeline with a very precise script such as *Pro Tools*[6], or a more flexible script using cue lists, for instance, the theater cue manager *Qlab*[7]. Another software to design such scenarios is

[3] http://en.wikipedia.org/wiki/Musique_concr%C3%A8te.
[4] http://en.wikipedia.org/wiki/Electroacoustic_music#Tape_music.
[5] http://en.wikipedia.org/wiki/Interactive_Art
[6] http://www.avid.com/US/resources/digi-orientation
[7] http://figure53.com/qlab/

Ableton Live[8]. Live is often used in Electroacoustic music and performing arts.

Example 1.1.1. Figure 1.3 shows the user interfaces of cue lists and timeline based sequencers, respectively.

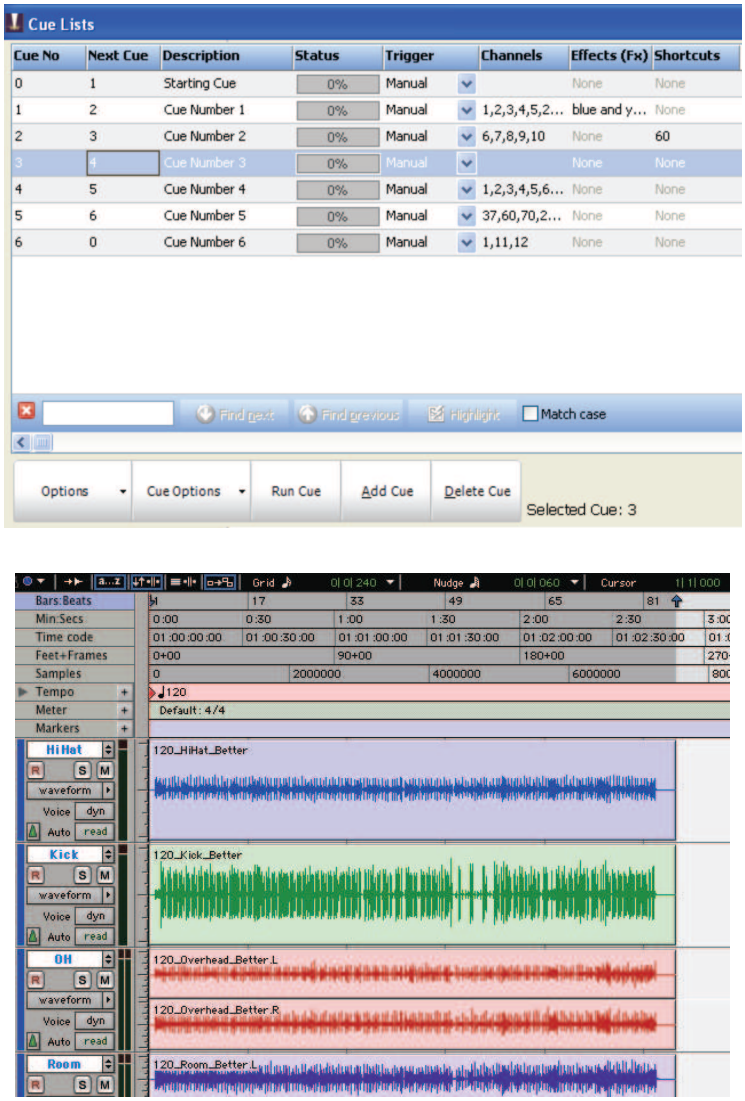

Figure 1.3: Cue-list based *Qlab* (above) exhibits a list on events and associated actions; it also defines whether and event is triggered by the computer or by the user. Timeline sequencer *Protools* (below) exhibits a timeline with several sound objects; starting and ending times are fixed and cannot be changed during performance.

Another well-known fixed timeline sequencer is the *Acousmograph* which is a software to represent graphically sounds in a composition. In fact, the acousmograph has been used

[8] http://www.ableton.com/

by Pierre Couprie for musicological analysis [Couprie 1999]. It is also worth to note that the acousmograph has been used to represent Gyorgy Ligeti's *artikulation (1958)*, as shown in Figure 1.4[9].

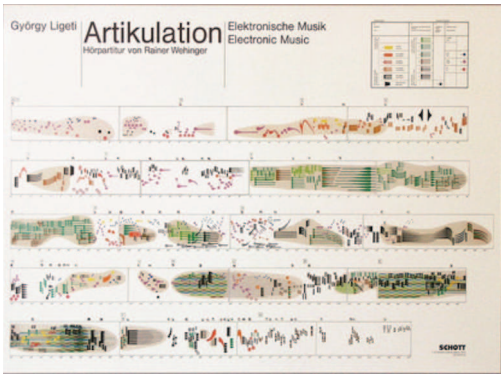

Figure 1.4: Visual listening score of Gyorgy Ligeti's *artikulation (1958)* created by Rainer Wehinger using *acousmograph*.

In what follows, we define the fixed timeline and the cue-lists time models, and the problems that have arisen because of the duality between these two time models, among other problems.

1.2 Problems with Interactive Multimedia Scenarios: Philosophical Discussion

The main problem with interactive multimedia scenarios is that there are two different time models, but existing tools only use one, and tools that allow both, offer both time models temporally unrelated. To understand this problem, we must travel 2500 years back in time. Desainte-Catherine *et al.* argued that this problem was already discussed by Parmenides of Elea and Heraclitus of Ephesus long before the invention of computers [Desainte-Catherine 2012] .

Problems with the time models. According to Desainte-Catherine *et al.*, what we call today Tape music, that began by editing and mixing sounds in magnetic tapes, is composed in a writing-oriented manner that corresponds to the *arrow metaphor* discussed by Parmenides. Parmenides argued that there are eternal properties and ordered events; for instance, "Socrates was born before he died". According to Parmenides, timeline goes from past to future. In this paradigm, it is difficult to define changes in the objects in the timeline. In fact, the only changes allowed at performance time of Tape music are in pan, volume, spacialization, among others parameters, but not on the starting and ending time of the sounds nor individual parameters for each sound.

[9] A video can be found at `http://wn.com/artikulation_ligeti`.

In contrast, many pieces of Experimental and Electroacoustic music, are based on real-time sound synthesis. They are usually written in *asynchronous dataflow languages* such as Max. According to Desainte-Catherine *et al.*, interactive multimedia is performance-oriented, and, for that reason, multimedia objects and time representation are quite poor. Performance-oriented software corresponds to the *river metaphor* described by Heraclitus: "we never bath twice in the same river". In this paradigm, the inference of the events flows is from the future, backwards because events are being "scheduled".

Identity is hard to define in Heraclitus' paradigm; for that reason, according to Desainte-Catherine *et al.*, we cannot define a permanent environment in asynchronous dataflow languages such as Max/MSP [Puckette 1998]. Time-stamped data is handled as a queue and there is only available a limited timeline to schedule the triggering of static events in most asynchronous dataflow languages. Nonetheless, it is worth noticing the effort made my Puckette to include a timeline in Pure Data, as shown in Figure 1.5 [Puckette 2002].

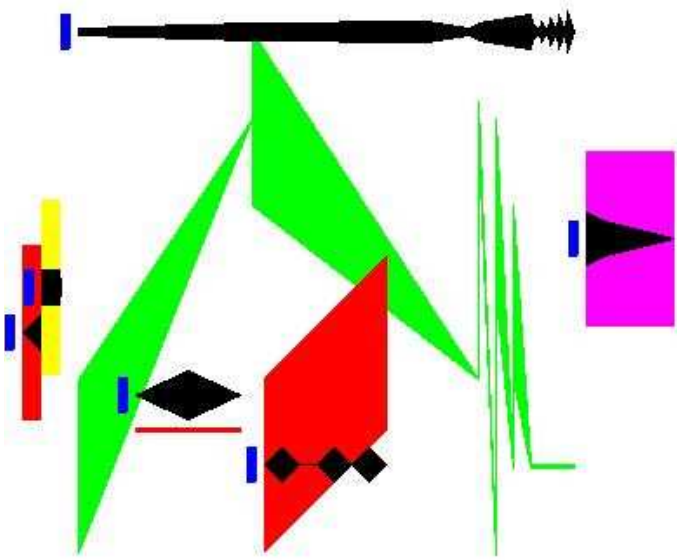

Figure 1.5: Writing a score in Pure Data. Horizontal axis represents time and the vertical axis frequency. Objects represent Pure Data's data structures. Shapes and colors are chosen by the composer to represent the data structures of the piece.

The problem of identify is important for both Electroacoustic and Experimental music. One implication is the ownership of Electroacoustic music, as explained by Dahan *et al.* [Dahan 2008]. According to Nyman, Cardew argued that when we hear on a tape or disk is indeed the same playing, but divorced from its natural context. As an example, Nyman argued that David Tudor (pianist) played Cage's *4'33" (1952)* and people think that Cage's *4'33" (1952)* is a piece for piano, but it is a piece that can be played by the means of any instrument [Nyman 1999].

Problems with time scales. In addition to the problem of identity, Schwer discussed another philosophical problem related to linguistics [Schwer 2005], which we believe that it is also related to music: Aristotle argued that between two time instants there is always a time instant. Therefore, the metaphoric timeline seams like the set of real numbers. Nonetheless, according to Schwer, there is a discrete understanding of time in Physics; for instance, in *quantic mechanics*, *Planck's time* is the smallest measure of time 10^{-41} seconds; in the atomic clock is 10^{-21} seconds; however, humans only discriminate at 10^{-1} seconds.

In Computer Science, as in Physics, time is also discrete because it is defined by the occurrence of events. For events to occur they have to be observed and this is discrete in nature. In favor of discrete time, the Stoics argued that the set of atomic instants is a discrete structure, thus we can pass from one instant to the next instant, according to Schwer.

The duality between discrete and continuous time is also a problem in multimedia interaction when we think about all the time scales available; for instance, user gestures, control events, sound processing and video processing. All those processes work at different time scales, and they are usually unrelated one from another in existing tools. Multimedia signals are continuous when they are analogic. Once they are sampled into the computer, they become discrete; however, they can be though of continuous in the sense that a listener will perceive them as continuous. In contrast, control signals, used to synchronize different media, are discrete time, and they are also perceived as discrete by the listeners.

Problem with synchronization. There is another problem derived from the time scales, as we discussed in [Toro 2012a]. The description of a multimedia scenario requires a consistent relationship between the representation of the scenario in the composition environment and the execution. Artistic creation requires a composition of events at different time scales. As an example, it is easy to describe that a video begins when the second string of a guitar arpeggio starts, but how can we achieve it in practice if the beginning of the notes of the arpeggio is controlled by the user?

The problem emerges at runtime. The example given above is very simple, but under high CPU load, a system interruption at the point of playing the arpeggio and the video can often lead to desynchronization, which is the case with Pure Data and Max. Usually, these eventualities are not considered by developers, as the quality of systems is evaluated according to an average performance. Nonetheless, during performance, it is desired that the system works well even under high CPU load, which is common when these systems process sound, video and image simultaneously.

The synchronization between the arpeggio and the video must be achieved in every execution. If it does not work for a performance, concert or show, the system performance is not satisfactory. Usually, artists prefer that an event is canceled if the event is not going to be properly synchronized with all the other media. Most users want a system that ensures that the events are either launched as they were defined in the score or they are not produced. Another alternative is based on the synchronization strategies for score following systems proposed by Echeveste *et al.* [Echeveste 2011]. Echeveste's strategies are designed to define behaviors for the cases in which events are not always properly synchro-

nized with other media due to musician's mistakes during performance or due to incorrect tempo calculations by the score following system.

Interactive multimedia belongs to the realm of *soft real-time*. We argue that in soft real-time, the usefulness of a result degrades after its deadline, thereby degrading the system's quality of service; whereas in *hard real-time* missing a deadline is a total system failure (e.g., flight control systems). It is difficult to ensure determinism in the execution of multimedia processes (e.g., sound, video and 3D images) in the soft real-time realm. Some *hard real-time* operating system like RT *Linux*[10] or *RedHawk*[11] include priority queues for processes to respect hard real-time constraints; however, in common operating systems, the user does not have this type of control. Note that software like Max and Live do not work on Linux.

Problems with conditional branching. Another issue arises when we think of non-linear music. When we think about choices based on conditions, we must consider *causality*. Causal relation is studied by *metaphysics*. According to Keil, substances are not causes; for instance, "if knife then always wound" is incorrect: An event and a verb are missing [Keil 2006]. In interactive multimedia, "If note 1 then always note 2" is also incorrect. A causal relation could be "when note 1 starts, then note 2 starts","whenever note 1 ends the note 2 ends", or "when the note 1 gets to a volume peak, then note 2 starts"; however, most tools do not provide this kind of causal relations.

Keil explains that physical systems are described in non-perturbed situations, but such rules may not always apply in real-life situations. As an example, a fire match will not light without oxygen, although a cause of lighting a match is to rub it against a striker. For that reason, when we model non-linear multimedia, we must consider user interactions. We must also consider that these interaction may arrive at any time.

Keil also points out that an event always has different causes susceptible of exceptions because the causes include less than the total state of the universe. For that reason, the causal relation is not transitive; therefore, the flapping of a butterfly's wing is not the cause of a storm on the other side of the world, according to Keil. As a consequence, we argue that users' choices should be made over single temporal objects (e.g., sounds or videos), instead of sequences of temporal objects. To choose a sequence of temporal objects, the sequence should be contained in one temporal object. In conclusion, each object must know who was its direct cause. As an example, consider Figures 7.2 and 7.3. In both figures, there is a mutually exclusive choice between two objects. If a composer wants to write a choice between two sequences of two objects, each two-object sequence must be contained inside a bigger object, as in Figure 7.2. We will discuss this issue in detail in Chapter 7.

Up to now we have considered causality dissociated from time, as treated by Keil; however, Russel gives a definition of causality that includes a time interval: "Given an event e, there is an event e_2 and a time interval τ, such that, every time that e_1 occurs, it is followed by e_2, after such an interval has passed" [Russel 2006]. We believe that

[10] http://www.windriver.com/index.html

[11] http://real-time.ccur.com/concurrent_redhawk_linux.aspx

Russels' definition is appropriate for multimedia interaction; however, with this definition, it is hard to understand scenarios with loops, for instance, when an "instance" of e_1 causes an "instance" of e_2, but then such an "instance" of e_2 causes another "instance" of e_1 in the future. What does this relation means? Are we traveling back to the time when e_1 was first executed? Are we creating a new "instance" of e_1 and executing it in the future? Are those two "instances" two different events with the same type (or action)?

The problem of "time travel" becomes even more difficult when we consider multiple instances of a temporal object that could be executed simultaneously. We must distinguish between the motive being repeated and the loop itself; we illustrate some cases in Figure 1.6. The problem gets even harder when we want to synchronize the ending times of motives and loops. In interactive multimedia, synchronization of loops and motives has been extensively studied by Berthaut *et al.* [Berthaut 2010].

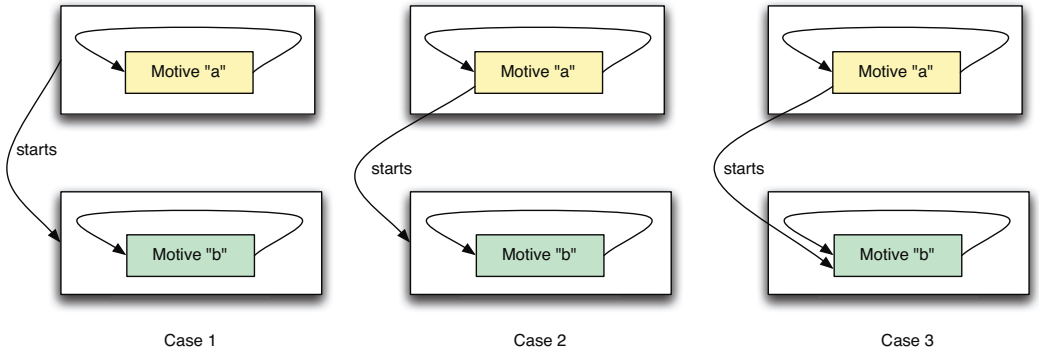

Figure 1.6: Possible scenarios synchronizing motives and loops. In case 1, the loop on the top starts the loop in the bottom; this means that the first repetition of motive "a" starts the first repetition of motive "b". In case 2, every repetition of motive "a" starts a new instance of the loop on the bottom. In case 3, each repetition of motive "a" starts at the same time than each repetition of motive "b".

There are some insights in metaphysics on how to solve the problem of having multiple instances of events. Laudisa argues that in *neoempirism*, leaded by Hume, everything that starts to exist must have a cause for its existence, but all human laws admit exceptions [Laudisa 2006]. To formalize such a principle, Laudisa proposes to distinguish between *singular events* and *event classes*: Let x and y be singular events, the existence of a causal connection means that (1) there are event classes of type X and of type Y, and (2) x is of type X and y is of type Y.

According to Laudisa's postulates, we could think about the start event of a temporal object as a class, and each time the temporal object starts, a different singular event that belongs to such a class is launched. Nonetheless, there is still a problem: how to model choices through time, should we consider a *branching time* or a *linear time*? Let us analyze what computer scientists have to say on this dichotomy.

According to Pratt, there is an analogy: branching time represents local time, and linear time represents global time, in the same way as true concurrency represents local information and false (or interleaving) concurrency represents global information [Pratt 1992]. In linear time, all choices are made at the beginning, it means that we cannot distinguish between a systems that performs actions $a.b + a.c$ from a system that performs $a.(b + c)$, where "." represents sequential composition and "+" represents blind choice. The first system chooses either to execute event a and then event b or event a and then event c, whereas the second system executes a and then chooses to execute either b or c.

As an example, Vardi argues that with *computational time logic* (CTL), it is possible to characterize bisimulation of concurrent systems. In terms of complexity of the model-checking problem, using CLT is exponentially easier than linear-time logic LTL, but in a competitive analysis, with formulae that can be expressed in both logics, model checkers behave similarly. There is an advantage of linear time: LTL is more intuitive to describe properties because its specifications describe computations, whereas CTL's specifications describe trees of computations [Vardi 2001].

Although branching time seams more appropriate to represent conditional branching in interactive multimedia, we believe that linear time is enough because we can consider that all the temporal objects in a scenario are always executed, but some execute silent actions and some execute visible actions, allowing us to express choices. We want to keep the specification of properties simple.

1.3 Problem Statements of this Dissertation

After analyzing the philosophical problems, the Electroacoustic and Experimental music pieces described above, and existing tools and formalisms for multimedia scenarios, discussed in Chapter 3, we have identified seven problems with existing software to design multimedia scenario: (1) there is no formal model for multimedia interaction, (2) multimedia scenarios have limited reusability and difficulties with their persistence, (3) time models are temporally unrelated, (4) multimedia interaction software products provide no hierarchy, (5) the different time scales are unrelated, (6) schedulers for multimedia scenario tools are not appropriate for soft real-time, and (7) there is no model to combine temporal relations and conditional branching. In what follows we explain each of those problems.

There is no formal model for multimedia interaction. As we explained before, designers usually create multimedia content for their scenarios, and then bind them to external interactive events controlled by *Max*/MSP programs. We advocate a model that encompasses facilities (1) to design multimedia scenarios having complex temporal relationships among components and (2) to define effective mechanisms for synthesis control based on human gestures. We claim that no such model has been proposed.

Such a general model must have formal semantics, as required for automated verification of properties of the scenario that are fundamental to its designers and users. As an example, to verify that temporal objects will be played as expected during performance. In general, we need to prove some property of each execution trace; for instance, that the

music motive with notes C-D-E appears in all the traces of execution (or at least in one). Another example is to state that there is at most one temporal object being executed simultaneously. This property is useful in some theater performances to state that there is at most one curtain being moved at the time because of power constraints. Such properties cannot be verified in applications based on informal specifications, as it is the case for most existing scenarios with interactive controls.

Limited reusability and difficult preservation. Limited reusability is also a problem caused by the lack of formal semantics: A module made for one scenario might not work for another one because the program may have dependencies on external parameters that are not stated explicitly. The lack of semantics also makes it difficult to preserve multimedia scenarios because there is usually not a score nor a technology-independent precise way for describing the objects, and the temporal and dataflow relations among them.

Time models are unrelated. Software to design multimedia scenarios is usually based either on a fixed timeline with a very precise script or a more flexible script using cue lists, as we stated before. A commonly used software to design such scenarios is Live because it allows to use both the fixed timeline and the cue lists, but the two time models are unrelated temporally. In fact, most software products, for instance sequencers, provide only one time model or they are unrelated temporally, as we argued previously.

No hierarchy for temporal objects. Most software do not provide a hierarchy to represent the temporal objects of the scenario. As an example, using a hierarchy, it is possible to control the start or end of an object by controlling those from its parent. In interactive music, Vickery argues that using a hierarchy is useful to control higher-order parameters of the piece; for instance, to control the volume dynamics instead of the volume of each note [Vickery 2004]. Concentrating on foreground parameters can lead to music that is too superficial as multiple serialism, according to Vickery.

Time scales are unrelated temporally. The different time scales are often unrelated and cannot be controlled in the same tool. *Discrete user gestures* (e.g., clicking the mouse), *control events* (e.g., control messages) and *sound processing* have different sampling frequencies and different computing models. As a consequence of having the time scales unrelated, it is difficult to associate, for instance, a human gesture to both control events and sound processing parameters.

Event schedules are not appropriate for real-time. Schedulers for asynchronous dataflow languages (e.g., those from Pd and Max) control both signals and control messages together and they do not support parallelism, thus they often fail to deliver control messages at the required time; for instance, when they work under a high CPU load, which is common when they process video and 3D graphics in addition to sound.

To solve the problem of scheduling and to write high-performance *digital signal processors (*DSPs*)* for Max and Pd, users often write C++ plugins to model DSPs with loops or

independent threads. C++ plugins solve part of the problem, but the control messages –for the input and output of these plugins– are still being scheduled by Max or Pd's schedulers.

Another solution for the scheduler problem –often used during live performance– is to open one or two instances of Max/MSP or Pd simultaneously, running different programs on each one. Nonetheless, synchronization is usually done either manually during performance or by using *open sound control (*OSC*)*, which adds more complexity and latency.

No model for conditional branching and temporal relations. Up to our knowledge, there is not a model for interactive multimedia to represent scores in which is possible to combine complex temporal relations and conditional branching based on conditions over the variables defined in the scenario. In fact, Allombert proposes in [Allombert 2009] an extension with conditional branching to interactive scores, but in such a model he only considers conditional branching and no temporal relations.

1.4 Background Overview

Before we can present a solution to the problems described above, in what follows, we present an overview of the three subjects discussed in Chapter 4.

Event structures. Langerak's *timed event structures* (henceforth *event structures*) is a mathematical model to represent systems with non-determinism, real-time and concurrency [Baier 1998]. Event structures allow to define a partial order among concurrent events. Event structures include a set of *labeled events* and a *bundle delay relation*. The bundle delay relation establishes which events must happen before some other occurs. Actions can be associated to events. Events are unique, but two events may perform the same action. Events can be defined to be "urgent". An *urgent event* occurs as soon as it is enabled. In addition to the bundle relation, event structures include a *conflict relation* establishing events that cannot occur together. Events can also be given *absolute occurrence times*.

Non-deterministic Timed Concurrent Constraint (ntcc). Ntcc is a process calculus that models concurrency, non-determinism and asynchrony [Nielsen 2002]. In ntcc, a system is modeled in terms of processes adding to a common *store* partial information on the value of variables. Concurrent processes synchronize by blocking until a specified piece of information can be deduced from the store contents.

Ntcc includes the notion of discrete time as a sequence of time units. Each time unit starts with a (possibly empty) store supplied by the environment. Processes scheduled for that time unit are then run until quiescence. The resulting store is the output at that time unit. Residual processes might also result; these are scheduled for the next (or any future) time unit and computation starts all over again.

Faust. Faust is a functional programming language for signal processing. In Faust, DSP algorithms are functions operating on signals [Orlarey 2004]. Faust is based on *signal processors*, which are functions that operate over tuples of signals; for instance, to merge two

signals, recursively compose them or multiply them. Graphical user interface (GUI) objects in Faust can be defined in the same way as other signals. Therefore, we can control buttons, check boxes and integer inputs –originally designed for users– from another program such as *Ntccrt*[12] [Toro 2009], a ntcc interpreter.

1.5 Solution: The Interactive Scores Formalism

There are formalisms to model interactive scenarios such as *interactive scores*. Interactive scores has been a subject of study since the beginning of the century [Desainte-Catherine 2003]. The first tool for interactive scores is *Boxes*, developed by Beurivé [Beurivé 2001]. Boxes was conceived for the composition of Electroacoustic music with temporal relations; however, user interaction was not provided. A recent model of interactive scores [Allombert 2009], that significantly improves user interaction, has inspired two applications: *i-score* [Allombert 2008b] to compose and perform Electroacoustic music and *Virage* [Allombert 2010] to control live performances and interactive exhibitions. In Chapter 3, we give a further discussion on the history of interactive scores.

Scenarios in interactive scores are represented by *temporal objects*, *temporal relations* and *interactive objects*. Examples of temporal objects are sounds, videos and light controls. Temporal objects can be triggered by interactive objects (usually launched by the user) and several temporal objects can be executed simultaneously. A temporal object may contain other temporal objects: this hierarchy allows us to control the start or end of a temporal object by controlling the start or end of its parent. Hierarchy is ever-present in all kinds of music: Music pieces are often hierarchized by movements, parts, motives, measures, among other segmentations.

Temporal relations provide a partial order for the execution of the temporal objects; for instance, temporal relations can be used to express precedence between two objects. As an example of *relative temporal relations*, the designer can specify that a video is played strictly before a light show or between 10 and 15 seconds before. As an example of *absolute temporal relations*, the designer can specify that a loop starts three seconds after the video.

New semantics for interactive scores. We provide an abstract semantics for interactive scores based on *timed event structures*. The purpose of such a semantics is (1) to provide an easy, declarative way, to understand the behavior of a score, and (2) a simple theoretical background to specify properties of the system. In constraint programming, we can specify some properties of the scores such as playability. We can also specify those properties in event structures; moreover, the notion of *trace*, inherent in event structures, is more appropriate than temporal constraints for certain properties. As an example, to specify that a music motive appears in at least one trace of execution.

This study led us to discover that there is no difference between interactive objects and the other temporal objects in the event structures semantics: such a difference can only be observed in the operational semantics. That was the main reason to introduce an operational semantics based on ntcc, on the lines of Allombert *et al.* [Allombert 2006].

[12] ntccrt.sourceforge.net/

Nonetheless, in Allombert *et al.*'s models of interactive scores, it was not precisely stated how to execute scores whose temporal object durations are arbitrary integers intervals; for instance, a score in which object a must be executed between two and four time units after object b. Allombert *et al.*'s models handle *flexible-time intervals*: $\{0\}$ to express simultaneity, and $(0,\infty)$ and $[0,\infty)$ for precedence or for the flexible duration of the objects. Allombert *et al.*'s models also miss an abstract semantics.

We extend the interactive scores formalism with an abstract semantics based on event structures and an operational semantics specified in ntcc, providing (1) a new insight into the interactive scores model; (2) more complex temporal relations to bind objects, including arbitrary sets of integers in the event structures semantics and arbitrary intervals in the operational semantics; and (3) the possibility to verify properties over the execution traces. In order to use arbitrary integer intervals in our operational semantics, we show that several transformations to the event structures semantics are needed to define operational semantics that can dispatch the temporal objects of the score in real-time.

To complete our framework, we also present in this dissertation two extensions of the interactive scores formalism: one for conditional branching and one for signal processing. We also explain the implementation of interactive scores and the implementation of an automatic verification tool for ntcc.

Time conditional branching interactive scores. Non-linear music pieces are *open works*. According to Vickery, open works may have openness of interpretation or openness of semantic content [Vickery 2003]. Conditional branching is essential to describe pieces with openness of interpretation.

Conditional branching is commonly used in programming to describe control structures such as *if/else* and *switch/case*. It provides a mechanism to choose the state of a program based on a condition and its current state. In multimedia interaction, using conditional branching, a designer can create scenarios with loops and choices (as in programming).

In the domain of interactive scores, using conditional branching, the user or the system can take decisions on the performance of the scenario with the degree of freedom that the designer described. The designer can express under which conditions a loop ends; for instance, when the user changes the value of a certain variable, the loop stops; or the system non-deterministically chooses to stop. As an example, the designer can specify a condition to end a loop: When the user changes the value of the variable *end* to `true`, the loop stops. The designer can also specify that such choice is made by the system: The system non-deterministically chooses to stop or continue the loop.

We chose event structures because it is a powerful formalism for concurrency that allow us to extend the interactive scores semantics with conditional branching and loops in a very precise and declarative way. Conditional-branching timed interactive scores were introduced in [Toro 2010c, Toro 2010b]. Such an extension has operational semantics based on ntcc, but it misses an abstract semantics to understand the conflicts among the temporal objects that take place when modeling conditions and choices.

Interactive scores with signal processing. It is crucial that interactive multimedia software products preserve the *macroform* and the *microform* of the scenario. The macroform includes the structure of the scenario (e.g., the tempo and the duration of the scenes, movements, parts and measures). The microform comprises the operations with samples (e.g., micro delays, articulation, intonation, and envelop of the sound).

We propose an extension to the interactive scores formalism for sound synthesis. In this extension, we deal with microstructure and macrostructure of sound, not the structure of image nor other media. In the interactive scenarios we consider, we can deal with streams produced in real-time (e.g., a stream captured from the microphone).

We define a new type of temporal relations meant for high precision; for instance, to express micro delays. We also introduce *dataflow relations*; for instance, how the audio recorded by a temporal object is transferred to another object to filter it, add a micro delay, and then, send it to another temporal object to be diffused.

We also propose an encoding of the scenario into two models that interact during performance: (1) A model based on the ntcc for concurrency, user interactions and temporal relations, and (2) a model based on Faust for sound processing and micro controls. An advantage of having a formal model for ntcc and Faust interoperation is that we could prove properties such as playability, and predict the behavior of the system.

The novelty of our approach is using the constraints sent from ntcc to control Faust. We tested our examples in Pd, although they could also be compiled for Max or as a standalone program since both Faust and ntcc can be translated into C++ and Max. In fact, the final goal of our research is to develop a standalone program for interactive scores. Such a program should be general enough to interact with Pure Data, Live, Max/MSP and other existing software either by passing messages or by generating plugins for those languages.

Execution of interactive scores. We give operational semantics for interactive scores, but we need to execute those models. The execution must be able to interact with a user in real-time. Since the operational semantics are given in ntcc, we need an interpreter for ntcc capable of real-time interaction and being able to control multimedia objects such as sound, video and lights.

There are some interpreters for ntcc, but they are not suitable for real-time interaction [Muñoz 2004, Rueda 2006]. We chose a real-time capable interpreter for ntcc, *Ntccrt* [Toro 2009], to execute our models. Ntccrt is based on *Gecode* [Tack 2009]: state-of-the-art in constraint propagation. Ntccrt programs can be compiled into standalone programs, or plugins for Pd or Max. Users can use Pd to communicate any object with the Ntccrt plugin. In fact, Ntccrt can control all the available objects for audio processing defined in Pd, although our goal is to use Faust for such tasks.

Ntcc belongs to a bigger family of process calculi called *concurrent constraint programming* (CCP). In the last decade, there has been a growing interest for CCP models of multimedia interaction [Rueda 2002, Rueda 2001, Rueda 2004, Rueda 2005b, Rueda 2006, Allombert 2006, Toro 2009, Olarte 2009b, Olarte 2011, Toro 2012b][13].

Ntcc is not only useful for multimedia semantic interaction, ntcc has also been used

[13] We will discuss all these works in this dissertation.

in other fields such as modeling molecular biology [Rueda 2005a], analyzing biological systems [Gutiérrez 2007] and security protocols [López 2006]. Therefore, advances on the simulation of ntcc models will be useful not only for multimedia interaction, but also for other fields.

Automatic verification. A disadvantage of ntcc is the lack of automatic verification tools available. This limits the applicability of the verification techniques to small problems. We claim for the urgent need of a verification tool for ntcc. First, because ntcc has been widely used to model reactive systems and verify properties about them, but the verification had to be done by hand. Second, because there are not many frameworks to model and verify multimedia interaction systems, and ntcc has been proved to be successful in that field.

We developed a bounded-time model checking procedure for ntcc, *ntcc*MC[14]. The model checker is based on encoding ntcc processes and *constraint linear-time logic* (CLTL) formulae into deterministic finite state automata. Examples of CLTL formulae are "always the constraint $pitch = 60$ can be deduced from the output store", namely $\Box(pitch = 60)$; and "eventually object a and object b are launched at the same time", namely $\Diamond(launch_a \wedge launch_b)$. We explain CLTL in detail in Chapter 4.

Ntcc has been used since its beginnings to prove properties of multimedia interaction systems. Ntcc is a powerful formalism because it allows to simulate the behavior of a model and also to verify properties of the model. As an example, ntcc was used to verify properties of a musicological problem of western-african music [Rueda 2002]. The reader may also look at [Rueda 2004] and [Rueda 2001] for other examples of verification of multimedia interaction systems.

[14] http://sourceforge.net/projects/ntccmc/

CHAPTER 2

Contributions

Most of the material of this dissertation has been previously reported. In what follows, we present the structure of this dissertation; afterwards, we present the material already published.

2.1 Organization

In what follows we describe the structure of this dissertation which is divided in three parts: *Introduction*, *Models of Interactive Scores* and *Implementation*. Each part contains four chapters, and each chapter concludes with a summary of its contents and a discussion. We present the map of this dissertation in Figure 2.1.

Chapter 3. We discuss related work to interactive scores in this chapter. We discuss several formalisms such as score following, synchronous languages, asynchronous languages, process calculi, Petri nets, and their relation to the formalism of interactive scores. Afterwards, we discuss previous models of interactive scores and existing implementations. We recommend the reader to look through this chapter to understand how is the contribution in this dissertation related to other formalisms and tools.

Chapter 4. We already presented intuitively event structures, non-deterministic timed concurrent constraint (ntcc) calculus and Faust in Chapter 1. In Chapter 4, we present in detail these three formalisms. A reader interested in operational semantics of interactive scores, described in Chapters 5 and 6, and the simulation of ntcc models described in Chapter 9, should read in detail the intuitive semantics and operational semantics of ntcc. A reader interested in ntcc model checking, presented in Chapter 11, and the proof of correctness of the hierarchical model of interactive scores, presented in Appendix A, should also read denotational semantics and ntcc's logic. Event structures are required to understand abstract semantics of interactive scores presented in Chapter 5 and 6. Finally, to understand the signal processing extension of interactive scores, presented in Chapter 8, a reader should read the section on Faust.

Chapter 5. A structural definition and event structures semantics of interactive scores without hierarchy are presented here. This is a simplified version of the results presented in [Toro 2012b]. We also present new results on the complexity of the playability of a scores , results that also apply for the general case of scores with hierarchy. In Chapter 5, we also introduce properties of the score that can be expressed using the event structures semantics, previously published in [Toro 2012b].

Chapter 6. In this chapter, we present an enhanced structural definition and event structures semantics of interactive scores with hierarchy. We also present an operational semantics on ntcc. Most of the results of this chapter are published in [Toro 2012b].

Chapter 7. In this chapter, we present an extension of interactive scores with conditional branching without loops. We define an structural definition and event structures semantics. Afterwards, we define an extension of such model with loops. We also present ideas on how to define operational semantics for both. The model of interactive scores with conditional branching that includes loops was published in [Toro 2010c] and [Toro 2010b]; the event structures semantics remains unpublished.

Chapter 8. In this chapter, we present an extension of interactive scores with audio processing. In previous chapters, we only dealt with the macroform of multimedia content; in this chapter we also include the microform of sound, allowing us to express microdelays between temporal objects and dataflow relations stablishing how the input and output of temporal objects is transferred from one to another. These results were published in [Toro 2012a].

Chapter 9. In this chapter we present some simulations of interactive scores models using a real-time interpreter for ntcc, Ntccrt. We explain how Ntccrt was designed and how it works. Afterwards, we explain how we ran the models, and, finally, we present quantitative results on the execution of the such models. Quantitative results were already published in [Toro 2010b, Toro 2012a].

Chapter 10. In this chapter, we introduce a *extended markup language* (XML) file format for interactive scores presented in Chapter 6 and for the conditional branching extension presented in Chapter 7. We also discuss related work on file formats for multimedia interaction. Results presented on this chapter have not been published yet.

Chapter 11. In this chapter, we present related work on model checkers and verification techniques for that could be used for interactive scores. Afterwards, we introduce the model checker we developed, ntccMC, to verify properties of ntcc models. The purpose of this model checker is to verify properties of the scores before the execution; for instance, that the score is playable. The contribution has not been published yet.

Chapter 12. In this chapter we present a summary of this dissertation. Afterwards, we give concluding remarks with respect to the previous models of interactive scores, the problems stated in Chapter 1, and the advantages and disadvantages of interactive scores with respect to other tools and formalisms. Finally, we present some future work.

2.2 Published Contributions

In what follows, we present the material already published.

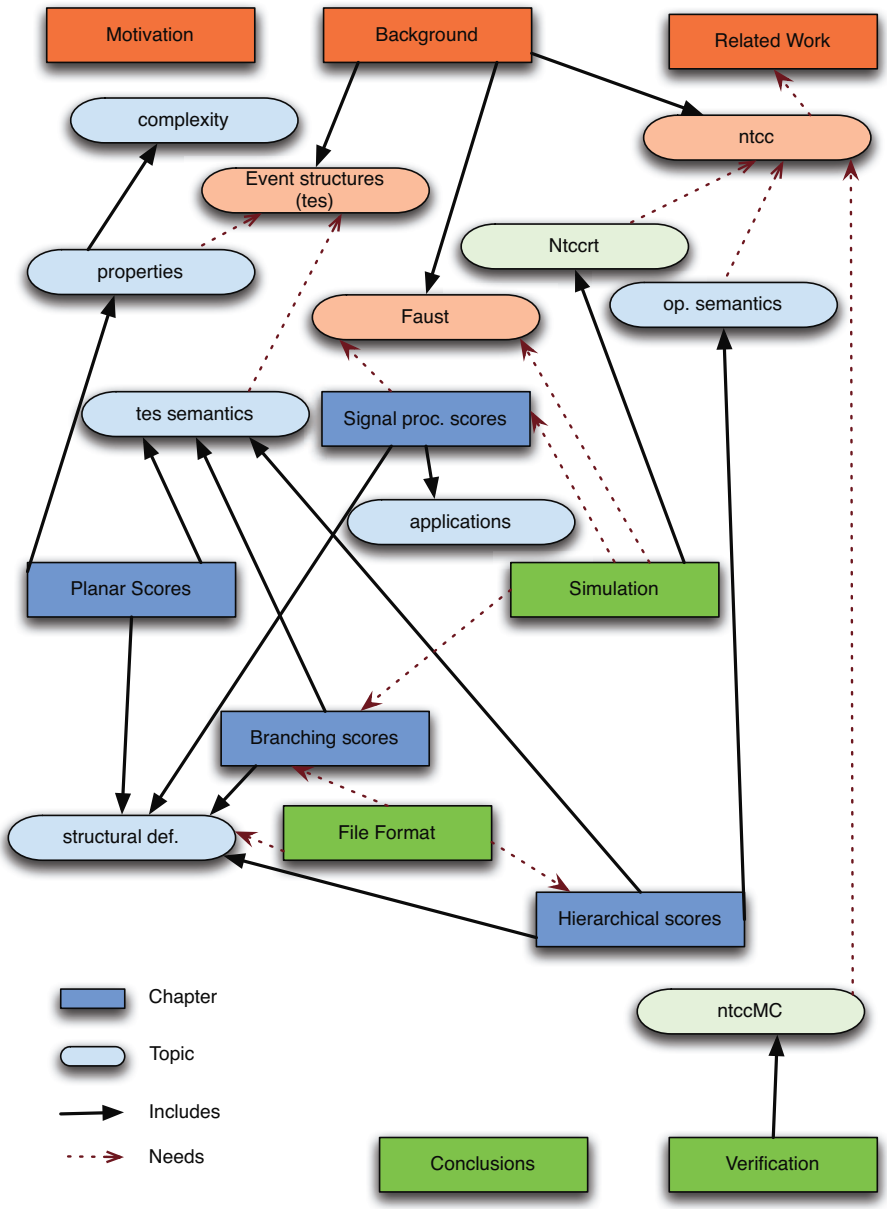

Figure 2.1: Map of this dissertation. This figure explains the relations between the main topics of this dissertation. Motivation and Related Work chapters are crucial to understand the rest of the dissertation. Conclusions chapter summarizes all other chapters and discusses their implications.

Book chapters

- Antoine Allombert, Myriam Desainte-Catherine and Mauricio Toro. *Modeling Temporal Constrains for a System of Interactive Score*. In Gérard Assayag and Charlotte Truchet, editors, Constraint Programming in Music, Chapter 1, pages 1–23. Wiley, Hoboken, NJ, USA., 2011. [Allombert 2011].

  The contributions of this paper are given in Chapter 7.

- Carlos Olarte, Camilo Rueda, Gerardo Sarria, Mauricio Toro and Frank Valencia. *Concurrent Constraints Models of Music Interaction*. In Gérard Assayag and Charlotte Truchet, editors, Constraint Programming in Music, Chapter 6, pages 133–153. Wiley, Hoboken, NJ, USA., 2011. [Olarte 2011].

  Some of the main contributions of this paper are discussed in Chapter 3.

Journal article

- Mauricio Toro, Myriam Desainte-Catherine and Camilo Rueda. *Formal semantics for interactive music scores: A framework to design, specify properties and execute interactive scenarios*. Journal of Mathematics and Music. To appear in late 2012. [Toro 2012b].

  The main contributions of this paper are discussed in Chapter 5 and 6.

Proceedings of international conferences

- Mauricio Toro, Myriam Desainte-Catherine and Julien Castet. *An Extension of Interactive Scores for Multimedia Scenarios with Temporal Relations for Micro and Macro Controls*. In Proc. of Sound and Music Computing (SMC) '12, Copenhaguen, Denkmark, July 2012. [Toro 2012a].

  The main contributions of this paper are discussed in Chapter 8.

- Mauricio Toro and Myriam Desainte-Catherine. *Concurrent Constraint Conditional Branching Interactive Scores*. In Proc. of SMC '10, Barcelona, Spain, 2010. [Toro 2010b].

  The main contributions of this paper are discussed in Chapter 7.

Proceedings of national conferences

- Mauricio Toro, Myriam Desainte-Catherine and P. Baltazar. *A Model for Interactive Scores with Temporal Constraints and Conditional Branching*. In Proc. of Journées d'informatique musical (JIM) '10, May 2010. [Toro 2010c].

  The main contributions of this paper are given in Chapter 7.

Extended abstracts and short papers

- Mauricio Toro. *Structured Musical Interactive Scores (short)*. In Proc. of International Conference on Logic Programming (ICLP) 2010, 2010. [Toro 2010a].

  This abstract describes some of the work on the extension of interactive scores with signal processing and automatic verification of ntcc, presented in Chapters 8 and 11.

CHAPTER 3

Related work

Contents

In this chapter, we present software and formalisms related to interactive scores, and we discuss previous models and implementations of interactive scores.

3.1 Related Software and Formalisms

In what follows we describe software and formalisms used in multimedia interaction such as sequencers, signal processing languages, dataflow languages and process calculi.

3.1.1 Sequencers

Software to design multimedia scenarios are usually based either on a fixed timeline with a very precise script or a more flexible script based on cue lists, as we stated in Chapter 1. As an example of fixed-timeline sequencers, there are two well-known sequencers for *Mac*

OS X: *Pro tools*[1] and *Final cut pro*[2]. As another example, the theater cue manager *Qlab*[3] is based on cue lists. In Qlab, the user programs a list of upcoming events; however, Pro tools, Final cut pro and Qlab only use one time model and cannot use both.

Another software to design multimedia scenarios is *Ableton live*[4]. Live is often used in Electroacoustic music and performing arts because it allows to use both the fixed timeline and the cue lists. Nonetheless, both time models are unrelated temporally.

An advantage of interactive scores over the previously mentioned sequencers is to relate temporally both time models and to model conditional branching.

3.1.2 Computer-assisted improvisation

Computer-assisted improvisation usually considers building representations of music, either by explicit coding of rules or applying machine learning methods. An interactive machine improvisation system capable of real-time must perform two activities concurrently: stylistic learning and stylistic simulation. As an example, the *Omax* system [Assayag 2006, Maniatakos 2010] and the *Continuator* [Pachet 2002] construct models to represent the sequences played by the musician and create their own sequences based on the musician's style.

Improvisation systems are interactive and concurrent, but they are different to interactive score systems: their goal is to create music based on the user style, whereas interactive scores is a formalism to compose music (or create multimedia scenarios). In interactive scores, the designer describes several rules that have to be respected during execution and the system does not produce new sequences nor sounds that are not written in the score.

3.1.3 Meta-instruments

A meta-instrument is a musician-machine interface and a gesture transducer intended for Electroacoustic music, multimedia work, and, more generally, for controlling a program in real-time. A class of meta-instruments allows to control the activation and release of notes. Interpretation of musical pieces based on activating and releasing notes has been studied by Haury [Haury 2008].

Haury identifies four ways for interpretation: *dynamic variations* as the possibility to continuously modify the volume of the notes during the performance, *accentuation* as temporary volume variations, *phrasing* as modifying the binding of the notes, and *agogic variations* as the possibility to change the date of beginning and end of the notes. Haury's research focuses on agogic modifications. As examples of agogic modifications, in Haury's meta-instrument, the *metapiano*, the musicians can start or stop a group of notes through control points placed in the piece that he calls *interaction points*. A pause is a good example of interaction point in instrumental music because the musician or the conductor can choose the duration of the pause. Haury's work inspired Allombert *et al.*'s models of interactive scores.

[1] http://www.avid.com/us/products/pro-tools-software
[2] http://www.apple.com/finalcutpro/
[3] http://figure53.com/qlab/
[4] http://www.ableton.com/

3.1.4 Score following

Another kind of systems capable of real-time interaction are *score following* systems [Cont 2008]. To use such systems, we must first write a score for the musician and for the computer. During execution, such systems track the performance of a real instrument and they may play multimedia associated to certain notes of the piece. Nevertheless, to use these systems it is necessary to play a music instrument; whereas to use interactive scores, the user only has to control some parameters of the piece, such as the starting and ending times of the temporal objects. Score following systems can also provide temporal relations and hierarchical relations [Echeveste 2011]; however, the system tracks the performance of a music instrument and is not meant to work with a meta-instrument. In contrast, one of the main advantages of interactive scores is meant to work with meta-instruments.

3.1.5 Asynchronous dataflow languages

Stream processing can be modeled as a collection of separate but communicating processes. Dataflow is the canonical example of stream processing. There is *synchronous dataflow* and *asynchronous dataflow* [Stephens 1997]. Synchronous dataflow they lack of FIFO queues to communicate channels like asynchronous dataflow languages. This is a main difference between the synchronous and asynchronous dataflow languages.

As an example, asynchronous dataflow languages *Max/*MSP and *Pure Data (Pd)* [Puckette 1998] are often used to control signal processing and control events by human gestures. Max and Pd distinguishes between two levels of time: the *event scheduler* level and the *digital signal processor (*DSP*)* level. Max and Pd programs, called *patches*, are made by arranging and connecting building-blocks of objects within a visual canvas. Objects pass messages from their outlets to the inlets of connected objects. The order of execution for messages traversing through the graph of objects is defined by the visual organization of the objects in the patch itself[5].

There are several problems with Max and Pd that we aim to overcome, as we explained in Chapter 1. First, their schedulers control both audio signals and control messages together and they do not support parallelism, thus they often fail to deliver control messages at the required time; for instance, when they work under a high CPU load, which is common when they process video, 3D images and sound. In Chapter 8, we present some insights on how to solve this problem; nonetheless, this is still an open problem.

To solve the scheduling problem and to write high-performance DSPs for Max and Pd, users often write C++ plugins to model loops and independent threads. C++ plugins solve part of the problem, but the control messages –for the input and output of these plugins– are still being scheduled by Max or Pd's schedulers.

Second, there is another problem with Max and Pd: they do not provide an environment to design scenarios. The different time scales are often unrelated and cannot be controlled in the same tool: *Discrete user gestures* (e.g., clicking the mouse), *control events* (e.g., control messages) and *signal processing* have different sampling frequencies and computing models.

[5] http://en.wikipedia.org/wiki/Max_(software)

As we explained in Chapter 1, one goal of the extension of interactive scores with signal processing is to overcome the existing problems of the asynchronous dataflow languages mentioned.

3.1.6 Synchronous dataflow languages

There are three well-known french *synchronous languages*: *Esterel*, *Lustre* [Halbwachs 1991, Halbwachs 1994] and *Signal* [Gautier 1987]. Benveniste *et al.* discussed the advantages and limitations of such languages 12 years after they were conceived [Benveniste 2003]. They argue that synchronous languages were designed to implement real-time embedded applications, thus such languages work on the deterministic concurrency paradigm and they are meant to model deterministic system behavior. Synchrony divides time into discrete intervals and supposes that operations take no time (e.g., to assign a variable or read a value).

Benveniste *et al.* argue that Esterel is imperative and it is well-suited for describing control. Signal is based on the reactive programming paradigm: A program does something at each reaction and it may be embedded in some environment. Signal is a multiclock language. Lustre supports recursive definitions, but may not contain cyclic definitions, and a variable can only depend on past values. Both Lustre and Signal have clocks to align streams, but they lack of FIFO queues to communicate channels like asynchronous dataflow languages. This is a main difference between the synchronous and asynchronous dataflow languages.

A very useful feature of synchronous dataflow languages is multirate computation. Using multirate computation, it is possible to easily handle control signals, video signals and audio signals that have different sampling rates. In fact, Forget compared the mutirate capabilities of Esterel, Lustre and Signal [Forget 2009]. Forget argues that in Lustre each variable is a flow. Lustre has a clock, but multirate is hard to describe. In Signal, variables are signals instead of flows. Clocks in Signal are first class objects; therefore, it can be polychronous, but multirate is also hard to achieve. Finally, Esterel focuses on control flow, where several modules communicate through signals, Esterel also has some asynchronous extensions and automated verification, but does not support multirate.

Faust is a synchronous language with formal semantics for multirate; however, this functionality has not yet been implemented [Jouvelot 2011]. Faust is a functional programming language for signal processing. In Faust, DSP algorithms are functions operating on signals, as we explained in Chapter 1. Faust programs are compiled into efficient C++ code that can be used in multiple programming languages and environments; for instance, in Pure data [Gräf 2007]. Faust is the DSP language we chose for our extension of interactive scores with signal processing. In Chapter 4, we give more details on Faust.

There is another well-known synchronous dataflow language. *Csound*[6] has three types of variables with different time levels (and different sampling rates): instrument variables, control variables and audio variables. In fact, control variables correspond to event scheduler sampling rate and audio processes to DSP level in Max. Nonetheless, Csound does

[6] http://www.csounds.com/

not provide sophisticated mechanisms to temporally relate instrument, control and audio variables; for instance, to say that one microsecond after an audio signal reaches a peak, a control variable changes its value, causing three instruments to play a note whose duration is the distance between such peak and the last peak the audio signal reached.

3.1.7 Heterogeneous systems

A *heterogeneous system* combines subsystems from different computational models such as *continuous time models*, *asynchronous message passing* and *synchronous message passing*. Continuous time is often used by analog systems and it is represented by real numbers, whereas discrete time is represented by integers and can be obtained from the continuous time models by performing sampling.

According to Lee *et al.* [Eker 2003], there are different computational models: *continuous time* models based on ordinary differential equations, *discrete event* models for digital circuits networks and traffic control, *synchronous reactive* model with discrete ticks, *synchronous message passing* like the *pi-calculus* [Milner 1999], and *asynchronous message passing* such as asynchronous dataflow.

A programming language for heterogeneous systems is *Ptolemy* [Lee 2001]. Ptolemy's key feature is the assemblage of components from different paradigms. In the continuous time domain, signals can communicate with finite state machines. Ptolemy extends synchronous dataflow with asynchrony: Process networks communicate by sending messages through channels that can have buffers. Ptolemy also supports synchronous reactive models where discrete signals take values every clock tick. Unfortunately, Ptolemy is not meant for real-time interaction and it does not allow to represent a timeline of temporal objects; for that reason, it has not been used in multimedia interaction.

There is another approach proposed by Kim and Ha [Soonhoi 1999]. They combine a dataflow model that is controlled by a finite state machine. The finite state machine can send messages to change the state of the dataflow model at any time. As an example, they modeled a mp3 decoder: a dataflow graph for decoding mp3 and a finite state machine to start, stop and suspend the decoding.

Common combinations of the models above are *finite state machine* with other models, dataflow and discrete event models, and finite state machine with dataflow. Another example is our extension of interactive scores. Our extension combines a discrete model for temporal relations and a synchronous dataflow model for signal processing.

3.1.7.1 Hybrid systems

A *hybrid system* is a dynamic system that exhibits both continuous and discrete dynamic behavior. A hybrid language developed by Lee *et al.* is *HyVisual* [Lee 2005]. HyVisual provides support for finite state machine and continuous time.

Interactive scores are not currently modeled as hybrid systems because interactive scores do not work on continuous time. At this time we want to clarify that, in multimedia interaction, the terms *continuous control* and *continuous time* often refer to software and algorithms dealing with audio and video signals. Although audio and video sampling

rate is much higher than control signals, audio and video in the computer are discrete signals because they are being sampled, thus differential equations and other techniques used for hybrid systems are rarely used, except for physical models of instruments or in acoustic models.

3.1.8 Globally asynchronous, locally synchronous (GALS)

Although circuits remain synchronous, many designs feature multiple clock domains, often running at different frequencies, according to Teehan *et al.* [Teehan 2007]. Using an asynchronous interconnection decouples the timing issues of the separate blocks. Systems employing such schemes are called *globally asynchronous, locally synchronous* (GALS). They are another interesting example of heterogeneous systems, which is closely related to the philosophy of interactive scores extended with signal processing.

Crossing clock domains is the central problem in GALS design. GALS design is often used in circuits to facilitate fast block reuse by providing wrapper circuits to handle interblock communication across clock domain boundaries. Nonetheless, we believe that GALS approach could be very useful in software design as well; for instance, in multimedia systems where user gestures, discrete event control and signal processing work on different sampling rates and they need to be synchronized.

Teehan *et al.* identified three broad categories of GALS design, as shown in Figure 3.1: *pausible clock, asynchronous clock*, and *loosely synchronous*.

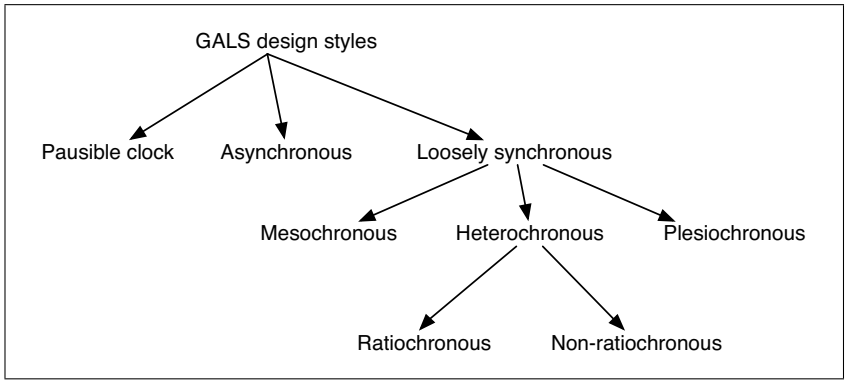

Figure 3.1: Taxonomy of broad categories in GALS design.

The pausible clock design relies on locally generated clocks that can be stretched or paused to prevent the transmitter or the receiver stall because of a full or empty channel. The asynchronous design style involves the general case in which no timing relation between the synchronous clocks is assumed. The loosely synchronous design style is for cases in which there is a well-defined, dependable relationship, between clocks.

Teehan *et al.*, argues that there are three types of loosely synchronous design: (1) *mesochronous*, in which the sender and receiver operate at exactly the same frequency with an unknown stable phase difference, for instance, *Intel's 80-core processors*; (2) *plesiochronous*, in which the sender and receiver operate at the same nominal frequency but may have a slight frequency mismatch, for instance, *gigabit ethernet*; and (3) *hete-*

rochronous, in which the sender and receiver operate at nominally different clock frequencies.

An interesting subset of hetereochronous relationships is the case of *rationally related clock frequencies* or *ratiochronous*, in which the receiver's clock frequency is an exact rational multiple fo the sender's, and both are derived from the same source clock such that there is a predictable periodic phase relationship. We believe, that translating GALS design taxonomy to software, interactive scores extended with signal processing belong the *ratiochronous* design style.

There are some advantages and disadvantages of each design technique, according to Teehan *et al*. Pausible clocks prevent dissipating dynamic power; hence, it is useful in power-critical designs. Asynchronous interfaces suffer from low throughput, but this limitation can be overcame with careful designs. Finally, loosely synchronous interfaces require timing analysis on the paths between the sender and receiver and is less amenable to dynamic changes in the clock frequency. In rationchronous designs, it is possible to use binary-rate multipliers to generate an approximation of the other clock. In general, loosely synchronous techniques offer the highest performance by removing synchronization delays from latency-critical paths; however, these methods required timing analyses that are not provided by standard computer-aided circuit design tools.

3.1.9 Petri nets

Petri nets is a natural extension of automata for concurrency with an intuitive graphical notation. *Hierarchical time stream Petri nets* (HTSPN) is an extension of Petri nets developed to model hypermedia systems [Sénac 1995]. As an example, HTSPN has been used to model synchronization between video and audio streams. As another example, Allombert *et al.* used HTSPN to model interactive scores, as we describe in Section 3.2. An interesting feature of petri nets is verification of properties.

Popova has intensively studied the properties of liveness and boundedness in time Petri nets [Popova-Zeugmann 1999]. Such a problem could be used to detect dead-locks in a multimedia scenario; however, there is not much research on properties –such as liveness, boundedness and reachability– for HTSPN. In addition, Boyer *et al.* proved that HTSPN are not equivalent to time Petri nets [Boyer 1999] and most available tools for temporal extensions of Petri nets are meant for time Petri nets.

3.1.10 Statecharts

The most used formalism for concurrent process synchronization is *statecharts*, as stated by Labiak *et al.* [Labiak 2008]. A reason is that statecharts is closely related to UML, which is frequently used in the industry. According to Labiak *et al.*, Petri nets is well-known by scientists but unknown in the industry.

Statecharts is a semi-formal model for concurrent systems, thus semantics for statecharts are not completely formalized as Petri nets. Statecharts have, arguably, some advantages over Petri nets semantics. First, process synchronization can be improved with global variables. Second, transitions can have boolean predicates. Third, the modularity

available in hierarchical Petri nets is very useful, but a transition crossing the state border is forbidden and broadcast events too. A solution for the third problem in statecharts is to have a global variable to "broadcast" an event; however, global variables are not good for modularity nor clarity.

Statecharts are preferred to Petri nets because of Petri nets lack of abstract modeling. In Petri nets, we are forced to deal with small details such as places and transitions. Such an approach is not good for a top-down approach, which is the motivation of statecharts according to Labiak *et al.* [Labiak 2004]. Nevertheless, there are hierarchical extensions for Petri nets (e.g., HTSPN), but they are not well developed; in particular, to formally verify properties.

3.1.11 Process calculi

Process calculi (or process algebras) are a diverse family of related approaches to formally model concurrent systems. Process calculi provide high-level description of interactions, communications, and synchronizations between a collection of independent processes. They also provide algebraic laws that allow process descriptions to be manipulated and analyzed, and permit formal reasoning about equivalences between processes; for instance, using *bisimulation* [Sangiorgi 2012]. Intuitively, two systems are bisimilar if they match each other's moves. In this sense, each of the systems cannot be distinguished from the other by an observer. A well-known process calculus is the pi-calculus. Unfortunately, the pi-calculus is not well suited to model reactive systems with partial information.

Concurrent constraint programming (CCP) [Saraswat 1992] is a process calculus to model systems with partial information. In CCP, a system is modeled as a collection of concurrent processes whose interaction behavior is based on the information (represented by constraints) contained in a *global store*. Formally, CCP is based on the idea of a *constraint system*. A constraint system is composed of a set of (basic) constraints and an entailment relation specifying constraints that can be deduced from others.

Although constraint systems suppose a big flexibility and modeling power for concurrent systems, Garavel argues that models based on process calculi have not found widespread use because there are many calculi and many variants for each calculus, making difficult to choose the most appropriate [Garavel 2008]. In addition, he argues that it is difficult to express an explicit notion of time and real-time requirements in process calculi. Finally, Garavel argues that existing tools for process calculi are not user-friendly and there are not many tools available.

A position in favor of process calculi is defended by Olarte *et al.* [Olarte 2008, Olarte 2011]. They showed that CCP calculi have been used in several applications such as multimedia interaction, security protocols and systemic biology. They explained that CCP has different variants to model mobility, probabilistic behavior, hybrid systems, discrete time and real-time.

We also argue, in favor of CCP, as we discussed in Chapter 1, that there has been a growing interest for CCP models of multimedia interaction in the last decade [Rueda 2002, Rueda 2001, Rueda 2004, Rueda 2005b, Rueda 2006, Allombert 2006, Toro 2009, Olarte 2009b, Olarte 2011, Toro 2012b]. CCP processes can be analyzed from

both a behavioral and declarative point of view, making them suitable for simulation and for verification of properties. Some programming languages have also been developed following the concepts of CCP. As an example *Mozart/Oz* [Roy 2004, Van Roy 2004] is a multiparadigm programming language inspired in the CCP paradigm.

Although there are programming languages based on CCP, as Garavel argued, the explicit notion of time is missing in most process calculi and, unfortunately, it is also the case of CCP. In CCP it is not possible to delete nor change information accumulated in the store. For that reason, it is difficult to perceive a notion of discrete time, useful to model reactive systems communicating with an external environment (e.g., motion sensors and speakers).

The temporal concurrent constraint (*tcc*) [Saraswat 1994] calculus circumvents this limitation by introducing the notion of discrete time as a sequence of *time units*. At each time unit, a CCP computation takes place, starting with an empty store (or one that has been given some information by the environment). In fact, tcc has been shown to be very expressive to model synchronous languages such as Lustre and Esterel [Tini 1999]. There is also an interpreter to execute tcc models [T. Sjoland 2001].

The non-deterministic timed concurrent constraint (*ntcc*) [Nielsen 2002] adds non-determinism and asynchrony to tcc. Ntcc has been extendedly used for musical applications. We chose ntcc to express operational semantics of interactive scores because it allows for verification of temporal properties; for instance, it has been used to model music improvisation systems and a western-african music problem [Rueda 2002, Rueda 2004]. In addition, there is a real-time capable interpreter for ntcc [Toro 2009], and verifications tools and techniques are being developed in the recently started Colciencia's REACT+ project[7]. Finally, another advantage of ntcc is that it handles very naturally temporal constraints.

3.1.12 Temporal constraints

Temporal constraints have gained interest among scientists ever since the invention of artificial intelligence. Temporal constraints are often used for temporal planing of autonomous robots. Lately, the multimedia interaction community developed an interest on temporal constraints for the design of interactive multimedia.

There are two well-known types of temporal constraints: *metric (or quantitative) constraints* and *qualitative constraints*. Metric constraints restrict the distance between points and qualitative constraints are relative positions. A metric constraint is, for instance, "a point occurs five time units after another", and a qualitative constraint is, for instance, "a point occurs strictly before another".

There are some well-known classes of qualitative constraints: *interval-interval* (also known as *Allen's relations* [Allen 1983], shown in Figure 3.2), *point-to-point* and *point-interval*. Interval-interval temporal relations were conceived to model dense (continuos) time, but they can also be used for discrete time. According to Gennari, point-to-point are more expressive than point-interval relations when interval-interval does not include disjunction. When interval-interval temporal relations include disjunctions, they are more expressive than the other classes, but its satisfiability is NP-Hard [Gennari 1998].

[7] REACT+ is a colombian project supported by *Colciencias* to develop verification and simulation tools for ntcc calculi. http://cic.javerianacali.edu.co/wiki/doku.php?id=grupos:avispa:react-plus.

There are also some well-known classes of quantitative constraints: *unary constraints* and *binary constraints*. They express location and distance respectively, both concepts important in music, but useless without the concept of relative positions.

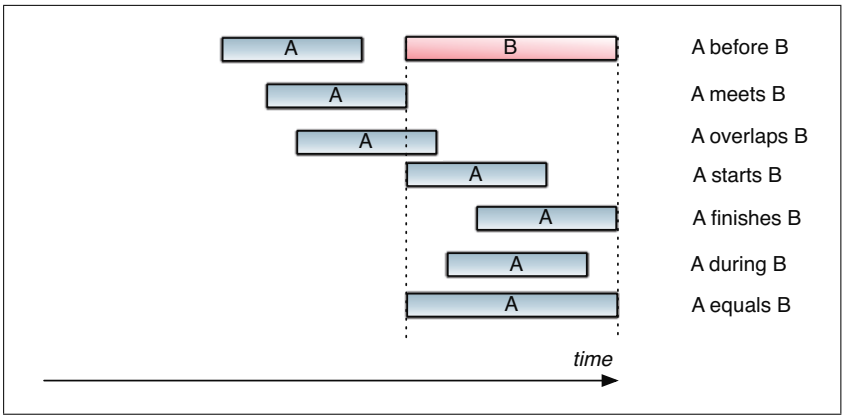

Figure 3.2: Allen's interval-to-interval relations.

Fortunately, Meiri formalized a new class: the combination of both qualitative and metric constraints [Meiri 1996]. Meiri's approach is simple: qualitative constraints can be represented as quantitative constraints; for instance, the relation $<$ can be represented by the interval $(0,\infty)$. A subset of Meiri's new class can be represented as a *simple temporal problem* [Dechter 1991] when each temporal constraint is given by a single interval. We introduce the simple temporal problem and its relation to interactive scores in Chapter 5. In interactive scores, we combine point-to-point qualitative relations with unary and binary quantitative constraints, as proposed by Meiri.

There is another kind of temporal constraints: *hierarchical unification-based temporal pattern grammar* [Biundo 2004]. The unification-based temporal grammar is meant to describe multivariable time series. Such a grammar is an extension of context-free grammars with *Prolog clauses* evaluated as side conditions. Temporal patterns use logical disjunction and they have been successfully applied to the recognition of sleeping disorders. It has also been used to data mining hierarchical temporal patterns in multivariable time series. Nonetheless, Biundo *et al.*'s temporal grammar is not meant for real-time operations.

3.2 Previous Models of Interactive Scores

The idea of temporal relations among temporal objects was introduced by Beurivé and Desainte-Catherine in [Beurivé 2001]. They found out that relative times are a more efficient representation than absolute times for music scores. Soon after, they developed *Boxes*: a software to model a hierarchy and temporal constraints [Beurivé 2001]. In fact, Boxes uses Allen's relations to describe temporal constraints. A few years later, Desainte-Catherine and Brousse came up with the idea of the interactive scores formalism [Desainte-Catherine 2003].

Another system dealing with a hierarchy of temporal objects is *Maquettes of OpenMu-*

sic [Bresson 2005]. However, we argue that *OpenMusic* [Bresson 2011] is a software for composition and is not meant for real-time interaction. Allombert and Desainte-Catherine figured out that the multimedia interaction community needed a software for composition capable of describing a hierarchy of temporal objects and capable of real-time interaction! In 2005, they introduced a new model of interactive scores [Allombert 2005], extending the previous model developed by Desainte-Catherine and Brousse, and following the concepts of Haury's meta-instrument [Haury 2008]. This model admits modification of the starting and ending times of the notes of the score during execution.

In Allombert and Desainte-Catherine's new model, a score is composed by temporal objects, interactive events and temporal relations. This approach does not allow to define interactive user events inside the hierarchy, as we can do it today. They extended Allen's relations with quantitative relations to express the duration of temporal objects in a similar manner as Mieri did it back in 1995. They introduced the very first notions of temporal reduction: intervals can be reduced if an event is launched before its nominal (expected) time and intervals can be extended if the event is launched after its nominal time; however, the operational semantics of the temporal objects with nominal times, was not very well defined back then. They also introduced a semantics based on Petri nets. Finally, they introduced the *environment, control, output (*ECO*)* machine: an abstract machine to execute an interactive score in real-time.

Allombert, Desainte-Catherine and Assayag presented a new extension in 2007 [Allombert 2007]. They changed the definition of a score: A score is defined as a pair $\langle T, R \rangle$ where T is a set of temporal objects and R a set of temporal constraints. This new definition considers an interactive user event as a kind of temporal object, thus they are included in the hierarchy, as opposed to the extension they presented in 2005. They also argued that interactive scores must have two modes: the edition mode, which they implemented using constraint propagation, and the execution mode, which they made using Petri nets. The edition model is a linear constraint satisfaction problem with a cyclic constraint graph, according to Allombert *et al.*

In the extension of interactive scores developed in 2007, Allombert *et al.* realized that some transformations were needed to the Petri nets to execute them properly. They proposed to collapse two places that occur at the same time in the same place (state). Those transformations inspired what we call in this dissertation the *normal form*. They also introduced global constraints, but not the details on how to implement them. They also developed an implementation using OpenMusic. The implementation they made in OpenMusic will be the base for the *Iscore* library, developed one year after.

In 2008, Allombert *et al.* developed a new extension of interactive scores [Allombert 2008d]. They introduced a new kind of temporal relations: linear constraints over durations; for instance, to say that the duration of an object is k times bigger than another. They made an implementation in OpenMusic that only includes flexible-time durations and does not include linear constraints. Examples of their quantitative relations are those involving a proportional or explicit duration; for instance, "the duration of A is one third of the duration of B" or "the duration of A is 3 seconds". Examples of their qualitative temporal relations are those to represent the precedence between the start and end points of two temporal objects; for instance, "A must be played during B" or "C must be

played after *D*". They also improved the concept of *temporal reductions*: left reductions (chronological) and right reductions (anti-chronological). Temporal reductions are a mechanism to reduce or stretch the duration of a temporal object when an interactive event is, respectively, delayed or speeded up, while respecting the temporal constraints of the score.

It was most likely that they realized at that time that including linear constraints over the duration of the temporal objects will change the complexity of the satisfiability and dispatching of the temporal constraints because they could no longer represent the temporal constraints as a simple temporal problem. Constraints over the durations of temporal objects were never again presented in interactive scores models.

Allombert *et al.* explored other alternatives to Petri nets as semantics for interactive scores. After reading all the previous extensions of interactive scores, Rueda had in mind that a process calculus based on constraint programming would be more appropriate to represent temporal constraints (and even other constraints, such as harmonic and rhythmical) than Petri nets. Rueda worked with Allombert, Assayag and Desainte-Catherine to develop a model based on ntcc in 2006 [Allombert 2006]. They used Allen's relations as temporal relations. There is a disadvantage: The model does not consider the problems that arises when two objects are constraint to start at the same time nor the problems associated to dispatching efficiently a simple temporal problem, as described by Muscettola *et al.* [Muscettola 1998]. We explain this last problem in detail in Chapter 6.

Sarria found another disadvantage with the ntcc model of interactive scores developed by Rueda *et al.*: time units in ntcc may have different (unpredictable) durations. Sarria extended Allombert's model in his Ph.D thesis in 2008. He proposed a different approach to cope with real-time issues using his own CCP variant, *the real-time concurrent constraint (rtcc)* calculus [Sarria 2008]. Rtcc is an extension of ntcc capable of modeling time units with fixed duration. This new calculus is capable of interrupting a process when a constraint can be inferred from the store. Rtcc is also capable of delays within a single time unit.

Olarte *et al.* also extended Rueda's ntcc model. They extended the model to change the hierarchy of temporal objects during execution [Olarte 2009b]. The spirit of such a model is different: they focus on changing the structure of the score during execution to allow the user to "improvise" on a written piece, whereas we are interested on a simpler model that we can execute in real-time. It is worth noticing that it may be also possible to model such changes in the structure during execution using a special kind of Petri nets in which tokens are also nets, introduced by Köhler *et al.* [Köhler 2003].

Finally, in 2009, Allombert explained in his Ph.D. the results published previously in his models [Allombert 2009]. He also introduced some ideas on how to deal with durations of arbitrary intervals, he introduced music processes that can be associated to temporal objects, and he introduced conditional branching. Conditional branching is the base for some non-linear models in music. Non-linear models are used to create openworks. We recall from Chapter 1 that open works can have openness of interpretation or openness of semantic content, as explained by Vickery [Vickery 2003].

Allombert presented in his thesis conditional branching and temporal relations separately, but he did not show an unified way to represent conditional branching together with temporal relations in the same scenario. His work on conditional branching was partially based on previous results developed during Ranaivoson's M.Sc. thesis in 2009

[Ranaivoson 2009]. These two works are the base of our conditional branching extension.

3.3 Software for Interactive Scores

In what follows we describe existing software for interactive scores.

Iscore. *Iscore* is a library developed by Allombert *et al.* that implements the ECO machine to execute interactive scores. It was originally developed in Lisp, and then it was ported to C++ during the ANR *Virage*[8] project in 2008. Allombert *et al.* introduced Iscore as a new tool that replaces Boxes [Allombert 2008b]. The comparison with Boxes is given in detail in [Allombert 2008c]. Iscore uses Petri nets as its underlying model because Allombert argued that solving constraint satisfaction problems during execution may be incompatible with real time [Allombert 2009]. The first implementation of Iscore uses the OpenMusic Maquettes environment and the constraint solving library Gecode in the edition mode. During execution, OpenMusic communicates with Max or Pd. Max and Pd are in charge of the contents of the temporal objects. The communication is done using the *open sound control (*OSC*)* protocol. The library was ported to C++ during the project *Virage* and it is currently being used by *Acoumouscribe*.

Virage. Virage is a software that uses Iscore and provides a user-friendly interface for edition and execution of interactive scores [Allombert 2010]. It was designed for interactive theater performances, but it can also be used for Electroacoustic music. Recently, Marczac *et al.* describe an extension of Virage with *fast forward* and *go to jumps* functionalities [Marczak 2011]. *Fast forward* is used to modify the execution speed of the score, and *go to jumps* can be seen as very fast a acceleration in which the artist do not want intermediate values.

Acousmouscribe. "The Acousmoscribe is a free software coming from the former software, *Boxes*, which aim was to write scores and compose electroacoustic music. Acousmoscribe is built around two possible uses: notation and composition. This software offers concrete and symbolic approaches of electroacoustic music at the same time. The user interface allows the writing of electroacoustic music scores, following the phenomenological approach initiated by Pierre Schaeffer. Around twenty signs, that can be combined thanks to a palette to write a "sound object", produce more than 20000 combinations: In this way, its use is intuitive while allowing quite a precise description of sounds. The length of each created box corresponds to the length of the associated sound in time. Regarding composition, a software built in Max/MSP named Acousmosynth receives messages

[8] ANR Virage Project Virage was a research platform project that aimed at developing new writing and management interfaces for artistic creation and cultural industries. This platform included businesses (JazzMutant, Blue Yeti, RSF), academic laboratories (LIMSI-CNRS Paris Sud, MSH Paris Nord-CICM, LaBRI Bordeaux) and artists (GMEA, the Albi-Tarn centre national de création musicale and didascalie.net).

from Acousmoscribe thanks to the open sound control protocol, and translates its symbolic notation into control parameters for audio synthesis modules." [9]

i-score. The latest software for interactive scores is *i-score*. This software combines the edition interface of Acousmouscribe with the execution model of Virage. It is currently maintained by *Scrime*[10] and distributed as *opensource*.

3.4 Summary and Discussion

In this chapter we discussed several software and formalisms related to interactive scores and we discussed existing models and implementations of interactive scores.

We described sequencers which are software to design multimedia interaction. Sequencers are usually based on a fixed timeline or on cue lists. Some software provide both time models but they are temporally unrelated. An advantage of interactive scores is to relate temporally both time models and to model conditional branching.

There is also hardware to control multimedia interaction. Meta-instruments are musician-machine gesture transducers intended for controlling a program in real-time. As an example, a meta-instrument can control the start and end of groups of notes, allowing for the interpretation of complex pieces with interfaces as simples as a one-touch piano. This work inspired the first models of interactive scores. In contrast, there are score-following systems, in which a real-instrument is needed. A score-following system tracks the performance and plays electronics associated to the notes of the score.

There are also synchronous and asynchronous dataflow paradigms, which are paradigms closely related to interactive scores. Asynchronous dataflow is meant to handle asynchronous events such as user interactions, whereas synchronous languages are meant to design real-time applications and they are based on a model of deterministic concurrency. Heterogeneous systems are systems that combines several paradigms, for instance, asynchronous and synchronous languages.

Heterogeneous systems combining asynchronous and synchronous circuits can be designed using schemes such as global asynchronous, locally synchronous. A special case, of interest for interactive scores is called *ratiochronous*, in which the receiver's clock frequency is an exact multiple to the sender's, and both are derived from the same source clock. This design scheme could be useful to synchronize interactive scores with a signal processing system, but also with other systems such as a score following system.

Process calculi are approaches to formally model concurrent systems. As an example, ntcc describes partial information by the means of constraints, it provides discrete time units, and it models asynchrony and non-determinism. Ntcc has been used in the past to model interactive scores. It handles naturally temporal constraints. A similar approach is Petri nets, which is is another model of concurrency with an intuitive graphical notation. An extension of Petri nets with time and hierarchy has been used to model interactive scores

[9] http://scrime.labri.fr/index.php?option=com_content&view=article&id=11%3Aacousmoscribe&catid=41%3Athemesderecherche&Itemid=81&lang=en

[10] http://scrime.labri.fr/

in the past and for synchronization of multimedia streaming systems. Ntcc has also been used to model interactive scores

There are other existing models of interactive scores. First models were conceived to control the starting and ending times of the notes of a score. They also included different temporal relations; for instance, to model two temporal objects that overlaps, by the means of Allen's relations. Later extensions included a Petri nets operational semantics. Finally, there are extensions of interactive scores with conditional branching. Note that the Petri nets semantics of interactive scores were implemented in an efficient C++ library called Iscore, and it is currently being used by i-score.

CHAPTER 4

Background

Contents

In this chapter, we discuss three subjects that are crucial to understand the rest of this dissertation: *timed event structures*, the *non-deterministic timed concurrent constraint* calculus and the *Functional AUdio Stream* language.

4.1 Timed Event Structures (TES)

Langerak's *timed event structures* is a mathematical model to represent systems with non-determinism, real-time and concurrency [Baier 1998]. Henceforth, we call them *event structures*. The reader should not confuse these timed event structures with the real-time event structures proposed by Dubtsov *et al.* in [Virbitskaite 2008]. In what follows we introduce a simplified definition of event structures in which there are no conflicts among events: this definition is used in the semantics presented in Chapters 5 and 6. Afterwards, we introduce a definition of event structures with conflicts: this definition is used for the conditional branching semantics in Chapter 7.

We recall from Chapter 1 that event structures include a set of *labeled events* and a *bundle delay relation* (denoted $\mapsto$). The bundle delay relation establishes which events must happen before some other occurs. Actions can be associated to events. Events are unique, but two events may perform the same action. Events can be defined to be "urgent". An *urgent event* occurs as soon as it is enabled. In addition to the bundle relation, event structures include a *conflict relation* (denoted $\rightsquigarrow$) establishing events that cannot occur together. Events can also be given *absolute occurrence times*.

4.1.1 Event Structures without conflicts

In this section, we present simplified versions of the definitions from [Baier 1998]. All events are urgent, there is no conflict relation and absolute occurrence times are in the interval $[0,\infty)$. We represent delays as subsets of $\mathbb{N}\cup\{\infty\}$, originally defined with real numbers. All bundle delays are between a given event and some other; therefore, we will call them *event delays*. The causality relation is implicitly represented in the event delay function.

Definition 4.1.1 (Event Structure (TES))**.** *An* event structure ε *is a tuple* $\langle E,l,R\rangle$ *with*

- E*, a set of events,*
- $l : E \longrightarrow Act$*, the labelling total function,*
- $R : E \times E \longrightarrow \mathbb{P}(\mathbb{N}\cup\{\infty\})$*, the event delay function.*

We denote by the functions $\mathrm{E}(\varepsilon),\mathrm{l}(\varepsilon),\mathrm{R}(\varepsilon)$ *each component of* ε *and the set of all* TES*s by* $\mathscr{E}$*. A* timed event trace *is a sequence* $\sigma = (e_1,t_1)\ldots(e_n,t_n)$ *where* $e_i \in E$ *(all events being pair-wise distinct) and the* time point $t_i \in \mathbb{N}\cup\{\infty\}$.
For all $0 < i,j \leq n$*:* $e_j \mapsto^{\Delta} e_i \Rightarrow t_i \in t_j + \Delta$.

Example 4.1.2. An event structure without conflicts is shown in Figure 4.1.

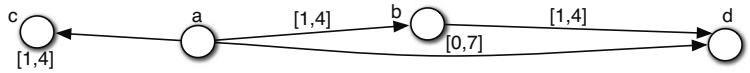

Figure 4.1: Events b and c happen between one and four time units after a. In addition, event d happens between two and eight time units after a (the intersection between $[1,4]+[1,4]$ and $[0,7]$).

We denote an event delay $R(e,e') = \Delta$ by $e \overset{\Delta}{\mapsto} e'$ or by $e \mapsto^{\Delta} e'$ and we omit $[0,\infty)$ delays. The *labeling function* for events assigns an element in the set Act to each event. The function $\mathrm{le} : E \to Act$ returns the label of an event. The *set of all timed event traces* of ε is denoted by $\mathrm{Traces}(\varepsilon)$. Timed event traces comply with causality, but not necessarily the advance of time: This means they can be well-caused but ill-timed. Nonetheless, for any ill-timed trace σ there is always a time-consistent event trace σ' that can be obtained from σ (proved in [Baier 1998]). As an example, $\sigma = (a,1),(c,4),(b,3)$ is ill-timed because $(c,4)$ appears before $(b,3)$, and $\sigma' = (a,1),(b,3),(c,4)$ is an equivalent well-timed trace. We only consider well-timed traces in the set $\mathrm{Traces}(\varepsilon)$ in this dissertation.

4.1.2 Event Structures with conflicts

In what follows, we present a more complete definition of event structures that includes a conflict relation establishing which events cannot occur together. Conflicts are useful, for instance, to model a choice between two parts of a music piece or that two parts of the piece are mutually exclusive. In fact, we use event structures with conflicts in Chapter 7 to model conditional branching.

Definition 4.1.3 (Event Structure with Conflicts)**.** *An* event structure ε *is a tuple* $\langle E, l, R, \rightsquigarrow \rangle$ *with*

- E, *a set of events,*
- $l : E \longrightarrow Act$, *the labelling total function,*
- $R : E \times E \longrightarrow \mathbb{P}(\mathbb{N} \cup \{\infty\})$, *the event delay function,*
- $\rightsquigarrow$: $E \times E$, *the conflict relation.*

such that for all $e' \in E$ *and* $e'' \in E$
(P1) $((e' \rightsquigarrow e \wedge e \rightsquigarrow e') \wedge e'' \rightsquigarrow e) \Rightarrow (e'' \mapsto e' \wedge e'' \rightsquigarrow e')$

As in Def. 4.1.1, we denote by the functions $\mathrm{E}(\varepsilon)$,$\mathrm{l}(\varepsilon)$, $\mathrm{R}(\varepsilon)$, and $\mathrm{C}(\varepsilon)$ each component of ε and the set of all TESs by $\mathscr{E}$. Figure 4.3 exemplifies property (P1), which is crucial to understand conflicts. The justification of this constraint is to be able to "locally" decide whether an event can occur by only considering its direct causal predecessors and conflicts, according to Baier *et al.*

Example 4.1.4. Figure 4.2 is an example of an event structure with conflicts.

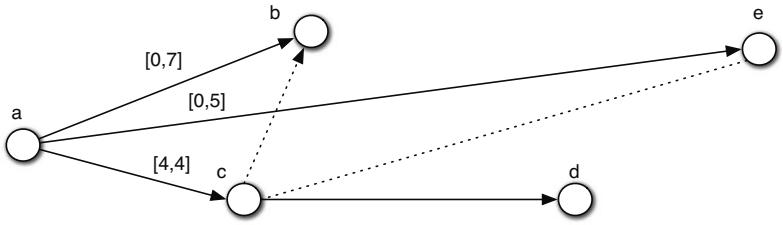

Figure 4.2: Example of an event structure with conflicts. There is an event delay between a and b, a and e, a and c, and c and d. Arrows with no label represent an event delay with a duration of $[0,\infty)$. There is a conflict from c to b, one from c to e and one from e to c. Pointed lines represent conflicts; when they have no direction, the conflict is symmetrical.

The concept of system run for event structures is also defined by a timed event trace, but it is more complex to define when there are conflicts. First, we must define the *enabled events after a sequence of events* σ. The definition is a simplified version from [Baier 1998] because we do not consider bundles. Stated in Baier *et al.*'s words, an event is enabled after σ if it is not disabled by one of the events in σ, and if any event pointing to it appears in σ.

Definition 4.1.5 (Enabled events after σ)**.** *For* σ *a sequence of distinct events, let the set of events enabled in* ε *after* σ *be defined as*

$$\mathrm{en}^{\varepsilon}(\sigma) \stackrel{def}{=} \{e \in E - \sigma | (\forall e_i \in \sigma : e \not\rightsquigarrow e_i) \wedge (\forall e_i \mapsto e : e_i \in \sigma)\}$$

To define timed event traces, we also need to define the *potential time of occurrence of an event*. Let $\mathrm{tm}^{\varepsilon}_{\sigma}(e)$ denote the set of time instants at which an enabled event e after σ could happen, given that each event e_i in σ occurred at time t_i. We present a simplified definition from [Baier 1998] because we do not consider absolute occurrence times. Event

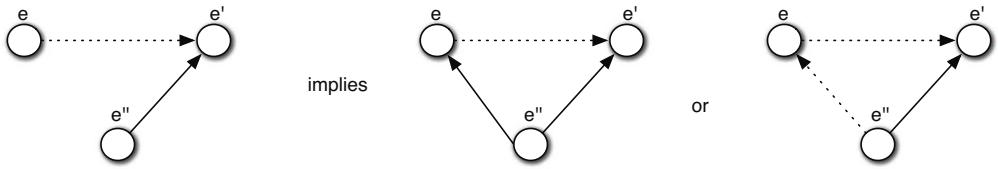

Figure 4.3: Constraint (P1) enforces that as soon as e is enabled, either e' is also enabled (provided e' is not disabled in some way), or as soon as e' occurs, e will be permanently disabled, since some event pointing to e, namely e'', is disabled by e' [Baier 1998].

e can occur if (1) for each event e_i with $e_i \rightsquigarrow e$, we have that e occurs at least at t_i, and (2) the time relative to all its immediate causal predecessors is respected. Those two conditions take care of the fact that events cannot occur before their causes, entailing that causal ordering implies a temporal ordering, according to Baier *et al*. It is easy to check that for an urgent event e, we have $\mathrm{tm}^{\varepsilon}_{\sigma}(e) = \emptyset$ or $\mathrm{tm}^{\varepsilon}_{\sigma}(e) = \{t\}$, for some $t \in \mathbb{N} \cup \{\infty\}$.

Definition 4.1.6 (Potential time of occurrence)**.** *For $\sigma = (e_1,t_1)...(e_n,t_n)$ a timed sequence of distinct events and event $e \in \mathrm{en}^{\varepsilon}(\sigma)$, let*

$$\mathrm{tm}^{\varepsilon}_{\sigma}(e) \stackrel{def}{=} \bigcap_{e_i \rightsquigarrow e} [t_i, \infty) \cap \bigcap_{e_i \stackrel{\Delta}{\mapsto} e} t_i + \Delta$$

Given the enabled events after a sequence σ and the potential time of occurrence, we can define a timed event trace of an event structure with conflicts in what follows.

Definition 4.1.7 (Timed Event Trace of an Event Structure with Conflicts)**.** *A* timed event trace *is a sequence $\sigma = (e_1,t_1)...\ (e_n,t_n)$ where $e_i \in E$ (all events being pair-wise distinct) and the* time point *$t_i \in \mathbb{N} \cup \{\infty\}$. For all $0 < i, j \leq n$:*

1. $e_j \stackrel{\Delta}{\mapsto} e_i \Rightarrow t_i \in t_j + \Delta$
2. $e_j \rightsquigarrow e_i \Rightarrow j < i \wedge t_j \leq t_i$
3. $(e_i \rightsquigarrow e \vee e \rightsquigarrow e_i) \Rightarrow t_i \leq \mathrm{tm}^{\varepsilon}_{\sigma}(e)$, *for all* $e \in \mathrm{en}^{\varepsilon}(\sigma_{i-1})$

Last constraint takes care of the fact that events may prevent the events they disable (or by which they are disabled) to occur after a certain time. That is, event e_i can occur at time t_i provided there is no enabled event e that disabled e_i (or that is disabled by e_i) and that (if it occurs) must occur before t_i, according to [Baier 1998].

Note that the results for ill-timed event traces were proved for the general case when time event structures may include conflicts; therefore, ill-timed event traces for event structures with conflicts can also be always to transformed into well-timed traces.

4.2 Non-deterministic Timed Concurrent Constraint (ntcc)

In *concurrent constraint programming* (CCP) [Saraswat 1992], a system is modeled as a collection of concurrent processes whose interaction behavior is based on the information (represented by constraints) contained in a *global store*. Formally, CCP is based on the idea of a *constraint system (*CS*)*. A constraint system is composed of a set of (basic) constraints and an entailment relation specifying constraints that can be deduced from others.

Definition 4.2.1 (Constraint System (CS)). *A* CS *is a pair* (Σ,Δ) *where* Σ *is a signature specifying constants, functions and predicate symbols, and* Δ *is a consistent first-order theory over* Σ *(i.e., a set of first-order sentences over* Σ *having at least one model)* [1].

Constraints can be though of as first-order formulae over Σ, thus there is an underlying first-order language $L = \langle \Sigma, \mathscr{V}, S \rangle$, where $\mathscr{V}$ is a countable set of variables and S is a set of logic symbols $\neg, \wedge, \vee, \Rightarrow, \forall, \exists, \texttt{true}, \texttt{false}$. We can decree that $c \vdash d$ (d can be deduced from c) if the implication $c \Rightarrow d$ is valid in Δ, as proposed by Valencia [Valencia 2002].

For operational reasons, $\vdash$ must be decidable and it is desirable for real-time purposes to be decidable in polynomial time. A commonly used constraint system is *bounded finite domain* (**FD**$[n]$) that defines arithmetic relations among variables whose domains are finite ranges of values $\{0,...,n-1\}$. Operator $\vdash$ is not decidable in polynomial time for all possible constraints in finite domain; however, for some commonly used subsets of linear constraints, $\vdash$ can be decided in polynomial time, as proved in *et al.* [Bordeaux 2011]. In this dissertation, we use a *bounded finite domain* CS, providing arithmetic relations over natural numbers. As an example, using a finite domain CS, we can deduce $pitch \neq 60$ from the constraints $pitch > 40$ and $pitch < 59$.

Although CCP has the advantage of dealing with partial information and allowing several constraint systems in the same model, it cannot easily represent the advance of time, useful to model reactive systems. The *non-deterministic timed concurrent constraint* (*ntcc*) [Nielsen 2002] calculus circumvents this limitation by introducing the notion of discrete time as a sequence of time units. At each time unit, a CCP computation takes place, starting with an empty store (or one that has been given some information by the environment). Concurrent constraints processes operate on this store as in the usual CCP model to accumulate information into the store.

In ntcc, as opposed to the CCP model, processes can schedule other processes to be run in future time units. In addition, since at the beginning of each time unit a new store is created, information on the value of a variable can change (e.g., it can be forgotten) from one unit to the next. A description of ntcc processes can be found in Table 4.1.

Example 4.2.2. As an example, process **tell** $(pitch_1 = 52) \parallel$ **when** $48 < pitch_1 < 59$ **do next tell** $(instrument = 1)$ adds to the store the constraint $pitch_1 = 52$ and postpones one time unit the constraint $instrument = 1$.

[1] This definition was taken from Valencia's dissertation [Valencia 2002].

Process	Meaning
tell (c)	Adds c to the current store
when c **do** A	If c holds now, run A now
local x **in** P	Runs P with local variable x
$A \parallel B$	Parallel composition
next A	Runs A at the next time unit
unless c **next** A	Unless c holds now, next run A
$\sum_{i \in I}$ **when** c_i **do** P_i	Chooses P_i such that c_i holds
$!P$	Executes P each time unit
$*P$	Executes P eventually

Table 4.1: Intuitive semantics of the ntcc processes.

4.2.1 Examples in multimedia interaction

In what follows, we present some examples of the ntcc processes modeling multimedia interaction. We represent pitches by natural numbers, following the *musical instrument digital interface (*MIDI*)* standard.

- Using *tell* it is possible to add constraints to the store such as **tell**$(60 < pitch_2 < 100)$, which means that $pitch_2$ is an integer between 60 and 100.

- Process *when* can be used to describe how the system reacts to events; for instance, **when** $pitch_1 = c4 \wedge pitch_2 = E4 \wedge pitch_3 = g4$ **do tell**$(cmayor = true)$ adds the constraint $cmayor = true$ to the current store as soon as the pitch sequence C, E, G has been played.

- *Parallel composition* ($\parallel$) makes it possible to represent concurrent processes; for instance, **tell** $(pitch_1 = 52) \parallel$ **when** $48 < pitch_1 < 59$ **do tell** $(Instrument = 1)$ tells the store that $pitch_1$ is 52 and concurrently assigns the *instrument* to one, since $pitch_1$ is in the desired interval, as shown in Figure 4.4.

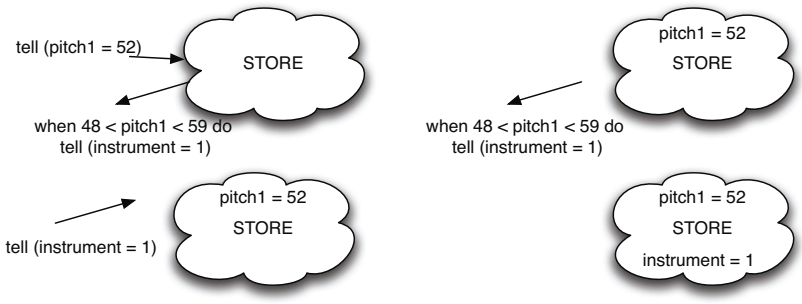

Figure 4.4: An example of ntcc processes.

- Process *next* is useful when we want to model variables changing over time; for instance, **when** $(pitch_1 = 60)$ **do next tell** $(pitch_1 <> 60)$ means that if $pitch_1$ is equal to 60 in the current time unit, it will be different from 60 in the next time unit.

- Process *unless* is useful to model systems reacting when a condition is not satisfied or when the condition cannot be deduced from the store; for instance, **unless** $(pitch_1 = 60)$ **next tell** $(lastPitch <> 60)$ reacts when $pitch_1 = 60$ is false or when it cannot be deduced from the store (e.g., note labeled by $pitch_1$ was not played in the current time unit).

- Process *star* (*) can be used to delay the end of a process indefinitely, but not forever; for instance, $*$**tell** (end). This process is used to represent temporal non-determinism. Note that to model interactive scores we do not use *star*; instead we use a bounded-time version.

- After its execution, *Bang* (**!**) executes a process every time unit; for instance, **!tell** $(c_4 = 60)$.

- Process *sum* $(\sum)$ is used to model non-deterministic choices; for instance, $\sum_{i\in\{48,52,55\}}$ **when** $i \in PlayedPitches$ **do tell** $(pitch = i)$ chooses a note among those played previously that belongs to the C major chord (composed by pitched 48, 52 and 55).

Recursion can be defined with the basic processes described above. In ntcc, recursion can be defined with the form $q(x) =^{def} P_q$, where q is the process name and P_q is restricted to call q at most once and such call must be within the scope of a *next*. The reason of using *next* is that ntcc does not allow recursion within a time unit.The reader should not confuse a *simple definition* with a *recursive definition*; for instance, $Interval_{i,j,\Delta} =^{def}$ **tell**$(P_i + \Delta < P_j)$ is a simple definition where the values of i, Δ and j are replaced "statically", like a macro in a programming language. Instead, a recursive definition such as $Clock(v) =^{def}$ **tell**$(clock = v) \| $**next** $Clock(v+1)$ is like a recursive function in a programming language.

4.2.2 Operational Semantics

Ntcc has two type of transitions: internal transitions that occur within a time unit $(\longrightarrow)$ and observable transitions that occur from one time unit to the other $(\Longrightarrow)$. A rule states that whenever the conditions have been obtained in the course of some derivation, the conclusion is also obtained, as shown in Table 4.2.

Rule PAR states that whenever a process P can evolve to a process P' by performing an internal transition, it can also evolve from $P \| Q$ to $P' \| Q$. Rules TELL, SUM and UNL are interpreted in a similar way to rule PAR. Rule REP specifies that process $!P$ produces a copy of P at the current time unit, and then persists in the next time unit. Rule STR says that structurally congruent (defined in detail in [Nielsen 2002]) configurations have the same reductions. Rule ABO realizes the intuition of a process *abort* causing interaction with the environment to cease by generating infinite sequences of internal transitions. Process *abort* is used to define an equivalence between ntcc processes and its logic. Rule LOC

TELL	$\dfrac{}{(\mathbf{tell}(c),d) \longrightarrow (\mathbf{skip}, d \wedge c)}$	PAR	$\dfrac{(P,c) \longrightarrow (P',d)}{(P \Vert Q,c) \longrightarrow (P' \Vert Q,d)}$
REP	$\dfrac{}{(!P,d) \longrightarrow (P \Vert \mathbf{next}\ !P,d)}$	STR	$\dfrac{\gamma_1 \longrightarrow \gamma_2}{\gamma_1' \longrightarrow \gamma_2'}$ **if** $\gamma_1 \equiv \gamma_1'$ and $\gamma_2 \equiv \gamma_2'$
SUM	$\dfrac{}{(\sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i,d) \longrightarrow (P_j,d)}$ **if** $d \models c_j, j \in I$		
UNL	$\dfrac{}{(\mathbf{unless}\ c\ \mathbf{next}\ P,d) \longrightarrow (\mathbf{skip},d)}$ **if** $d \models c$		
ABO	$\dfrac{}{(\mathbf{abort},d) \longrightarrow (\mathbf{abort},d)}$		
LOC	$\dfrac{(P, c \wedge \exists_x d) \longrightarrow (P', c' \wedge \exists_x d)}{((\mathbf{local}\ x,c)P,d) \longrightarrow ((\mathbf{local}\ x,c')P', d \wedge \exists_x c')}$		
OBS	$\dfrac{(P,c) \longrightarrow^* (Q,d) \not\longrightarrow}{P \overset{(c,d)}{\Longrightarrow} R}$ **if** $R \equiv F(Q)$		

Table 4.2: Operational semantics of ntcc.

explains how x can have a local scope in the sense of local variables in programming languages. To define operational semantics, Valencia *et al.* extended the syntax with a construct **local** (x,d) **in** P, to represent the evolution of a process **local** x **in** Q, where d is the local information produced in this evolution. The rule for asynchrony (*) provided by ntcc is omitted in this section for simplicity because we will not use it in our models nor implementations in this dissertation.

Rule OBS says that an observable transition from P labeled by (c,d) is obtained by performing a sequence of internal transitions from the initial configuration (P,c) to a final configuration (Q,d) in which no further internal evolution is possible. We use c,d throughout this thesis to represent constraint stores. The residual process R to be executed in the next time unit is equivalent to the process $F(Q)$, where the future function $F : Proc \rightarrow Proc$ is the partial function defined by

$$F(Q) = \begin{cases} \mathbf{skip} & if\ Q = \sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ Q_i \\ F(Q_1) \Vert F(Q_2) & if\ Q = Q_1 \Vert Q_2 \\ \mathbf{local}\ x \mathbf{in}\ F(R) & if\ Q = \mathbf{local}\ x\ \mathbf{in}\ R \\ R & if\ Q = \mathbf{next}\ R\ or\ Q = \mathbf{unless}\ c\ \mathbf{next}\ R \end{cases}$$

The process $F(Q)$, defined above, is obtained by removing from Q summations that did not trigger activity within the current time unit and any local information which has been stored in Q. Process *skip* is a short form for an empty summation: a process that does not perform any transition.

Process	Definition
$x:(z)$	**tell**$(x=z) \parallel$ **unless** change(x) **next** $x:(z)$
$x :\leftarrow g(x)$	**local** v **in** $\sum_{v\in\{0\ldots n\}}$ **when** $x=v$ **do** (**tell** change$(x) \parallel$ **next** $x:g(v)$)
$exch_g[x,y]$	**local** v **in** $\sum_{v\in\{0\ldots n\}}$ **when** $t=v$ **do** (**tell**(change$(x) \parallel$ (**tell**(change(y) $\parallel$ **next** $(x:g(v) \parallel y:(v))$

Table 4.3: Definition of cells. Cells represent variables whose value may be re-assigned in a different time unit.

4.2.2.1 Derived operators

There are bounded-time versions of $*P$ and $!P$. Process

$$*_{[0,n]}P \stackrel{def}{=} P + \textbf{next}\ P \ldots + \textbf{next}^n\ P$$

launches a process P at some time unit within the interval $[0,n]$. Similarly,

$$!_{[0,n]}P \stackrel{def}{=} P \| \textbf{next}\ P \ldots \| \textbf{next}^n\ P$$

launches a process P all time units within the interval $[0,n]$.

An important concept in reactive systems is the notion of persistence: to assign a certain value to a value for the rest of the execution. Process

$$x \leftarrow y \stackrel{def}{=} \sum_{v\in[0,n_\infty]} \textbf{when}\ v=y\ \textbf{do}\ \textbf{!tell}\ (x=v)$$

assigns persistently the current value of y to x, where n_∞ is the biggest number available in the constraint system. Processes in Table 4.3 are used to model cells. Cells are variables which value can be re-assigned in terms of its previous value. Process $x:(z)$ creates a new cell x with initial value z, process $x :\leftarrow g(x)$ changes the value of a cell (this is different from $x \leftarrow t$ which changes the value of x only once), and process $exch_g[x,y]$ exchanges the value of cell x and z. Their definition is based on recursion.

Formally, recursion is based on persistent binding [Valencia 2002]. A recursive definition is encoded as follows. Let $\widehat{P}$ the process obtained by replacing in P any call $q(t)$ with **tell**$(call(q)) \| $**tell**$(q_{arg}=t)$. The definition of a encoding of a recursive process is

$$[\![q(x) \stackrel{def}{=} P_q]\!] \stackrel{def}{=} \textbf{!}(\textbf{when}\ call(q)\ \textbf{do}\ (\textbf{local}\ x\ \textbf{in}\ x \leftarrow q_{arg} \| \widehat{P_q}).$$

Finally, the calls $q(t)$ in other processes are replaced by

$$\textbf{local}\ q, q_{arg}\ \textbf{in}\ [\![q(x) \stackrel{def}{=} P_q]\!] \| \widehat{q(t)}.$$

4.2.3 Denotational Semantics

The denotation of a ntcc process represents the *strongest post-condition* of a process. The denotation is a set of infinite sequences of stores (i.e., constraints). Each element of the sequence corresponds to each time unit. A function $[\![.]\!]$ associates such sets to each process (see Table 4.4). Notation $\exists_X \alpha$ denotes the application of $\exists_X$ to each constraint in α. Set C is the set of constraints in the constraint system. Henceforth C^ω, C^* are the set of infinite and finite sequences of constraints in C, respectively. We use α, α' to represent elements of C^ω, β to represent an element of C^*, and $\beta.\alpha$ to represent the concatenation of β and α.

D1	$[\![\textbf{tell}(c)]\!] = \{d.\alpha \mid d \vdash c, \alpha \in C^\omega\}$
D2	$[\![\sum_{i\in I} \textbf{when } c_i \textbf{ do } P_i]\!] = \bigcup_{i\in I}\{d.\alpha \mid d \vdash c_i, d.\alpha \in [\![P_i]\!]\} \cup \bigcap_{i\in I}\{d.\alpha \mid d \nvdash c_i, d.\alpha \in C^\omega\}$
D3	$[\![P \| Q]\!] = [\![P]\!] \cap [\![Q]\!]$
D4	$[\![\textbf{local } x \textbf{ in } P]\!] = \{\alpha \mid$ there exists $\alpha' \in [\![P]\!]$ s.t. $\exists_x \alpha = \exists_x \alpha'\}$
D5	$[\![\textbf{next } P]\!] = \{d.\alpha \mid d \in C, \alpha \in [\![P]\!]\}$
D6	$[\![\textbf{unless } c \textbf{ next } P]\!] = \{d.\alpha \mid d \vdash c, \alpha \in C^\omega\} \cup \{d.\alpha \mid d \nvdash c, \alpha \in [\![P]\!]\}$
D7	$[\![!P]\!] = \{\alpha \mid \forall \beta \in C^*, \alpha' \in C^\omega s.t. \alpha = \beta.\alpha'$, we have $\alpha' \in [\![P]\!]\}$

Table 4.4: Denotational semantics of ntcc.

Intuitively, $[\![P]\!]$ represents the sequences that a process P can output interacting with any possible environment. As an example, the sequences of $[[\textbf{tell}(c)]]$ are those whose first element is stronger than c (rule $D1$). Further information on the denotation of ntcc can be found in [Nielsen 2002]. We use denotational semantics for the correctness proofs presented in Appendix A.

4.2.3.1 Encoding the denotation of ntcc into an automaton

The infinite sequences of the denotation of a process can be represented as a *Büchi automaton*. Büchi automata are an extension of *finite state automata* for infinite input. Such automata accept exactly those runs in which at least one of the infinitely often occurring states is an accepting state. For simplicity, we label transitions as $c \vdash d$: There is a transition for every constraint $c \in C_{finite}$ stronger than d. The reader may find the encoding of the denotation of ntcc into Büchi in [Valencia 2005]. To define the encoding, Valencia *et al.* define a finite set of constraints (i.e., *the relevant constraints of P*) because the alphabet of a Büchi automaton must be finite. In Table 4.5, we present the relevant constraints of a process. We use the Büchi automaton representation for automatic verification in Chapter 11.

4.2.4 CLTL: A temporal logic for ntcc

Constraint linear-time logic (CLTL) is an extension of *linear-time logic* (LTL) to express properties over sequences of constraints [Valencia 2005]. It was proposed by Valencia to

Process (P)	**Relevant Constraints** (Relevant(P))
skip	$\{\texttt{true}\}$
tell(c)	$\{c\}$
$\sum_{i\in I}$ **when** c_i **do** P_i	$\bigcup_{i\in I}\{c_i\}\cup$Relevant($P_i$)
unless c **next** P	$\{c\}\cup$Revelant(P)
$P\|\|Q$	Relevant(P)$\cup$Relevant(Q)
$!P$ and **next** P	Relevant(P)
local x **in** P	$\{\exists_x c, \forall_x c \| c \in \overline{\text{Relevant}(P)}\cup\text{Relevant}(P)$

Table 4.5: Revelant constraints of a ntcc process presented in [Valencia 2005]. Given $S \subseteq C$, let $\overline{S}$ be the closure under conjunction and implication of S, the relevant constraints is a function Relevant : $Proc \to \mathbb{P}(C)$.

prove properties of ntcc processes. The reader should not confuse this logic with other extensions of LTL with constraints such as Demri *et al.*'s CLTL logic [Demri 2007].

Definition 4.2.3 (CLTL syntax [Valencia 2005])**.** *The formulae $F,G,\ldots \in \mathscr{F}$ are built from constraints $c \in C$ and variables $c \in \mathscr{V}$ in the underlying constraint system by:*

$$F,G,\ldots := c \,|\, \dot{true} \,|\, \dot{false} \,|\, F \dot{\wedge} G \,|\, F \dot{\wedge} G \,|\, \dot{\neg} F \,|\, \dot{\exists}_x F \,|\, \bigcirc F \,|\, \Box F \,|\, \Diamond F$$

At this point, we shall quote Valencia's postulate on CLTL: "Here [in CLTL] c is a constraint (i.e., a first-order formula in the underlying constraint system) representing a *state formula* c. The symbols $\dot{\texttt{true}}, \dot{\texttt{false}}, \dot{\wedge}, \dot{\wedge}, \dot{\neg}, \dot{\exists}$ represent linear-temporal logic true, false, conjunction, disjunction, negation and existential quantification. As clarified later, the dotted notation is needed as in CLTL these operators may have slight differences with the symbols $\texttt{true}, \texttt{false}, \wedge, \wedge, \neg, \exists$ in the underlying constraint system. The symbols $\bigcirc, \Box$ and $\Diamond$ denote the temporal operators *next, always* and *sometime*. Intuitively, $\bigcirc F$, $\Diamond F$ and $\Box F$ means that the property F must hold next, eventually and always, respectively" [Valencia 2005].

A formula F is a restricted-negation formula *iff* whenever $\dot{\neg} G$ appears in F and G is a state formula (i.e., $G = c$ for some c). Table 4.6 shows how to translate a restricted-negation formula into a locally-independent process. A locally-independent process is such that the guards of the sums do not contain local variables.

Example 4.2.4. Examples of CLTL formulae are "always the constraint $pitch = 60$ can be deduced from the output store", namely $\Box(pitch = 60)$; and "eventually, object a and object b are launched at the same time", namely $\Diamond(launch_a \dot{\wedge} launch_b)$.

4.3 Functional Audio STream (Faust)

Faust is a functional programming language for signal processing. In Faust, *digital signal processing* (DSP) algorithms are functions operating on signals. Graphical user interface (GUI) objects in Faust can be defined in the same way as other signals. Therefore, we

Formula F	Process $h(F)$
$\dot{\texttt{true}}$	**skip**
c	**tell**(c)
$\texttt{false}$	**abort**
$\dot{\neg}c$	**when** c **do abort**
$F \dot{\wedge} G$	$h(F) \| h(G)$
$F \dot{\wedge} G$	$h(F) + h(G)$
$\dot{\exists} x F$	**local** x **in** $h(F)$
$\bigcirc F$	**next** $h(F)$
$\Box F$	$!h(F)$
$\Diamond F$	$*h(F)$

Table 4.6: Map from restricted-negation formulae into locally-independent processes.

can control buttons, check boxes and integer inputs –originally designed for users– from another program such as a ntcc interpreter (e.g., Ntccrt), as shown in Figure 4.5.

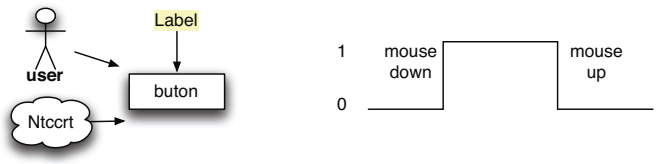

Figure 4.5: The signal delivered by the button reflects the user actions (or the Ntccrt output): one when the button is pressed; zero otherwise.

Although Faust programs can be compiled into efficient C++ programs, they are limited because all signals must have the same sampling rate. For that reason, Faust was recently extended for multirate [Jouvelot 2011]. With such an extension, Faust is capable to handle signals at different frequencies. This is useful, for instance, for scenarios with different media such as audio and video. Unfortunately, this extension is not yet implemented.

Another extension of Faust is the *Pd-Faust* interface [Gräf 2007]. This interface is useful for DSPs that cannot be efficiently implemented in Pd because of a restriction of Pd: the 1-block minimum delay for feedback loops. An example of such DSP is the *Karplus-Strong* algorithm [Orlarey 2004]. Furthermore, Pd-Faust can also be used for other DSPs.

There is another reason to choose Faust: its extension for automatic parallelization and vectorization [Orlarey 2010]. This extension has been proved to be very efficient; for instance, for the Karplus-Strong, which we will use in several examples in this Chapter 8. Orlarey *et al.* found that using automatic parallelization, a program playing 32 strings based on Karplus-Strong is twice faster [Orlarey 2010]. Automatic parallelization is possible due to Faust formal semantics.

Formal semantics of Faust are based on the *block-diagram algebra* (BDA), widely used in visual languages [Orlarey 2004]. Using the BDA, Orlarey *et al.* defined how things are

connected, but not what they do. On top of the BDA, Faust is extended with the primitives we described in Table 4.7.

4.3.1 Overview of Faust semantics

Signals in Faust are defined as discrete functions of time $s : \mathbb{N} \to \mathbb{R}$ and signal processors as functions from n-tuples of signals tu m-tuples of signals $p : \mathbb{S}^n \to \mathbb{S}^m$. Signals can be constant (i.e., they always deliver the same value). Signals can also deliver only integer values. Faust semantics are given in [Orlarey 2004]. A curious reader should also read [Jouvelot 2011] for details on the multirate semantics because such a semantics is out of the scope of this dissertation.

Signal processors derived from the BDA are used to connect one signal processor to another. There are two types of signal processors: signal processors derived from the BDA and *Faust primitives*. There are five BDA operators for the composition of signal processors: sequential, parallel, split, merge and recursive. Signals and signals processors available in Faust are summarized in Table 4.7. Afterwards, we describe, in Figure 4.6, the intuition of how BDA operators work; however, recursive composition requires more explanations.

A recursive composition $A \sim B$, where $A : \mathbb{S}^n \to \mathbb{S}^m$ and $B : \mathbb{S}^p \to \mathbb{S}^q$, means that the first p output signals of A are the p input signals of B, but they are delayed one sample, and the first q input signals of A are the q output signals of B. In the first sample of execution, the first q input signals of A are initialized by 0.

Faust primitives also include most of C/C++ operators; for instance, arithmetic, comparison and bitwise primitives. There are also GUI primitives; for instance, buttons, checkboxes, horizontal sliders and integer entries.

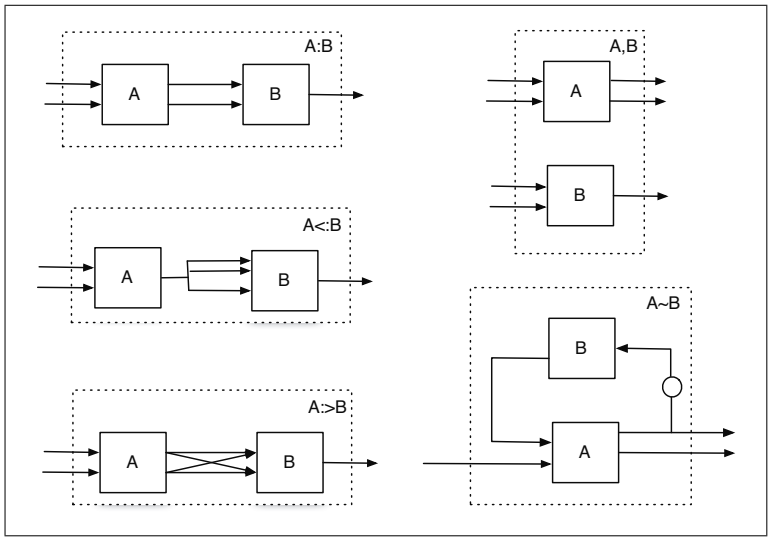

Figure 4.6: Block-diagram algebra operators. We show some basic cases, there are other ways to combine signals processors with different arities described in [Orlarey 2004].

There are some other Faust primitives slightly different from C/C++ operators. Con-

ditional branching is obtained using the selector primitives; for instance, the `select2` receives two streams and an index signal, which can have a value of 0 or 1, to choose the output stream. Finally, there are read-only and read-write tables (buffers); for instance, the read-only table which has three inputs: a constant-size signal to choose the table size, an initialization signal with the values to be stored and an index signal determining the position of such values. The read-only table produces an output signal by reading the content of the table on the position given by the index signal. The read-write table is similar, but the data stored at initialization time can be modified.

Signals	**Meaning**
Constant signals	Always same value
Integer signals	Always same integer
Tuples of signals	n-tuples of signals
Basic Signal Processors (SPs)	**Meaning**
Identity box	Identity function
Cut box	one input, no output
SPs composition	**Symbol**
Sequential	:
Parallel	;
Split	$<:$
Merge	$:>$
Recursive	$\sim$
Faust primitives	**Examples**
Arithmetic	$+,-,\times,/$
Comparison	$<,\leq,>,\geq,=$
Bitwise	$<<,>>,\&,\vert$
Constants	$1,2,3,4.5$
Read Tables	read only buffer
Read and write table	read&write buffer
Selectors	conditional branching
Graphical interface	buttons and checkboxes

Table 4.7: Intuitive semantics of Faust.

Example 4.3.1. Faust programs that can be controlled during execution from GUI controls; for instance, to stop and restart its execution. Figure 4.7 is the block diagram of a program that has a counter that increases the amplitude of the output of an oscillator in each iteration, and the counter can be reseted by the user, using a checkbox GUI control. The oscillator is defined in `music.lib`, a library distributed with Faust.

```
import("music.lib");
process = osci(50)*(+(0.0001)~(select2(==(checkbox("Reset"),0),_,1)));
```

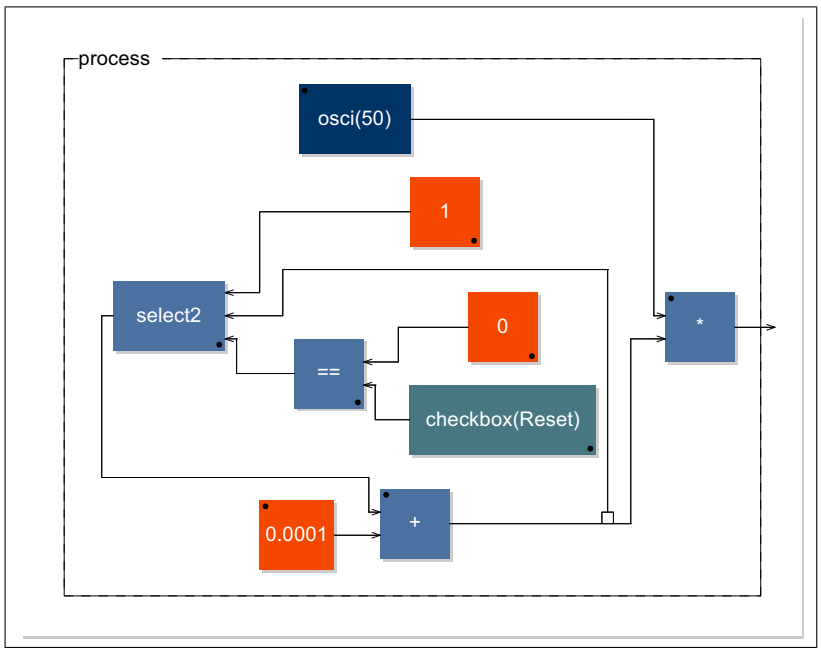

Figure 4.7: A simple Faust program. A counter that can be restarted by the user.

Part II

Models of Interactive Scores

CHAPTER 5

Nonhierarchic Interactive Scores

Contents

In interactive scores, it is possible to specify a variety of relations among temporal objects such as temporal relations, hierarchical relations, harmonic relations, rhythmical constraints and conditional branching. Nonetheless, in this chapter, we only take into account relations limited to point-to-point temporal relations without disjunction nor inequality ($\neq$) and quantitative temporal relations. We combine qualitative and quantitative temporal relations on the lines of previous independent works by Meiri and Gennary [Meiri 1996, Gennari 1998], as explained in Chapter 3.

In what follows, we introduce a mathematic definition of the structure of interactive scores, a formal semantics based on timed event structures, the temporal constraints of a score, and some formal properties such as playability. We also discuss the complexity of the playability problem.

5.1 Structural Definition of the Score

Interactive scores are composed by temporal objects and temporal relations. We consider that all temporal objects have only a start and end point and it is not possible to define intermediate points.

5.1.1 Temporal objects

A temporal object has two *point identifiers*: to control its starting and ending times. An external action is usually associated to each of them (e.g., turn on the lights, play a video or stop a sound). Some temporal objects are interactive, thus we call them *interactive objects*.

Definition 5.1.1 (Temporal object (TO))**.** *Let P be a set of point identifiers. A* Temporal object *is a tuple $o = \langle sp, ep, \Delta \rangle$, where $sp, ep \in P, sp \neq ep$, are called* start *and* end *points, respectively, and $\Delta \subseteq \mathbb{N}$ is a set of* durations*. A temporal object whose duration $\Delta = \{0\}$ is called an* interactive object*. Functions* $\mathrm{sp}(o)$*,* $\mathrm{ep}(o)$ *and* $\mathrm{d}(o)$ *return the start, end and duration, respectively, of object o. The set of all temporal objects is $\mathscr{T}$.*

5.1.2 Temporal relations

Points $p, q \in P$ are supposed to be positioned on a timeline. Temporal positions of points could be fully or partially determined. Temporal relations constrain the set of possibilities for these positions. A partial order among points is given by *quantitative relations*; for instance, point q is executed between four and ten time units after point p. *Qualitative temporal relations* can be easily expressed as quantitative relations; for instance, point-to-point *before* relation is the interval $(0, \infty)$ and point-to-point *equal* relation is the set $\{0\}$, a proposed in [Meiri 1996].

Our quantitative relations are close in spirit to the temporal relations described by Allombert *et al.* which contain time intervals [Allombert 2008d]. A limitation of Allombert's interactive scores is that all intervals must be *flexible*: intervals must have the form $(0, \infty)$, $[0, \infty)$ or $\{0\}$. In Allombert's thesis [Allombert 2009], the model is extended to general integer intervals, but arbitrary durations cannot be expressed. The durations contained in our temporal relations are usually intervals, but they can be any set of integers.

Definition 5.1.2 (Temporal Relation)**.** *Let function $\nu : P \to \mathbb{P}(\mathbb{N})$ give the set of* potential time positions *for each point $p \in P$. A temporal relation is a tuple $\langle p, \Delta, q \rangle$ where $\Delta \subseteq \mathbb{N}$ is the set of durations between points $p, q \in P$. We use the notation $T + \Delta =^{def} \{t' | t' = t + \delta, t \in T, \delta \in \Delta\}$ for temporal constraints of duration. Temporal positions of p and q are said to be constrained by $\nu(q) = \nu(p) + \Delta$. The set of all temporal relations is $\mathscr{R}$.*

We recall from Chapter 3 that *Allen's relations* [Allen 1983] without disjunction, over discrete time, can be easily expressed as point-to-point relations [Meiri 1996]. Furthermore, with point-to-point relations we can express relations that cannot be expressed in Allen's relations without disjunction; for instance, that the end of a temporal objects is before the end of another temporal object.

Example 5.1.3. Figure 5.1 shows how the Allen's relation "red light overlaps green light" can be represented by three point-to-point *before* relations.

5.1.3 Interactive scores

Definition 5.1.4 (Interactive Score)**.** *An* interactive score *is a set of temporal objects equipped with a set of temporal relations: a tuple $\langle P, O, R \rangle$, where P is a set of point*

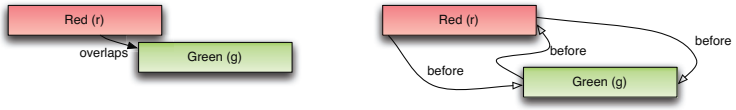

Figure 5.1: Encoding of the Allen relation *overlaps* into point-to-point relations.

identifiers, $O \subseteq \mathscr{T}$ is a set of temporal objects, $R \subseteq (P \times \mathbb{P}(\mathbb{N}) \times P)$ the temporal relations. Set R also includes the relations derived from the duration of temporal objects. For each $o \in O$, $\langle \mathrm{sp}(o), \mathrm{d}(o), \mathrm{ep}(o) \rangle \in R$. In addition, a relation $\langle p, \Delta, q \rangle \in R$ iff

1. *p, q are distinct points and $\nu(q) = \nu(p) + \Delta$;*
2. *two interactive objects do not occur at the same time; and*
3. *there is only one temporal relation between the start and end point of a temporal object.*

Property 2 takes care of the fact that two interactive points cannot happen at the same time; it means, that they cannot be related with zero-duration temporal relations, not even transitively by the means of other objects. The reason for this constraint is that interactive objects are usually launched by the user of the scenario; therefore, we cannot guarantee that the user will launch them at the same time. This simplifies the model.

Example 5.1.5. Figure 5.2 is an example of a score. Objects *red light, green light* and *sound* produce visible actions at their start and end. Objects a, b are interactive. Temporal relations *starts* represents a zero-duration between the start points of the two objects they connect. Relations *ends* represents a zero-duration between the end points of the two objects they connect. Allen's relation *overlaps* can be represented by the three point-to-point relations, as shown in Figure 5.1.

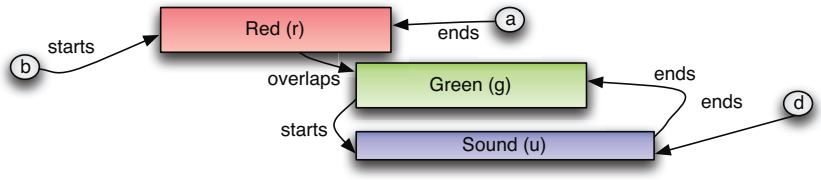

Figure 5.2: Example of an interactive score.

Let $\mathscr{S}$ be the set of all interactive scores and $s = \langle P, O, R \rangle$ a score in $\mathscr{S}$, function

$$\mathrm{ts}(s) = \bigwedge_{(p,\Delta,q) \in R} \nu(q) \in \nu(p) + \Delta, \textit{ where } \nu(p), \nu(q) \in \mathbb{N}_F \textit{ and } \mathbb{N}_F \textit{ is a finite subset of } \mathbb{N} \tag{5.1}$$

gives the conjunction of the temporal constraints of the potential time positions of the points of the score.

Constraints of duration	Explicit temporal relations
$v(\text{ep}(r)) \in v(\text{sp}(r)) + \text{d}(r)$	$v(\text{sp}(g)) = v(\text{sp}(u))$
$v(\text{ep}(g)) \in v(\text{sp}(g)) + \text{d}(g)$	$v(\text{ep}(a)) = v(\text{ep}(r))$
$v(\text{ep}(a)) \in v(\text{sp}(a)) + \{0\}$	$v(\text{sp}(g)) > v(\text{sp}(r))$
$v(\text{ep}(b)) \in v(\text{sp}(b)) + \{0\}$	$v(\text{ep}(g)) > v(\text{ep}(r))$
$v(\text{ep}(d)) \in v(\text{sp}(d)) + \{0\}$	$v(\text{sp}(g)) < v(\text{ep}(r))$
$v(\text{ep}(u)) \in v(\text{sp}(u)) + \text{d}(u)$	$v(\text{sp}(d)) = v(\text{ep}(u))$
	$v(\text{sp}(b)) = v(\text{sp}(r))$

Table 5.1: Implicit and explicit temporal constraints of the score in Figure 5.2. Relations "$<$" and "$>$" are represented by the interval $(0,\infty)$; relation "$=$" is represented by the set $\{0\}$.

Example 5.1.6. The constraints of the score in Figure 5.2 are presented in Table 5.1.

5.2 Event Structures Semantics

We recall that interactive scores must have formal semantics, as required for automated verification of properties of the scenario that are fundamental to its designers and users. We also recall that we denote by the functions $\text{E}(\varepsilon)$,$\text{l}(\varepsilon)$,and $\text{R}(\varepsilon)$ each component of an event structure ε.

5.2.1 Temporal objects

The events represent the start or end points of a temporal object. An interactive object is represented by a single event. Temporal relations are modeled with event delays. A static temporal object a is represented by two events s_a, e_a (start and end events). The labels of events are pairs $(type, o)$, where $type \in \{startPoint, endPoint, interactiveObject\}$ and o is the temporal object giving rise to the event.

Example 5.2.1. Figure 5.3 shows the encoding of three temporal objects.

Definition 5.2.2 (Temporal object encoding)**.** *The encoding of a temporal object (a) is a function* eto : $\mathscr{T} \to \mathscr{E}$ *defined by*

1. *if* $a = \langle sp, ep, \{0\}\rangle$ *(i.e.,* a *is an interactive object),*
   then $\text{eto}(a) = \langle \{s_a\}, \{(s_a, (interactiveObject, i))\}, \emptyset, \rangle$

2. *if* $a = \langle sp, ep, \Delta\rangle$ *(i.e.,* a *is a static temporal object), then* $\text{eto}(a) = \langle E, l, \emptyset, \rangle$,
   where $E = \{s_a, e_a\}$ *and* $l = \{(s_a, (startPoint, a)), (e_a, (endPoint, a))\}$.

The above definition guarantees that there are unique start and end events in the translation of a static temporal object, thus we know that each event is related to a single point.

Definition 5.2.3 (Relation between points and events)**.** *Let o be a temporal and P_o the set of points contained in o, function* pe : $(\mathscr{T} \times P) \to \mathrm{E}$ *associates a point identifier $p \in P_o$ to its corresponding event in* eto(o).

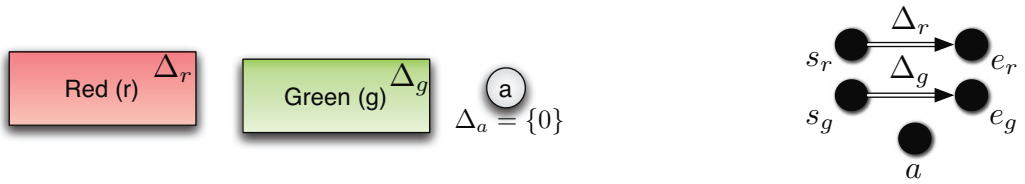

Figure 5.3: Encoding of a temporal object and its temporal relations of duration. There are two for r, two for g, and a single one for a. Double line arrows are just a visual notation for the event delays that model the duration of the temporal objects.

5.2.2 Temporal relations

Each point-to-point relation is represented by an event delay function.

Definition 5.2.4 (Temporal relation encoding)**.** *Let* p *be a point of temporal object* a *and* q *be a point of temporal object* b*. The encoding of a temporal relation r is given by the function* etr : $\mathscr{R} \to (E \times E \to \mathbb{P}(\mathbb{N} \cup \{\infty\}))$*. For each* $r = \langle p, \Delta, q \rangle \in \mathscr{R}$*, the encoding* etr$(r)$ *is defined by* pe$(a,p) \mapsto^{\Delta}$ pe(b,q).

Example 5.2.5. Figure 5.4 is the encoding of an *overlaps* relation between the objects r and g.

Figure 5.4: Encoding of two temporal objects, and the *overlaps* relation between them.

5.2.3 Interactive scores

The encoding of a score is given by adding the event delays from the encoding of the temporal relations to the encoding the temporal objects.

Example 5.2.6. The encoding of Figure 5.2 is presented in Figure 5.5.

Definition 5.2.7 (Interactive score encoding)**.** *The encoding of an interactive score* $s = \langle P, O, R \rangle$ *is given by the function* es : $\mathscr{S} \to \mathscr{E}$ *that translates interactive scores into event structures. Let* $o \in O$, eto$(o) = \langle E_o, l_o, \emptyset \rangle$*, then* es$(s) = \langle \bigcup_{o \in O} E_o, \bigcup_{o \in O} l_o, \bigcup_{r \in R} \mathrm{etr}(r), \rangle$.

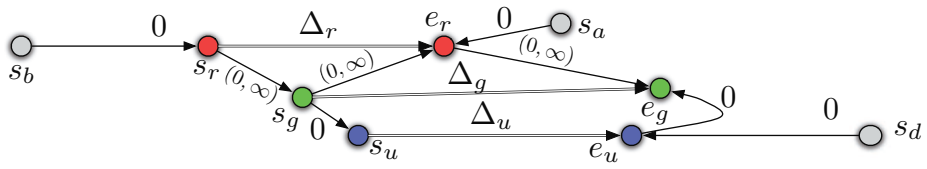

Figure 5.5: Encoding of the score in Figure 5.2.

We shall prove that the temporal constraint of the event structures semantics of a score corresponds to the temporal constraint of the score.

Definition 5.2.8 (Temporal constraint of an event structure)**.** *Let $\varepsilon = \langle E, R, l\rangle$ be an event structure without conflicts. The temporal constraint of an event structure* $\mathrm{tc}(\varepsilon)$ *is the conjunction of constraints $x_j \in x_i + \Delta$ for each $e_i \overset{\Delta}{\mapsto} e_j \in R$ with $x_i, x_j \in \mathbb{N}_F$, where $\mathbb{N}_F$ is a finite set of natural numbers.*

Given an event structure ε, $(e_1,t_1)...(e_n,t_n)$ is a valid trace of ε iff $(x_1,t_1)\dots(x_n,t_n)$ is a solution to $\mathrm{tc}(\varepsilon)$. The proof proceeds as follows. By the definition of event structures without conflicts 4.1.1, for all $0 < i, j \leq n$: $e_j \mapsto^{\Delta} e_i \Rightarrow t_i \in t_j + \Delta$ in any trace of ε because ε has no conflicts. By Def. 5.2.8, for each $e_i \mapsto^{\Delta} e_j \in \mathrm{R}(\varepsilon)$, we have the constraint $x_j \in x_i + \Delta$. Therefore, $(x_1,t_1)\ ...(x_n,t_n)$ is a solution to $\mathrm{tc}(\varepsilon)$.

Proposition 5.2.9 (Equivalence of interactive score constraints and its event traces)**.** *Let $s = \langle P, O, R\rangle$ be an interactive score, $\varepsilon = \mathrm{es}(s)$ the encoding of the score,* $\mathrm{ts}(s)$ *the temporal constraint of the score, and* $\mathrm{tc}(\varepsilon)$ *the temporal constraint of ε. It holds that* $\mathrm{ts}(s)[\forall_{p\in P}.p/\mathrm{pe}(c,p)] \Leftrightarrow \mathrm{tc}(\varepsilon)$*, where* $\mathrm{ts}(s)[\forall_{p\in P}.p/\mathrm{pe}(c,p)]$ *is obtained by replacing each point identifier by its corresponding event in the constraint* $\mathrm{ts}(s)$*, and p is the start or end point of temporal object $c \in O$.*

We recall that $\nu : P \to \mathbb{P}(\mathbb{N})$ gives the set of *potential time positions* for each point $p \in P$. We also recall the notation for temporal constraints: $t + \Delta = \{t' | t' = t + \delta, \delta \in \Delta\}$.

Proof. Let $s = \langle P, O, R\rangle$ be a score, $\langle p, \Delta, q\rangle \in R$ a temporal relation, $p \in P$ is a point of object $a \in O$, $q \in P$ is a point of object $b \in O$. By Def. 5.1.4, we have $\nu(q) = \nu(p) + \Delta$ for each temporal relation; by Def. 5.2.4, $\mathrm{etr}(\langle p, \Delta, q\rangle) = e_p \mapsto {}^{\Delta}e_q$; and by Def. 5.2.8, there is a constraint $x_{e_q} \in x_{e_p} + \Delta$ in $\mathrm{tc}(\varepsilon)$. It is trivial to show that the constraint $\mathrm{tc}(\varepsilon)$ obtained from Def. 5.2.8 is equivalent to the constraint $\mathrm{ts}(s)$ obtained from Def. 5.1.4: Replace the point identifiers by the time variables of the events using $\mathrm{pe}(p,a)$ and $\mathrm{pe}(q,b)$ introduced in Def. 5.2.3.

□

The proof above is presented for hierarchical interactive scores in [Toro 2012b].

5.3 Some Properties of the Scenarios

We insist that a motivation of defining an abstract semantics in event structures is to prove properties of the system execution; in particular, properties about the execution traces. As an example, to verify that temporal objects will be played as expected during performance (i.e., *playability*) or, in general, some property of each execution trace. We argued in Chapter 1 that such properties cannot be verified in applications based upon informal specifications, as it is the case for most existing software for multimedia scenarios with interactive controls. The following properties were already presented in [Toro 2012b].

- **Properties of the traces of execution.**
  - There exist a trace σ that contains a word w; for instance, the sequence of notes C-D-E is part of n traces of execution.
  - There exists n traces σ that contain a word w, possibly with other events in between; for instance, the sequence of notes C-D-E is contained in the trace $\sigma = (e_C,0),(e_C,1),(e_D,2),(e_C,3),(e_F,4),(e_E,6)$.
  - The number of possible traces of execution for a score ε is $\text{card}(\text{Traces}(\varepsilon))$.
  - If event e is launched before time unit n, the duration of object a is greater than m. For all $\sigma \in \text{Traces}(\varepsilon)$ and $(e,n),(s_a,t_i),(e_a,t_j) \in \sigma$, it holds that $t_j - t_i \geq m$.
  - After event e is played, there are n traces where event f is launched before event g.
  - Between time units a to b, there is no more than n objects playing simultaneously.
- **Minimum duration of a score.** Let s be a score and $\varepsilon = \text{es}(s)$ the encoding of s, the trace whose duration is minimum corresponds to a path from the start event of ε to the end event of ε such that the sum of the delays in the event delay relation is minimal among all paths connecting start and end.
- **Maximum and minimum number of simultaneous temporal objects.** Let $\sigma = (e_1,t_1)\ ...(e_n,t_n)$ be a trace of $\varepsilon = \text{es}(s)$, and $\text{maxS}(\sigma), \text{minS}(\sigma)$ the maximum and minimum number of events executed simultaneously in σ, respectively. The maximum and minimum number of simultaneous temporal objects of a score correspond, respectively, to the maximum and minimum value of $\text{maxS}(\sigma)$ and $\text{minS}(\sigma)$ among all $\sigma \in \text{Traces}(\varepsilon)$. We recall from Chapter 1 that this property is useful, for instance, to argue that there is only one curtain moving at the time during a theater performance.
- **Playability of a score.** This property states that all temporal objects will be *played* during execution; this is desirable because a score can be over-constrained and therefore not playable. Formally, let $PE_\sigma = \{e | (e,t) \in \sigma\}$ be the events played in trace σ. We say that a score is *playable iff* for all $\sigma \in \text{Traces}(\text{es}(s))$ it holds that $PE_\sigma = \text{E}(\text{es}(s))$.

The playability of a score can be decided by solving a *constraint satisfaction problem (*CSP*)*. There exists a $\sigma \in \text{Traces}(\text{es}(s))$ such that $PE_\sigma = \text{E}(\text{es}(s))$ *iff* the following CSP has at least one solution: a variable x_i for each event $e_i \in E$; the domain $\mathbb{N}_\mathbb{F} \cup \{\infty\}$ for each variable, where $\mathbb{N}_\mathbb{F}$ is a finite subset of $\mathbb{N}$; and the single constraint $\text{tc}(\varepsilon)$. This holds as a direct consequence of Prop. 5.2.9.

5.3.1 Time complexity of the playability of a score

In what follows we will show that deciding the playability of a score is NP-complete in the general case, but there is an interesting subclass that is tractable.

5.3.1.1 The NP-complete case

We will show that the decision problem of the *subset sum* [Martello 1990] can be encoded as the playability of an interactive score. The subset decision problem is stated as follows: Given a set of integers $\{w_1, w_2 ... w_n\}$ of n objects and an integer W, does any non-empty subset sum to W?

$$\sum_{i=1}^{n} w_i . x_i = W, x_i \in \{0,1\}, \text{ where at least one } w_i \neq 0. \tag{5.2}$$

There are several algorithms to solve the subset sum, all with exponential time complexity in n, the number of objects. The most naïve algorithm would be to cycle through all subsets of $1 \leq k \leq n$ numbers and, for every one of them, check if the subset sums to the right number. The running time is of order $\text{O}(n2^n)$, since there are 2^n subsets and, to check each subset, we need to sum at most n elements. The best algorithm known runs in time $\text{O}(2^{n/2})$, according to Martello [Martello 1990]. In what follows we show that the playability of a score is a NP-complete problem by following the methodology described in [Sipser 1996].

Proposition 5.3.1 (The playability of a score is a NP-complete problem)**.** *(1) The subset sum decision problem can be encoded as the playability of an interactive score. (2) If the score is not playable, there is not a subset whose sum is W. (3) If the score is playable, then it exists at least a subset whose sum is W. (4) To check whether a solution satisfies the playability problem can be done in polynomial time.*

Proof. In what follows we prove each postulate.

1. As shown in Figure 5.6[1], the subset problem can be encoded into an interactive score whose temporal constraint is $\text{ts}(s)$ The reader can verify that these constraints are equivalent to the constraint in Equation 5.2.

2. If the score is not playable, it means the conjunction of temporal constraints is unsatisfiable; therefore, there is no subset whose sum is W.

[1] Knapsack picture is taken from the Wikipedia. http://en.wikipedia.org/wiki/Knapsack_problem

3. If the score is playable, it means that there is at least one solution to the temporal constraint of the score; therefore, since the temporal objects are associated to the set of integers in the subset sum problem, we can construct a subset whose sum is W.

4. There is a polynomial algorithm to check whether a solution satisfies the problem. Such algorithm replaces the values of $x_i, 1 \leq i \leq n$ in Equation 5.2.

In conclusion, the subset sum can be encoded as the playability (i.e., satisfiability) of an interactive score and there is a polynomial-time algorithm to check if the solution is satisfied, thus the satisfiability of an interactive score is NP-complete.

□

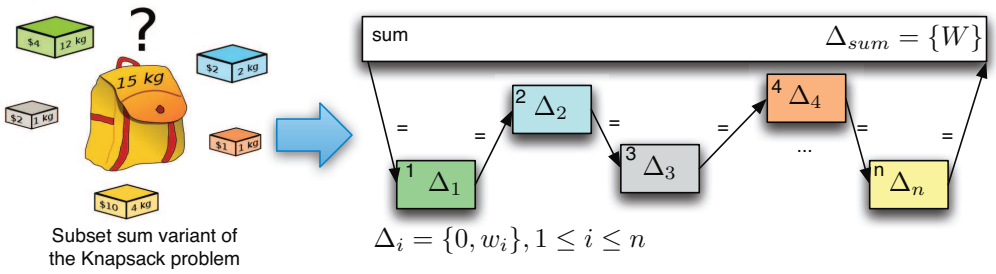

Figure 5.6: Encoding of the subset sum problem into an interactive score. Note that the subset sum problem is a variant of the knapsack decision problem where costs are not taken into account and the goal is to find if there is a subset of the elements that fills exactly the knapsack capacity.

5.3.1.2 A polynomial-time subclass

The conjunction of temporal constraints of an interactive score can be represented as a *simple temporal problem* (STP) when the domains of the durations are intervals of integers without holes [Dechter 1991]. The translation of the playability of a score into a STP consists in a set of point variables $P = \{p_1, ...p_n\}$, one for each point in the score, and a set of binary constraints over those variables $C = \{c_1, ..., c_n\}$, one for each temporal constraint of the score. Each constraint has the form $c_k : p_i - p_j \in [a,b]$ with $p_i, p_j \in P$ and $a,b \in \mathbb{N} \cup \infty$. Constraints of the form $p_i - p_j \in [a,b]$ can be easily obtained from the temporal constraint of an interactive score, defined in Equation 5.1. As an example, constraint of the form $p_i = p_j + [a,b]$ can be translated into two constraint a constraint $p_i - p_j \in [a,b]$. It is left to the reader the encoding of the inequalities into constraints of the form $p_i - p_j \in [a,b]$.

The satisfiability of a STP can be easily computed with an algorithm to find *all-pairs shortest-path* of a graph, such as *Floyd-Warshall* [Cormen 2001] algorithm which has a

polynomial time and space complexity. In fact, Floyd-Warshall has a time complexity of $O(n^3)$, where n is the number of points of the score. There are faster algorithms for this problem in the literature [Planken 2010, Xu 2003]; however, they are efficient to calculate if a STP has a solution, but do not guarantee that the constraint problem remains satisfiable when dispatching the events during the execution of a score.

Fortunately, with some transformations, a STP can be dispatched online efficiently by relying only on local propagation: looking only to the neighbors of the event launched, as proposed by Muscettola *et al.* [Muscettola 1998]. In Chapter 6, we extend the approach of Muscettola *et al.* to event structures: Transform an event structure in such a way that the events of the event structure can be dispatched online efficiently.

5.4 Summary and Discussion

In this chapter we presented a simple model of interactive scores with temporal objects without hierarchy, equipped with point-to-point temporal relations.

First, we described the structure of the score. Temporal objects contain start and end points, and temporal relations define a partial order for the temporal positions of such points, using function $\nu(p)$. In addition, function $\mathrm{ts}(s)$ defines the conjunction of the temporal constraints of the score.

Second, we defined the event structures semantics of a score $\mathrm{es}(s)$. Static temporal objects are represented by two events and interactive objects by a single event, events are labeled with both a type and the object that gives raise to the event, temporal relations are represented by an event delay function, and $\mathrm{tc}(\mathrm{es}(s))$ gives the temporal constraint of an event structure. We proved that the constraint $\mathrm{tc}(\mathrm{es}(s))$ is equivalent to the constraint $\mathrm{ts}(s)$.

Finally, we defined several properties of the score using its event structures semantics. We also show that the time complexity of the playability of a score is NP-complete, but the problem can be solved in polynomial time when the durations are intervals.

Although the definitions, propositions and properties presented in this chapter are very simple, they are the base for those in the following chapters. We recommend the reader to fully understand all the definitions of this chapter before reading the following chapters.

CHAPTER 6

Hierarchic Interactive Scores

Contents

Up to our knowledge, as we discussed in Chapter 3, constraints were not used in interactive multimedia contexts until the beginning of the century when independent works by Beurivé and Desainte-Catherine, and Rueda and Valencia took place. On the one hand, Beurivé and Desainte-Catherine proposed the first model of interactive scores to allow composers represent music scores as a hierarchy of temporal objects that contain local variables, with constraints on the duration of the objects and their local variables [Beurivé 2001]. They compared absolute and relative-time models, and they found out that the most appropriate for representing musical hierarchies is the relative-time model. On the other hand, Rueda and Valencia proposed a simple music improvisation model using the *nondeterministic timed concurrent constraint (ntcc)* calculus, and proved several properties of the model [Rueda 2001].

In this chapter, we extend the model of interactive scores introduced in Chapter 5 with a hierarchy on the lines of [Beurivé 2001]. We also define operational semantics based on ntcc on the lines of [Allombert 2006]. We impose a limitation to our model: We consider that all durations in the score are intervals. The reason is that we want to be able execute a score in real-time. The temporal constraints of a score can be represented as a *simple temporal problem* (STP) when durations are intervals, as shown in Chapter 5. In addition, the problem of dispatching a STP can be done efficiently online, as stated in Chapter 5; on

the contrary, dispatching a temporal problem when the durations can be any set of integers is equivalent to dispatch a *disjunctive temporal problem* (DTP).

A DTP consists on a set of point variables $P = \{p_1, ..., p_n\}$ and binary constraints over those variables $C = \{c_1, ..., c_n\}$ of the form $c_1 \vee c_2 ... \vee c_n$ with $c_k : p_i - p_j \in [a,b]$; $p_i, p_j \in P$ and $a,b \in \mathbb{N} \cup \infty$. There are algorithms to dispatch online a DTP, for instance, Tsamardinos *et al.*'s strategy [Tsamardinos 2001]; however, such algorithms are not mean for real-time interaction because they require backtracking.

In what follows, we introduce a mathematic definition of the structure of hierarchic interactive scores, a formal semantics based on event structures and an operational semantics based on ntcc.

6.1 Structural Definition of the Score

Allombert *et al.* proposed a structural definition in which a temporal object is a tuple that contains several attributes and a set of temporal objects [Allombert 2007]. The recursive definitions proposed by Allombert *et al.* pose difficulties as a formal semantics of interactive scores; for instance, awkward conditions would have to be added to those definitions just to specify the hierarchy as a directed tree. In this section, hierarchical relations are explicitly represented as a directed tree to avoid such a problem.

6.1.1 Temporal objects

A temporal object may contain other temporal objects: this hierarchy allows us to control the start or end of a temporal object by controlling the start or end of its parent.

Another key concept in interactive music scores is the nominal duration of a temporal object: a value preferred by the composer, which may change due to the interactive objects. A model of interactive scores based on Petri nets [Allombert 2007] includes a *nominal duration* and a *nominal start-time* for the temporal objects. In the following definition of temporal object, we include a set of possible durations –represented as an integer interval–, a nominal duration, and a nominal starting time. We consider the nominal starting time relative to its parent starting time, except for the root, whose nominal starting time is zero.

Definition 6.1.1 (Temporal object (TO))**.** *Let P be a set of point-ids. A* Temporal object *$\langle I, H \rangle$ is a directed tree with nodes I and arcs $H \subseteq I \times I$. Elements of I, the* intervals*, are tuples $\langle sp, ep, \Delta, snt, dnt \rangle$, where $sp, ep \in P, sp \neq ep$, are called* start *and* end *points, respectively, $\Delta \subseteq \mathbb{N}$ is a set of* durations*, and $snt \in \mathbb{N}, nd \in \Delta$ are the* nominal starting time *and the* nominal duration*, respectively. All points sp (resp. ep) of the intervals of a TO are pairwise distinct. Arcs of a TO represent hierarchical relations: $(i_1, i_2) \in H$ means that interval i_1 contains interval i_2 (and thus the TO defined by the subtree rooted by i_2). For $i \in I$, functions* $\mathrm{sp}(i)$, $\mathrm{ep}(i)$, $\mathrm{d}(i)$, $\mathrm{stn}(i)$ *and* $\mathrm{dtn}(i)$ *return the start, end and duration, nominal start-time and nominal duration, respectively, of interval i.*

As in Def. 5.1.1, a temporal object whose interval has $\Delta = \{0\}$ and has no subtrees is called an *interactive object*, and the set of all temporal objects is $\mathscr{T}$.

6.1.2 Temporal relations

Temporal relations are defined as in Def. 5.1.2, but the time distance between two points must be an integer interval. We recall that the function $\nu : P \to \mathbb{P}(\mathbb{N})$ gives the set of *potential time-positions* for each point $p \in P$.

6.1.3 Interactive scores

The definition of an interactive scores is equipped with a hierarchy; therefore, instead of a set of temporal objects, it contains a single temporal object, which represents the root of the hierarchy tree. We also include nominal starting times and nominal durations to preserve the intuition of traditional music scores. We recall from Equation 5.1 that $\mathrm{ts}(s) = \bigwedge_{(p,\Delta,q)\in R} \nu(q) \in \nu(p) + \Delta$, *where* $\nu(p), \nu(q) \in \mathbb{N}_F$ returns the conjunction of the temporal constraints of a score.

Definition 6.1.2 (Interactive Score)**.** *An* interactive score *is a temporal object equipped with a set of temporal relations: A tuple* $\langle P, o, R\rangle$*, where* P *is a set of point ids,* o *is a temporal object and* $R \subseteq (P \times \mathbb{P}(\mathbb{N}) \times P)$*.*

Set R also includes the relations derived from (1) the duration of temporal objects and (2) the nominal starting time and (3) the hierarchical structure. As an example, when i_2 *is contained in* i_1*, the start point of* i_2 *must not occur before the start point of* i_1*. We use the notation* $t + \Delta =^{def} \{t' | t' = t + \delta, \delta \in \Delta\}$ *for temporal constraints of duration.*

1. *For each* $i \in I$, $\langle \mathrm{sp}(i), \mathrm{d}(i), \mathrm{ep}(i)\rangle \in R$*, as proposed in Def. 5.1.4*

2. *For each* $p \in P$*, such that* p *is the start point of object* $a \neq o$*,* $\nu(\mathrm{sp}(\mathrm{parent}(a))) + \mathrm{stn}(a) \in \nu(p)$*. If* $a = o$ *(i.e., a is the root),* $\nu(\mathrm{sp}(o)) = \{0\}$.

3. *For each* $(i_1, i_2) \in H$, $\langle \mathrm{sp}(i_1), [0,\infty), \mathrm{sp}(i_2)\rangle \in R$ *and* $\langle \mathrm{ep}(i_2), [0,\infty), \mathrm{ep}(i_1)\rangle \in R$

4. *There exist at least a solution for the temporal constraints of the score* $\mathrm{ts}(s)$ *in which we can observe all the nominal starting times and nominal durations of the temporal objects of the score.*

Example 6.1.3. Figure 6.1 is an example of a score with hierarchy: an extension of the score in Figure 5.2. As an example, the structural definition of the score in Figure 6.1 is presented in Table 6.1, and the constraints of the score are presented in Table 6.2.

Example 6.1.4. As a music example, consider that the object *sound* presented in Figure 6.1 is the one described in Figure 6.2. There is a sequence of notes *G,G,A,G,B,C* accompanied by the chords *G major* and *C major*. The duration of the last note (*C*) is determined by an interactive object h that controls the start of the note and by the interactive object d that controls the end of object *sound* and, indirectly, the end of the note *C*. Depending on the user interactions, the last note can last between one and three time units.

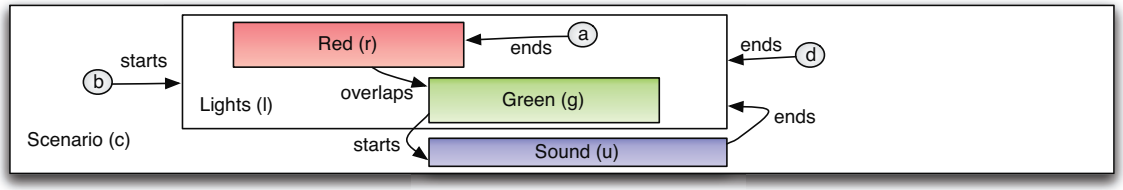

Figure 6.1: Example of a score with hierarchy.

Points and Temporal Objects	**Explicit Temporal Relations**
$I = \{c,l,r,g,u,a,b,d\}$	$R = \{$
$H = \{(c,l),(c,u),(c,b),(c,d),(l,r),(l,g),(l,a)\}$	$\langle ep_b,\{0\},sp_l\rangle$,
$P = \{sp_c,sp_l,sp_r,sp_g,sp_u,sp_a,sp_b,sp_d,$	$\langle sp_g,\{0\},sp_u\rangle$,
$\quad ep_c,ep_l,ep_r,ep_g,ep_u,ep_a,ep_b,ep_d\}$	$\langle ep_a,\{0\},ep_r\rangle$,
$c = \langle sp_c,ep_c,\Delta_c\rangle$	$\langle ep_u,\{0\},ep_l\rangle$,
$l = \langle sp_l,ep_l,\Delta_l\rangle$	$\langle ep_d,\{0\},ep_l\rangle$,
$r = \langle sp_r,ep_r,\Delta_r\rangle$	$\langle sp_r,(0,\infty),sp_g\rangle$,
$g = \langle sp_g,ep_g,\Delta_g\rangle$	$\langle ep_r,(0,\infty),ep_g\rangle$,
$u = \langle sp_u,ep_u,\Delta_u\rangle$	$\langle sp_g,(0,\infty),ep_r\rangle$
$a = \langle sp_a,ep_a,\{0\}\rangle$	$\}$
$b = \langle sp_b,ep_b,\{0\}\rangle$	
$d = \langle sp_d,ep_d,\{0\}\rangle$	

Table 6.1: Structural definition of the score in Figure 6.1. A temporal object is a directed tree $o = \langle I,H\rangle, H \subseteq I \times I$ and a score is a tuple $\langle P,o,R\rangle$.

Durations	**Hierarchy**	**Ex. Temporal relations**
$v(\text{ep}(c)) \in v(\text{sp}(c)) + \text{d}(c)$	$v(\text{sp}(l)) \geq v(\text{sp}(c)) \wedge v(\text{ep}(c)) \geq v(\text{ep}(l))$	$v(\text{ep}(b)) = v(\text{sp}(l))$
$v(\text{ep}(l)) \in v(\text{sp}(l)) + \text{d}(l)$	$v(\text{sp}(u)) \geq v(\text{sp}(c)) \wedge v(\text{ep}(c)) \geq v(\text{ep}(u))$	$v(\text{ep}(d)) = v(\text{ep}(l))$
$v(\text{ep}(r)) \in v(\text{sp}(r)) + \text{d}(e)$	$v(\text{sp}(b)) \geq v(\text{sp}(c)) \wedge v(\text{ep}(c)) \geq v(\text{ep}(b))$	$v(\text{sp}(g)) = v(\text{sp}(u))$
$v(\text{ep}(g)) \in v(\text{sp}(g)) + \text{d}(g)$	$v(\text{sp}(d)) \geq v(\text{sp}(c)) \wedge v(\text{ep}(c)) \geq v(\text{ep}(d))$	$v(\text{ep}(a)) = v(\text{ep}(r))$
$v(\text{ep}(u)) \in v(\text{sp}(u)) + \text{d}(u)$	$v(\text{sp}(r)) \geq v(\text{sp}(l)) \wedge v(\text{ep}(l)) \geq v(\text{ep}(r))$	$v(\text{ep}(u)) = v(\text{ep}(l))$
$v(\text{ep}(a)) \in v(\text{sp}(a)) + \{0\}$	$v(\text{sp}(a)) \geq v(\text{sp}(l)) \wedge v(\text{ep}(l)) \geq v(\text{ep}(a))$	$v(\text{sp}(g)) > v(\text{sp}(r))$
$v(\text{ep}(b)) \in v(\text{sp}(b)) + \{0\}$	$v(\text{sp}(g)) \geq v(\text{sp}(l)) \wedge v(\text{ep}(l)) \geq v(\text{ep}(g))$	$v(\text{ep}(g)) > v(\text{ep}(r))$
$v(\text{ep}(d)) \in v(\text{sp}(d)) + \{0\}$		$v(\text{sp}(g)) < v(\text{ep}(r))$

Table 6.2: Implicit and explicit temporal constraints of the score in Figure 6.1. Relation $=$ can be represented by $\{0\}$, relation $<$ and $>$ by $(0,\infty)$, and relations $\leq$ and $\geq$ by $[0,\infty)$.

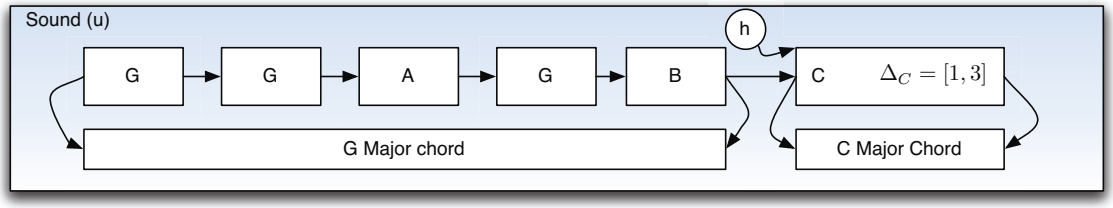

Figure 6.2: Example of the *sound* temporal object in Figure 6.1.

6.2 Event Structures Semantics

In this section, we define event structures semantics for interactive scores, extending the previous semantics presented in Chapter 5 with a hierarchy. Nominal starting times and nominal durations do not change the event structures semantics, they can only be observed in the operational semantics. It is left as future work an event structures semantics that allows to observe the nominal starting times and nominal durations.

We recall from Chapter 4 that we denote by the functions $\mathrm{E}(\varepsilon)$,$\mathrm{l}(\varepsilon)$,and $\mathrm{R}(\varepsilon)$ each component of an event structure ε.

6.2.1 Temporal objects

The events represent the start or end points of a temporal object, as it was presented in Chapter 5. An interactive object is represented by a single event. Temporal relations are modeled with event delays.

A *static temporal object a* is represented by two events s_a, e_a (start and end events) together with all events in the translation of the objects in the subtrees rooted by a. The labels of events are pairs $(type, i)$, where $type \in \{startPoint, endPoint, interactiveObject\}$ and i is the interval in the root of the temporal object giving rise to the event.

Example 6.2.1. As an example, Figure 6.3 shows the encoding of a temporal object.

Definition 6.2.2 (Temporal object encoding)**.** *The encoding of a temporal object (a) is a function* eto : $\mathscr{T} \to \mathscr{E}$. *Let i be the interval in the root of a, the encoding of a is defined by*

1. *if* $i = \langle sp, ep, \{0\}\rangle$ *and a has no subtrees (i.e,* a *is an interactive object), then* $\mathrm{eto}(a) = \langle \{s_a\}, \{(s_a, (interactiveObject, i))\}, \emptyset\rangle$

2. *if* $i = \langle sp, ep, \Delta\rangle$ *(i.e.,* a *is a static temporal object), then* $\mathrm{eto}(a) = \langle E, l, \emptyset\rangle$, *where*
$$E = \{s_a, e_a\} \cup \bigcup_{x \in \mathrm{subtrees}(a)} \mathrm{E}(\mathrm{eto}(x))$$
$$l = \{(s_a, (startPoint, i)), (e_a, (endPoint, i))\} \cup \bigcup_{x \in \mathrm{subtrees}(a)} \mathrm{l}(\mathrm{eto}(x))$$

The above definition also guarantees that there are unique start and end events in the translation of a static temporal object, as stated in Def. 5.2.2. Let P_o be the set of points of a temporal object o, we recall that function $\mathrm{pe} : P \times \mathscr{T} \to \mathrm{E}(\mathrm{eto}(o))$ associates a point identifier $p \in P_o$ to its corresponding event in $\mathrm{eto}(o)$.

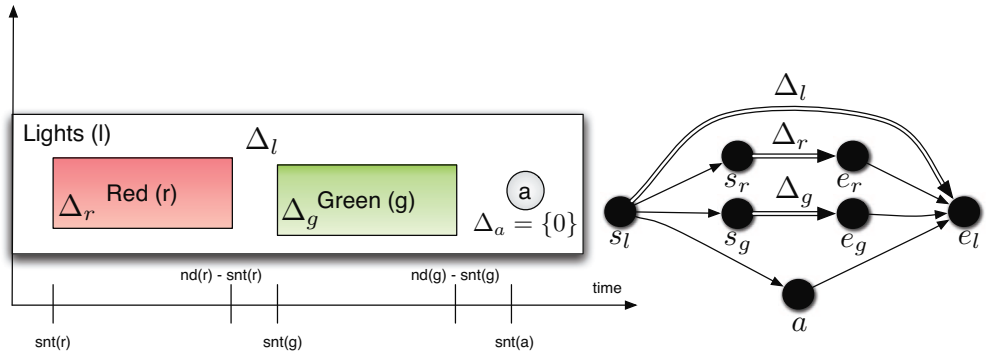

Figure 6.3: Encoding of a temporal object and its temporal relations of duration and hierarchy. There are two events for l, two for r, two for g, and a single one for a. Simple arrows starting from s_l and simple arrows arriving to e_l represent the event delays imposed by the hierarchy. Double line arrows are just a visual notation for the event delays that model the duration of the temporal objects.

6.2.2 Temporal relations

Temporal relations are encoded as in Def. 5.2.4: each temporal relation is encoded as an event delay between the corresponding events.

6.2.3 Interactive scores

Definition 6.2.3 (Interactive score encoding)**.** *The encoding of an interactive score* $s = \langle P, o, R\rangle$ *is given by the function* $\mathrm{es} : \mathscr{S} \to \mathscr{E}$ *that translates interactive scores into event structures. Let* $\mathrm{eto}(o) = \langle E, l, \emptyset\rangle$*, then* $\mathrm{es}(s) = \langle E, l, \bigcup_{r\in R}, \mathrm{etr}(r)\rangle$.

Example 6.2.4. As an example, the encoding of Figure 6.1 is presented as follows. The event delay relation is presented in three subsets: derived from the durations, from the hierarchy and from the explicit temporal relations. Figure 6.4 is a visual representation of the encoding. Let $\varepsilon = \langle E, l, R\rangle$ such that

- $E = \{ \quad s_c, e_c, s_l, e_l, s_r, e_r, s_g, e_g, s_u, e_u, s_a, s_b, s_d \}$

- $l = \{ \quad (s_c, (startPoint, c)), (e_c, (endPoint, c)), (s_l, (startPoint, l)),$
  $(s_r, (startPoint, r)), (e_r, (endPoint, r)), (s_g, (startPoint, g)),$
  $(e_g, (endPoint, g)), (s_u, (startPoint, u)), (e_u, (endPoint, u)),$
  $(s_a, (interactiveObject, a)), (s_b, (interactiveObject, b))$
  $(e_l, (endPoint, l)), , (s_d, (interactiveObject, d)) \}$

- $R = \{ \quad s_c \mapsto^{\Delta_c} e_c, s_l \mapsto^{\Delta_l} e_l, s_r \mapsto^{\Delta_r} e_r, s_g \mapsto^{\Delta_g} e_g, s_u \mapsto^{\Delta_u} e_u \}$
  $\cup \{ \quad s_c \mapsto s_b, s_c \mapsto s_u, s_l \mapsto s_r, s_l \mapsto s_g, e_l \mapsto e_c,$
  $e_u \mapsto e_c, e_d \mapsto e_c, e_g \mapsto e_l, e_r \mapsto e_l \}$
  $\cup \{ \quad s_r \mapsto^{(0,\infty)} s_g, s_g \mapsto^{(0,\infty)} e_r, e_r \mapsto^{(0,\infty)} e_g, s_b \mapsto^{\{0\}} s_l, s_d \mapsto^{\{0\}} e_l,$
  $s_a \mapsto^{\{0\}} e_r, e_u \mapsto^{\{0\}} e_l, s_g \mapsto^{\{0\}} s_u \}$

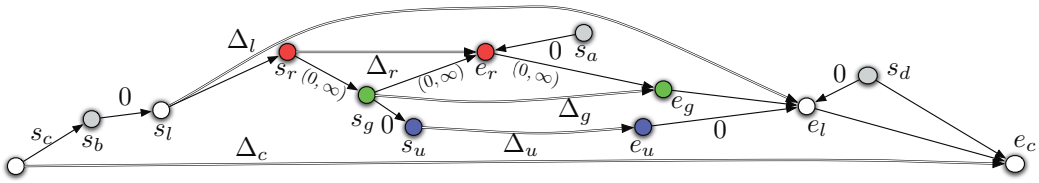

Figure 6.4: Encoding of the score in Figure 6.1. Some redundant event delays are removed for simplicity.

Correctness. We shall prove that the temporal constraints of the event structures semantics of a score corresponds with the temporal constraints of the score. This is a corollary of Proposition 5.2.9. The new hierarchical relations are added to the temporal relations in the structural definition of the score and they are translated to event delays in the event structures semantics; therefore, the proof proceeds the proof of Proposition 5.2.9.

6.3 Operational Semantics

Although event structures provide a theoretical background to specify properties and understand the system, there is no difference between interactive objects and static temporal objects in the event structures semantics: such a difference can only be observed in the operational semantics, as we argued in Chapter 1. In this section, we present an operational semantics based on the *dispatchable normal form* of the event structures of the score. A score is in *normal form* when it does not have zero-duration event delays. A normal form is *dispatchable online* if the events can be launched by relying only on local propagation. The computation of the normal form is similar to the algorithm to transform a score into a

Petri net proposed by Allombert *et al.* [Allombert 2007]. In Petri nets model of interactive scores, events that happen at the same time share the same *place* (i.e., state).

The ntcc model proposed in [Allombert 2006] is based on Allen's relations [Allen 1983]; fortunately, point-to-point relations can express to Allen's relations without disjunction [Meiri 1996], as we stated previously in Chapter 3.

In what follows we define the normal form of the event structures semantics of a score; then, we define the notion of label traces and we use it to prove that the normal form is equivalent to the event structures semantics of the score; then, we define the notion of dispatchable event structures; finally, we define the ntcc model of the score and we prove that it is correct with respect to the temporal constraints of the event structures semantics.

6.3.1 A normal form of the event structures semantics of a score

An event structure is in normal form if it has no zero-duration event delays. The normal form collapses simultaneous events into a single event. The normal form simplifies the definition of the operational semantics because we do not have to synchronize two processes to launch two events at the same time. If static and interactive events must happen at the same time and each one is represented as a separated concurrent process, synchronization is difficult because the continuation of a ntcc *unless* process must always happen in the next time unit.

Example 6.3.1. As an example, Figure 6.5 is the normal form of Figure 6.1.

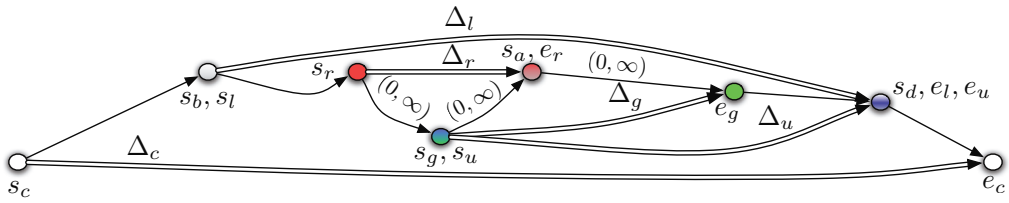

Figure 6.5: Normal form of the score in Figure 6.5. Single-line arrows represent delays whose duration is $[0,\infty]$ and double-line arrows represent the duration of temporal objects. Some redundant delays are removed for simplicity.

To calculate the normal form, we remove sequentially each event delay with zero duration. To remove a delay of the form $a \mapsto^{\{0\}} b$, we proceed as shown in Figure 6.6. To easily combine labels, we represent the labels of an event structure as sets using function $\mathrm{ls} : (E \to Act) \to (E \to \mathbb{P}(Act))$ defined as $\mathrm{ls}(l) = \{(e,\{label\}) | (e,label) \in l\}$, where Act is the set of labels of an event structure. Henceforth, we denote ε^* an event structure whose labels are sets.

Definition 6.3.2 (Normal Form of the Encoding of a Score)**.** *Let $\varepsilon = \langle E, l, R\rangle$ be the encoding of a score, $\varepsilon^* = \langle E, \mathrm{ls}(l), R\rangle$ and the relation $\overset{norm}{\Longrightarrow} \subseteq \mathscr{E} \times \mathscr{E}$ defined in Figure 6.6.*

The normal form is obtained by applying $\overset{norm}{\Longrightarrow}$ *until there are no zero-duration delays in the event structure*[1]*:* $\varepsilon^* \overset{norm^*}{\Longrightarrow} \text{normal}(\varepsilon^*) \not\Longrightarrow$.

Figure 6.6: Normalization rule ($\overset{norm}{\Longrightarrow}$). If there is a zero-duration event delay $a \mapsto^{\{0\}} b$, $\overset{norm}{\Longrightarrow}$ removes the zero-duration delay and the event a, it combines the labels of a and b, and connects a's predecessors and successors to b.

The normal form and the encoding of a score do not necessarily have the same event traces; however, they have the same *label traces*. Label traces are similar to time traces, but they relate each time point to the set of labels that appear in such time point.

6.3.1.1 Label traces

We use the notation $\varepsilon^* = \langle E, l^*, \mathscr{R} \rangle$ to represent an event structure whose labels are sets.

Definition 6.3.3 (Label Trace)**.** *Given an event structure* $\varepsilon^* = \langle E, l^*, R \rangle$ *and* $\sigma \in \text{Traces}(\varepsilon^*)$*, a label trace is a sequence* $\sigma_L = (L_1, t_1), \ldots, (L_n, t_n)$ *with* $L_i \in \{y | y \in \text{range}(\text{l}(\varepsilon))\}, t_i \in (\mathbb{N} \cup \{\infty\})$ *(all label sets being pairwise disjoint and all time points being pairwise different), such that for all* $1 \leq i \leq n$*, there is an element* $(e, t_i) \in \sigma$ *with* $\text{l}(e) \subseteq L_i$ *and* $(L_i, t_i) \in \sigma_L$*. We denote* $\text{l-Traces}(\varepsilon)$ *the set of all label traces of* ε*.*

We say that two event structures $\varepsilon^*, \varepsilon'^*$ have the same label traces *iff* for each $(L,t) \in \sigma, \sigma \in \text{l-Traces}(\varepsilon^*)$ and $(L', t') \in \sigma', \sigma' \in \text{l-Traces}(\varepsilon'^*)$, if $t = t'$ then $L = L'$.

Proposition 6.3.4. *An event structure* $\varepsilon = \langle E, l, R \rangle$ *and its normal form have the same label traces. Let* $\varepsilon^* = \langle E, \text{ls}(l), R \rangle$*. It holds that* $\text{l-Traces}(\varepsilon^*) = \text{l-Traces}(\text{normal}(\varepsilon^*))$.

Proof. We must prove that there is a finite sequence of derivations from ε^* to $\text{normal}(\varepsilon^*)$ applying the *norm* rule. Let $\varepsilon^* = \langle E, l^*, R \rangle$ and $\text{normal}(\varepsilon^*) = \langle E', l'^*, R' \rangle$. We will proceed by induction over the sequence $\varepsilon^* \overset{norm}{\Longrightarrow} \varepsilon'^* \overset{norm^*}{\Longrightarrow} \text{normal}(\varepsilon) \not\Longrightarrow$.

1. **Base case.** Let $f, g \in \mathscr{E}$ be the event structures in the left and right side of Figure 6.6, respectively. First we have to prove that $\text{l-Traces}(f) = \text{l-Traces}(g)$. Events a and b occur at the same time point in f, then labels L_a and L_b occur at the same time in the label traces of f. Event b occurs at the same time in g and f because b has the successors and predecessors of a in g, therefore, labels L_a and L_b occur at the same time point in the label traces of both f and g. Since L_a and L_b occurs at the same time in both f and g, for any $a \in \text{E}(\varepsilon^*), b \in \text{E}(\varepsilon^*)$ such that $a \mapsto^{\{0\}} b$, it holds that $\text{l-Traces}(\varepsilon^*) = \text{l-Traces}(\varepsilon'^*)$.

[1] Syntax $\overset{norm^*}{\Longrightarrow}$ means that the $\overset{norm}{\Longrightarrow}$ rule is applied zero or more times sequentially.

2. **Inductive case.** Induction over the sequence of derivations $\varepsilon^* \overset{norm}{\Longrightarrow} \varepsilon'^* \overset{norm^*}{\Longrightarrow} \text{normal}(\varepsilon^*) \not\Longrightarrow$. The argument follows as for the base case because there is only one rule. The sequence of derivations is finite because the normalization rule always reduces the number of events and event delays.

□

6.3.2 Dispatchable event structures

The normal form does not guarantee that the consistency of the constraints will be maintained during an execution with an algorithm that relies only on local propagation without the need of backtracking. The reader may see this problem in Figure 6.7. Local propagation is critical because neither backtracking nor global propagation are appropriate for real-time interaction, thus the encoding of the score must be transformed into a *dispatchable event structure (*DES*)*. The DES can be computed using an all-pairs shortest-path algorithm such as Floyd-Warshall. This applies when the durations in the event delays are integer intervals.

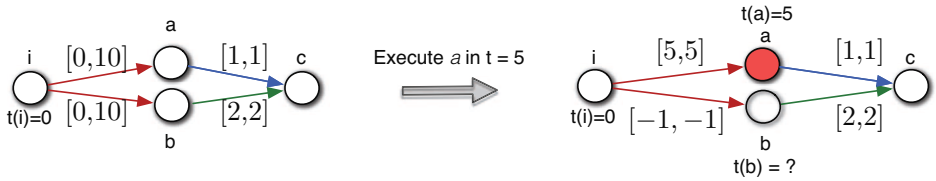

Figure 6.7: The problem with local propagation. Let ε be an event structure and the graph at the left its constraint graph after event i is executed at $t = 0$. It seams coherent to execute event a at $t = 5$, by regarding only the neighbors of a. This will lead to an incoherence because b should have been executed at $t = 4$ to achieve a consistent graph. The solution is to add a delay labeled by $[1,1]$ from b to a, which is obtained by transforming the event structure into a dispatchable event structure.

Proposition 6.3.5 (Correctness of the Dispatchable Event Structures)**.** *Let ε be an event structure and* $\text{des}(\varepsilon)$ *its* DES *form. Both event structures have equivalent constraints and local propagation in the constraint graph of* $\text{des}(\varepsilon)$ *simulates full propagation.*

Proof. This holds by previous results on simple temporal problems. Theorems "every *all-pairs shortest-path network (*APSPN*)* is dispatchable [using only local propagation]", presented in [Muscettola 1998]; locally consistent assignment can be extended to a global one in a dispatchable graph, presented in [Muscettola 1998]; and equivalence between a APSPN and the original network, presented in [Dechter 1991]. □

6.3.3 Ntcc model of a score

In what follows we present a ntcc model of an interactive score. We start from the dispatchable normal form of the event structure semantics of the score, namely $\varepsilon^* = \langle E, l^*, R \rangle$.

We associate a finite domain variable p_i (point i) with each $e_i \in E$ and represent every $e_i \mapsto^{\Delta} e_j \in R$ by a constraint $p_j \in p_i + \Delta$. The value ∞, potentially present in some Δ, is considered a fixed number of the model (denoted n_∞): a given "big" value in the finite domain constraint system. In this model, events are called points and event delays are called intervals to avoid confusion with the action-event occurring when some external agent triggers an interaction point of the score.

Ntcc processes start with capital letters; variables start with lower-case letters. We define a ntcc process $intervals_R$ posting the above constraints persistently[2]:

$$Interval_{i,j,\Delta} \stackrel{def}{=} \textbf{!tell}\ (p_i + \Delta = p_j)$$

$$Intervals_R \stackrel{def}{=} \prod_{(e_i,\Delta,e_j)\in R} Interval_{i,j,\Delta}$$

We have two types of points: interactive (*iPoint*) and static points (*sPoint*). Process $Launch_i$ adds the constraint $launch_i$ in the current time unit, and persistently assigns p_i with the current value of $clock$ and $launched_i$ to true. Constraint $launch_i$ is used to prove the correctness of the model and also during implementation to know in which time unit point i is executed. Event ev_i is the user event associated to point i: it represents a user interaction. Process *Score* is parametrized by a function $\Pr : P \to \mathbb{P}(P)$ that returns the set of points that are predecessors of a given point, a set $S \subseteq E$, a set $I \subseteq E$ and R.

- We recall from Section 4.2 that Process $x \leftarrow y$ assigns persistently the current value of y to x, and process $*_{[0,n]}P$ launches a process P at some time unit within the interval $[0,n]$. We also recall from Chapter 4 that function $\text{le} : E \to Act$ returns the label of an event.
- Function predecessors Pr is defined by $\Pr(i) = \{j | \langle e_j, e_i, \Delta\rangle \in R\}$.
- Set I represents the interactive objects, $I = \{e | e \in E \wedge (InteractiveObject, a) \in \text{le}(e)\}$.
- Set S represents the static points, $S = \{e | e \in E \wedge (InteractiveObject, a) \notin \text{le}(e)\}$.

$$Launch_i \stackrel{def}{=} \textbf{tell}\ (launch_i) \parallel p_i \leftarrow clock \parallel \textbf{!tell}\ (launched_i) \parallel \textbf{next !}\ \textbf{tell}\ (\neg launch_i)$$

$$\begin{aligned} iPoint_{i,\Pr} \stackrel{def}{=} \textbf{when}\ \textstyle\bigwedge_{j\in\Pr(i)}\ launched_j\ \textbf{do}\ (&\\ &\textbf{when}\ clock+1 > p_i\ \textbf{do unless}\ launched_i\ \textbf{next}\ Launch_i\\ &\parallel \textbf{unless}\ clock+1 < p_i\ \textbf{next when}\ ev_i\ \textbf{do}\ Launch_i)\\ \parallel &\textbf{unless}\ launched_i\ \textbf{next}\ iPoint_{i,\Pr} \end{aligned}$$

$$\begin{aligned} sPoint_{i,\Pr} \stackrel{def}{=} \textbf{when}\ \textstyle\bigwedge_{j\in\Pr(i)}\ launched_j\ \textbf{do}\ (&\\ &\textbf{unless}\ clock+1 < p_i \vee launched_i\ \textbf{next}\ Launch_i\\ &\parallel \textbf{when}\ clock+1 < p_i\ \textbf{do next}\ sPoint_{i,\Pr}) \end{aligned}$$

[2] Agent $\prod$ represents a generalized parallel composition of processes.

$$\| \textbf{unless } launched_i \textbf{ next } sPoint_{i,\mathrm{Pr}}$$

$$Points_{I,S,\mathrm{Pr}} \stackrel{def}{=} \prod_{e_i \in I} iPoint_{i,\mathrm{Pr}} \parallel \prod_{e_i \in S} sPoint_{i,\mathrm{Pr}}$$

Process *User* chooses between launching or not a user event. *Clock* ticks forever. Process *Score* is parametrized by I, S, Pr and R. *Score* adds a constraint $0 < p_i < n_\infty$ to allow that $clock + 1 < p_i$ (which appears in the third line of process *iPoint*) be eventually deduced even when an object's duration is said to be infinite.

$$User_I \stackrel{def}{=} \prod_{e_i \in I} *_{[0,n_\infty]} \textbf{ tell } (ev_i) + \textbf{skip}$$

$$Clock(k) \stackrel{def}{=} \textbf{when } k < n_\infty - 1 \textbf{ do } (\textbf{tell } (clock = k) \parallel \textbf{next } Clock(k+1))$$

$$Score_{I,S,Pr,R} \stackrel{def}{=} Points_{I,S,Pr} \| Intervals_R \| User_I \| Clock(0) \parallel \prod_{i \in S \cup I} !\textbf{tell}(0 < p_i < n_\infty)$$

We shall prove that the ntcc model is correct with respect to the dispatchable normal form of the event structures semantics of a score; therefore, any possible output of ntcc respects the temporal constraints of the event structures semantics of the score. The intuition of the following proposition is to show that the time unit indexes in which the process that represents a score entails $launch_i$ are contained among the possible time units in which the corresponding event e_i appears in the timed event traces in the event structures semantics.

Proposition 6.3.6. *Let s be a score, $\varepsilon^* = \mathrm{des}(\mathrm{normal}(\mathrm{es}(s))) = \langle E, l^*, R \rangle$ its event structures semantics, $\mathrm{tc}(\varepsilon^*)$ its temporal constraints, $P = Score_{S,I,\mathrm{Pr},R}$ the ntcc process that represents the score, $[[P]]$ the denotation of the ntcc process, and T_i is the set of indexes j such that $[[P]]_{j+1}$ entails $launch_i$. It holds for all sequences in $[[P]]$, $n = \mathrm{card}(E)$, that $T_1 \times T_2 \ldots \times T_n \subseteq \mathrm{Solutions}(\mathrm{tc}(\varepsilon^*))$.*

Proof. This proof is presented in detail in Appendix A.1. Table 6.3.3 summarizes the results of this proof.

□

ε^*	Values of T_i for process $Score_{\varepsilon^*}$	$\mathrm{tc}(\varepsilon^*)$
$((i))$	$T_i \subseteq [1, n_\infty]$	$t_i \in \mathbb{N}$
$(i) \stackrel{\Delta}{\mapsto} (j)$	$T_i = \{1\} \wedge T_j \in T_i + \min(\Delta)$	$t_i \in \mathbb{N} \wedge t_j \in t_i + \Delta$
$((i)) \stackrel{\Delta}{\mapsto} (j)$	$T_i \subseteq [1, n_\infty] \wedge T_j \in T_i + \min(\Delta)$	$t_i \in \mathbb{N} \wedge t_j \in t_i + \Delta$
$((i)) \stackrel{\Delta}{\mapsto} ((j))$	$T_i \subseteq [1, n_\infty] \wedge T_j \in T_i + \Delta$	$t_i \in \mathbb{N} \wedge t_j \in t_i + \Delta$
$(i) \stackrel{\Delta}{\mapsto} ((j))$	$T_i = \{1\} \wedge T_j \in T_i + \Delta$	$t_i \in \mathbb{N} \wedge t_j \in t_i + \Delta$

Table 6.3: Temporal constraints to launch the events in the ntcc model Vs. temporal constraints of the event structures semantics. We denote an interactive object as $((i))$ and a static one as (i).

6.3.3.1 Dealing with nominal times

We define two types of intervals: static and interactive. *Static intervals* are between two static points and *interactive intervals* involve at least one interactive point. After computing the normal form, if a point is associated to at least one interactive object, it is consider an interactive point. We define two sets

$R_{stn} \stackrel{def}{=} \{(e_i, \text{stn}(a), e_j) |$ the object a such that $\text{sp}(\text{parent}(a)) = e_i, \text{sp}(a) = e_j$ and its stn is the minimum $\}$

$R_{nd} \stackrel{def}{=} \{(e_i, \text{nd}(a), e_j) |$ exists an object a such that $\text{sp}(a) = e_i, \text{ep}(a) = e_j$ and its nd is the minimum $\}$

Note that we choose the object with minimum nominal starting time and minimum nominal duration to calculate sets R_{stn} and R_{nd} because once the event structures semantics are normalized, an event can be associated to several temporal objects. In what follows, we define a ntcc process $intervals_{R,R_{stn},R_{nd}}$ posting the following constraints persistently.

$iInterval_{i,j,\Delta} \stackrel{def}{=}$ **!tell** $(p_i + \Delta = p_j)$, where Δ is an interval

$sInterval_{i,j,nt} \stackrel{def}{=}$ **!tell** $(p_i + nt = p_j)$, where $nt \in \mathbb{N} \cup \infty$

$$Intervals_{R,R_{stn},R_{nd}} \stackrel{def}{=} \prod_{(e_i,\Delta,e_j)\in R} dInterval_{i,j,\Delta} \parallel \prod_{(e_i,stn,e_j)\in R_{stn}} sInterval_{i,j,stn} \parallel \prod_{(e_i,nd,e_j)\in R_{nd}} sInterval_{i,j,nd}$$

We argue that correctness is preserved because the temporal constraints of the dispatchable normal event structures semantics of the score contain the constraints derived from the nominal starting time and the nominal duration.

6.4 Summary and Discussion

We equipped the structural definition of a score, presented in Chapter 5, with a directed-tree hierarchy. We do not use an acyclic graph hierarchy because it adds a higher complexity in the semantics, although it could be useful to model some scenarios; for instance, a theater performance where the same background song is played in several scenes, thus the background-song temporal object could have several parents, for instance, when there are different events with the same action.

We extend the event structures semantics of Chapter 5 by adding the hierarchy. We proved that the temporal constraints of the event structure are equivalent to the temporal constraints of the score as a corollary of Proposition 5.2.9.

We also include a nominal starting time and a nominal duration which are useful, for instance, to visualize the score in a timeline and to model music pieces. Nominal starting times and nominal durations were not consider in [Toro 2012b]. Note that nominal starting

times and nominal durations are used in i-score and Virage.

Finally, we gave operational semantics to this model based on ntcc. In order to define operational semantics, we define a normal form in which no zero-delays are allowed, and a dispatchable form in which local propagation is equivalent to global propagation. We prove that the temporal constraints obtained from the ntcc model are a subset of those from the event structures.

The event structures semantics for interactive scores defined in this chapter use only a rather limited subset of time event structures. Nonetheless, we chose event structures because it easy allows us to represent conditional branching and choices by adding conflicts in Chapter 7.

We left as future work to model the behavior of the nominal starting times and the nominal durations in the event structures semantics. We also left as future work to compare event structures semantics and operational semantics by the means of a *commutative diagram*[3], well-known from category theory.

[3] http://en.wikipedia.org/wiki/Commutative_diagram

CHAPTER 7

Time Conditional-Branching Scores

Contents

We recall from Chapter 1 that most music pieces are linear: their scores are read from top to down and left to right. Non-linear pieces are also called *open works*. Open works may have openness of interpretation or openness of semantic content [Vickery 2003]. Conditional branching is essential to describe pieces with openness of interpretation. As we described in Chapter 3, Ranaivoson and Allombert have already studied the problem of extending interactive scores with conditional branching [Ranaivoson 2009, Allombert 2009]. Unfortunately, they represented conditional branching relations and temporal relations as separated entities, thus there is no unified way to represent conditional branching and temporal relations in the same scenario. In this chapter, we propose an extension of interactive scores that allows to specify conditional branching and temporal relations.

As we showed in Chapter 2, we have already introduced interactive scores with loops in [Toro 2010c, Toro 2010b]. Nonetheless, the lack of an abstract semantics makes it difficult

its comprehension and the formalization of properties of the scores. In what follows, we introduce a formal structural definition and an event structures semantics of interactive scores with conditional branching without loops. Afterwards, we introduce the structural definition of a more general model that includes loops. We also present some insights on how to define operational semantics for these two models.

7.1 Structural Definition without Loops

In this section, we propose an extension of interactive scores with conditional branching. These new interactive scores are composed by *time conditional relations, choices* and *temporal objects*. A temporal object has two *point identifiers* to control its starting and ending times, as usual. A temporal object also has variables visible by itself and its children. As usual, there is a special type of temporal objects called *interactive objects*. All the points of the score are executed. Time conditional relations include a duration and a constraint (i.e., a logical condition). Time conditional relations and choices are used to state weather some objects launch a *silent action* or a *visible action*; therefore, all temporal objects are executed and all branches starting on the same point have the same set of durations. Branches with different durations can produce incoherent executions; for instance, the reader may see the score in Figure 7.1. In the operational semantics, visible actions should include operations to add constraints to the variables.

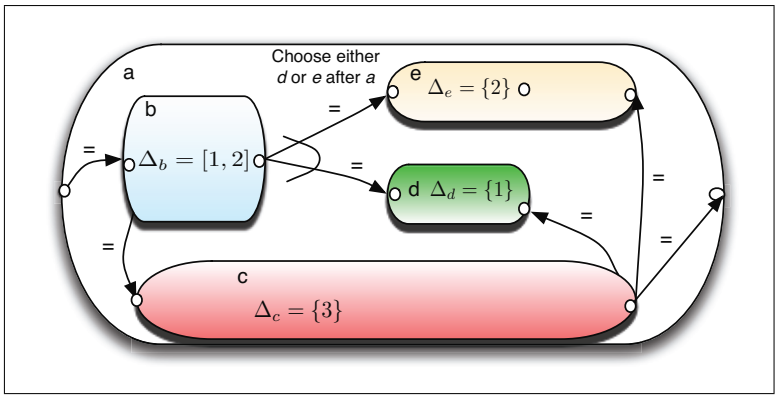

Figure 7.1: A score with conditional branching in which branches starting on the same point can have different durations. An arc crossing two temporal relations represent a mutually-exclusive choice among the successors of the point. The problem is that the choice between objects *d* and *e* depends on the duration chosen for object *a*; however, they are non-deterministically chosen. Relations labeled by "=" represent a duration of $\{0\}$.

7.1.1 Temporal objects

The duration of the temporal objects is an arbitrary interval and point identifiers are either the start or the end of a temporal object, as in Chapter 6.

Definition 7.1.1 (Temporal object)**.** *Let P be a set of point-ids and $\mathscr{V}$ a set of variable names. A* Temporal object *$\langle I,H\rangle$ is a directed tree with nodes I and arcs $H \subseteq I \times I$. Elements of I, the* intervals*, are tuples $\langle sp,ep,\Delta,V\rangle$, where $sp,ep \in P, sp \neq ep$, are called* start *and* end *points, respectively, $\Delta \subseteq \mathbb{N}$ is a set of* durations*, and $V \subseteq \mathscr{V}$ a set of* variables *for the temporal object and its children. For $i \in I$, functions* $\mathrm{sp}(i)$, $\mathrm{ep}(i)$, $\mathrm{d}(i)$, $\mathrm{v}(i)$ *return the start, end, duration and variables, respectively, of interval i.*

As it was defined in Chapter 6, a temporal object whose interval has $\Delta = \{0\}$ and has no subtrees is called an *interactive object*. All points sp (resp. ep) of the intervals of a temporal object are pairwise distinct. Arcs of a temporal object represent hierarchical relations: $(i_1,i_2) \in H$ means that interval i_1 contains interval i_2 (and thus the temporal object defined by the subtree rooted by i_2). The set of all temporal objects is $\mathscr{T}$.

7.1.2 Time conditional relations

Points $p,q \in P$ are supposed to be positioned on a timeline, as in Chapter 6. We extend temporal relations to *time conditional relations* by adding a condition (a constraint). Constraints are defined in a constraint system, thus a time conditional relation is parametric on a *constraint system* (see Def. 4.2.1). A score includes a constraint system that defines the constraints that can be used in the score and a relation to know which constraints are deduced from others. Time conditional relations contain conditions syntactically constructed with the variables in the scope of the departing temporal object. Choices represent mutual conflicts among start points. It is still an open issue how to define the lifespan of variables.

We recall from Section 4.2 , that constraints can be thought of as first-order formulae over Σ, thus there is an underlying first order language $L = \langle \Sigma, \mathscr{V}, S\rangle$, where $\mathscr{V}$ is a countable set of variables and S is a set of logic symbols $\neg, \wedge, \vee, \Rightarrow, \forall, \exists, \texttt{true}, \texttt{false}$. We can decree that $c \vdash d$ (d can be deduced from c) if the implication $c \Rightarrow d$ is valid in Δ, as it was proposed by Valencia *et al.* in [Valencia 2002].

We recall from the Chapter 4 that, for operational reasons, $\vdash$ must be decidable and it is desirable for real-time purposes to be decidable in polynomial time. A commonly used constraint system is *bounded finite domain* (**FD**$[n]$) that defines arithmetic relations among variables whose domains are finite ranges of values $\{0,\ldots,n-1\}$. Operator $\vdash$ is not decidable in polynomial time for all possible constraints in finite domain; however, for some commonly used subsets of linear constraints, $\vdash$ can be decided in polynomial time in [Bordeaux 2011].

As in Chapter 6, we use the notation $T + \Delta =^{def} \{t' | t' = t + \delta, t \in T, \delta \in \Delta\}$ for temporal constraints of duration. Temporal positions of $p,q \in P$ are said to be constrained by $\nu(q) \in \nu(p) + \Delta$. The set of all temporal relations is $\mathscr{R}$. We recall that function $\nu : P \to \mathbb{P}(\mathbb{N})$ gives the set of *potential time positions* for each point $p \in P$.

Definition 7.1.2 (Time Conditional Relation)**.** *Given a constraint system with variables V, a temporal relation is a tuple $\langle p,\Delta,c,q\rangle$ where $\Delta \subseteq \mathbb{N}$ is the duration; $p,q \in P$; and c is the condition (a constraint defined with variables from V) to decide whether the object started by q and its children perform a visible action or not.*

7.1.3 Interactive scores

A score is a tuple consisting of a constraint system, a set of choices (points in mutual conflict), a set of variables, a set of point identifiers, an object hierarchy and a set of time conditional relations over those objects. Two interactive objects cannot start at the same time; this constraint is also defined in hierarchical scores of Chapter 6.

There are some new restrictions concerning conditions in a score. If q is the ending point of a static object, c must be `true` because once a temporal object has started (and has launched its start action), the end action must be launched as well. The previous restriction is also valid if q is the start or end point of an interactive object. It will be awkward to disable interactive objects because they do not launch an external action. Finally, for simplicity, there is only one relation between the start and end points of an object. We also say that all the points in mutual conflict must have at least one common *predecessor*. A point p is a predecessor of a point q if there exists a relation $r = \langle p, \Delta, c, q \rangle$.

Example 7.1.3. As an example, consider the score in Figure 7.2 whose structural definition can be seen in Table 7.1. A reduced example that does not include hierarchy is presented in Figure 7.3.

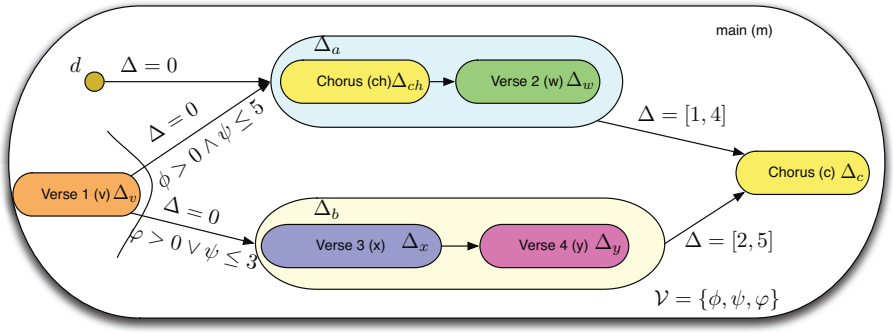

Figure 7.2: Example of a score. An arc crossing two or more arrows represents a choice. Using the hierarchy is possible to extend the exclusive choice to two pair of objects. To execute two instances of the same temporal object, the object must be duplicated, as it is done with *chorus*. Note that $\Delta_b + [2,5] = \Delta_a + [1,4]$ must hold because all branches starting on the same point have the same duration.

Definition 7.1.4 (Interactive Score)**.** *An* interactive score *is a temporal object equipped with choices, a set of time conditional relations and a set of variables: A tuple* $\langle CS, \mathbb{C}, \mathscr{V}, P, o, R \rangle$*, where* $CS = (\Sigma, \Delta)$ *is a constraint system whose underlying first order language is* $L = \langle \Sigma, \mathscr{V}, S \rangle$*,* $\mathbb{C} \subseteq \text{partitions}(P)$ *is a set of sets of start points that are in mutual conflict (representing choices) that contains at least two points,* $\mathscr{V}$ *is a generalized union of the sets of variables of each interval in* o *and its children,* P *is a set of point ids,* o *is a temporal object and* $R \subseteq (P \times \mathbb{P}(\mathbb{N}) \times C \times P)$ *the time conditional relations, whose conditions are defined over the variables in* $\mathscr{V}$ *and they are valid predicates in the constraint*

system CS. Set C is the set of constraints in the underlying constraint system composed by variables in $\mathscr{V}$. A relation $\langle p,\Delta,c,q\rangle \in R$, iff

1. *p,q are distinct points of intervals in o and $\nu(q) \in \nu(p)+\Delta$;*
2. *two interactive objects do not occur at the same time;*
3. *if q is the start or end point of an interactive object, the condition c must be equal to* `true`*;*
4. *all the points in a set in $\mathbb{C}$ have at least one common predecessor;*
5. *if p is the start point of a temporal object a and q is the end point of a, condition c must be equal to* `true`*;*
6. *$\forall S \subseteq P, S \subseteq R[S] \Rightarrow S = \emptyset$, where $R[S]$ is the map of all the points in the set S over the relation R, namely $R[S] = \{q | p \in S, \langle p,\Delta,c,q\rangle \in R\}$. This condition formalizes the fact that time conditional relations in a score are not used to describe loops.*

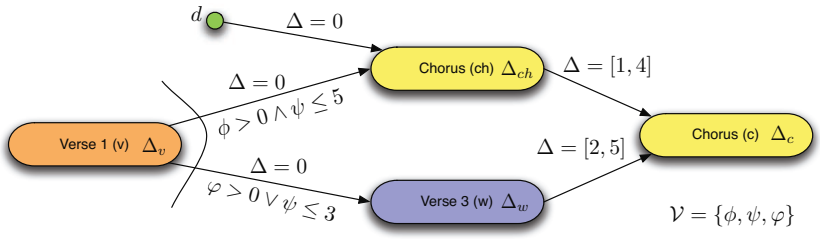

Figure 7.3: Non-hierarchical version of the score in Figure 7.2. Temporal objects *a* and *b* in Figure 7.2 are needed to define that temporal objects *ch* and *w* are mutual exclusive with objects *x* and *y*. If we only want to define choice between *ch* and *w*, no additional objects are needed.

As in Chapter 6, set R also includes the relations derived from (1) the duration of temporal objects and (2) the hierarchical structure. As an example, when i_2 is contained in i_1, the starting point of i_2 must not occur before the starting point of i_1.

1. For each $i \in I$, $\langle \mathrm{sp}(i), \mathrm{d}(i), \texttt{true}, \mathrm{ep}(i)\rangle \in R$
2. For each $(i_1,i_2) \in H$, $\langle \mathrm{sp}(i_1), [0,\infty), \texttt{true}, \mathrm{sp}(i_2)\rangle \in R$ and $\langle \mathrm{ep}(i_2), [0,\infty), \texttt{true}, \mathrm{ep}(i_1)\rangle \in R$

For simplicity, in this chapter, we omit nominal durations and nominal starting times. We recall that we denote $\mathscr{S}$ the set of all interactive scores, and function $\mathrm{ts}(s)$ gives the conjunction of the temporal constraints of the score.

Points, choices and temporal objects
$I = \{m, v, d, ch, w, c, x, y, a, b, d\}$
$H = \{(m,v),(m,d),(m,a),(m,b),(m,c),(m,d),(a,ch),(a,w),(b,x),(b,y)\}$
$P = \{sp_m, ep_m, sp_d, ep_d, sp_c, ep_c, sp_v, ep_v, sp_a, ep_a, sp_{ch}, ep_{ch}, sp_w, ep_w,$ $sp_b, ep_b, sp_x, ep_x, sp_y, ep_y, sp_c, ep_c\}$
$\mathbb{C} = \{\{sp_a, sp_b\}\}$
$m = \langle sp_m, ep_m, \Delta_m, \{\phi, \psi, \varphi\}\rangle$
$v = \langle sp_v, ep_v, \Delta_v, \emptyset\rangle$
$w = \langle sp_w, ep_w, \Delta_w, \emptyset\rangle$
$ch = \langle sp_{ch}, ep_{ch}, \Delta_{ch}, \emptyset\ \rangle$
$c = \langle sp_c, ep_c, \Delta_c, \emptyset\ \rangle$
$x = \langle sp_x, ep_x, \Delta_x, \emptyset\ \rangle$
$y = \langle sp_y, ep_y, \Delta_y, \emptyset\ \rangle$
$a = \langle sp_a, ep_a, \Delta_a, \emptyset\ \rangle$
$b = \langle sp_b, ep_b, \Delta_b, \emptyset\ \rangle$
$d = \langle sp_d, ep_d, \{0\}, \emptyset\ \rangle$
Explicit time conditional relations
$R = \{\langle sp_d, 0, \texttt{true}, sp_a\rangle$
$\langle ep_v, 0, \phi > 0 \wedge \psi \leq 5, sp_a\rangle$
$\langle ep_v, 0, \varphi > 0 \vee \psi \leq 3, sp_b\rangle$
$\langle ep_a, [1,4], \texttt{true}, sp_c\rangle$
$\langle ep_b, [2,5], \texttt{true}, sp_c\rangle$
$\langle ep_{ch}, [0,\infty), \texttt{true}, sp_w\rangle$
$\langle ep_x, [0,\infty), \texttt{true}, sp_y\rangle\}$

Table 7.1: Structural definition of the score in Figure 7.2. A temporal object is a directed tree $o = \langle I, H\rangle$, a time conditional relation is a tuple $\langle p, \Delta, c, q\rangle$ and the score is defined by $s = (FD[n], \{\{sp_a, sp_b\}\}, \{\phi, \psi, \varphi\}, P, m, R)$.

7.2 Event Structures Semantics without Loops

We define the formal semantics of interactive scores in event structures. The events represent the visible and invisible start or end points of a temporal object. An interactive object is represented by a single event. Time conditional relations are modeled with event delays and conflicts. We recall from Chapter 4 that we denote by the functions $\mathrm{E}(\varepsilon)$,$\mathrm{l}(\varepsilon)$, $\mathrm{R}(\varepsilon)$, and $\mathrm{C}(\varepsilon)$ each component of an event structure ε.

7.2.1 Temporal objects

A static temporal object a is represented by four events $s_a, s_a,' e_a, e'_a$ (visible start event, silent start event, visible end event, silent end event) together with all events in the translation of the objects in the subtrees rooted by a. The labels of events are pairs $(type, i)$, where $type \in Type = \{startPoint, endPoint, interactiveObject, silentStartPoint, silentEndPoint\}$ and i is the interval in the root of the temporal object giving rise to the event.

Example 7.2.1. As an example, Figure 7.4 shows the encoding of a temporal object along with the conditional temporal relations representing its duration. Figure 7.5 shows the encoding of a hierarchical temporal object.

Figure 7.4: The encoding (into event structures) of a temporal object and the time conditional relations representing its duration. Black-colored events have invisible actions.

Definition 7.2.2 (Temporal object encoding)**.** *The encoding of a temporal object (a) is a function* eto : $\mathscr{T} \to \mathscr{E}$. *Let i be the interval in the root of a, the encoding of a is defined by*

1. *if* $i = \langle sp, ep, \{0\}, V\rangle$ *and a has no subtrees (interactive object),*
   $\mathrm{eto}(a) = \langle \{s_a\}, \{(s_a, (interactiveObject, i))\}, \emptyset, \emptyset\rangle$

2. *if* $i = \langle sp, ep, \Delta, V\rangle$ *(static temporal object), then* $\mathrm{eto}(a) = \langle E, l, \emptyset, \emptyset\rangle$, *where*
   $E = \{s_a, e_a, s'_a, e'_a\} \cup \bigcup_{x \in \mathrm{subtrees}(a)} \mathrm{E}(\mathrm{eto}(x))$
   $l = \{(s_a, (startPoint, i)), (e_a, (endPoint, i)), (s'_a, (silentStartPoint, i)),$
   $(e'_a, (silentEndPoint, i))\} \cup \bigcup_{x \in \mathrm{subtrees}(a)} \mathrm{l}(\mathrm{eto}(x))$

As opposed to the event structures encoding for hierarchical interactive scores, the above definition does not guarantee that there are unique start and end events in the translation of a static temporal object. Nonetheless, given an object, a point and a type, there is only one event associated, according to Def. 7.2.2.

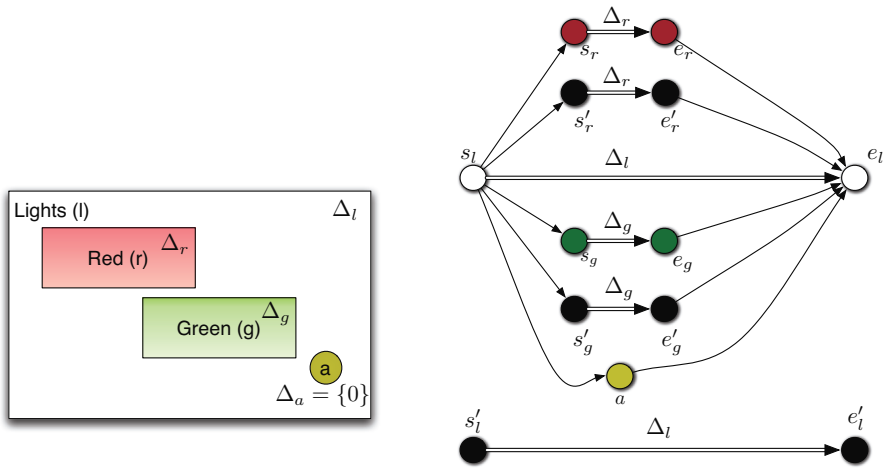

Figure 7.5: The encoding (into event structures) of a temporal object and the time conditional relations representing the hierarchy and duration of the objects. Black-colored events have invisible actions.

Definition 7.2.3 (Function from points to events (pe))**.** *Let o be a temporal object and P_o the set of points contained in o, function* $\mathrm{pe}(\mathrm{o},\mathrm{p},\mathrm{type}) : \mathscr{T} \times P \times Type \to \mathrm{E}$ *returns the event in* $\mathrm{eto}(o)$ *labeled with* type.

7.2.2 Time conditional relations

The encoding of a time conditional relation is a pair that contains a set of event delays and a set of conflicts; in contrast to the hierarchical model in which only event delays were needed.

Example 7.2.4. Figures 7.6 and 7.7 the encodings of time conditional relations.

Figure 7.6: The encoding of two temporal objects a and b. The time conditional relations representing its duration and a time conditional relations express precedence between a and b with a condition ϕ. Mutual conflicts are represented by a dashed line. Black-colored events have invisible actions.

Definition 7.2.5 (Time conditional relations encoding (etr))**.** *Each time conditional relation is represented by event delays and conflicts. The encoding of a relation r is given by the*

function $\text{etr} : \mathscr{R} \to ((E \times E \to \mathbb{P}(\mathbb{N} \cup \{\infty\})) \times (E \times E))$. *Function* etr *returns a pair that contains an event delay function and an event conflict relation.*

Let $r = \langle p, \Delta, c, q \rangle \in \mathscr{R}$, *where point p belongs to object a and point q belongs to object b. Events* ev_p, ev_q, ev'_p, ev'_q *are the (visible and silent) events associated, respectively, to points p and q by function* pe. *The encoding of the event delay function in* $\text{etr}(r)$ *is defined thus:*

- *Case* $a = b$. $R = \{ev_p \mapsto^{\Delta} ev_q, ev'_p \mapsto^{\Delta} ev'_q\}$
- *Case* $a \neq b$ *and b is contained in a.* $R = \{ev_p \mapsto^{\Delta} ev_q\}$
- *Case* $a \neq b$ *and b is not contained in a.*
  $R = \{ev_p \mapsto^{\Delta} ev_q, ev'_p \mapsto^{\Delta} ev_q, ev_p \mapsto^{\Delta} ev'_q, ev'_p \mapsto^{\Delta} ev'_q\}$

and the conflict relation in $\text{etr}(r)$ *is defined as follows*

- *Case* $c =$ `true`. *Conflict relation* $\rightsquigarrow = \emptyset$ [1]
- *Case* $c \neq$ `true` *and type of* ev_q *is startPoint.*
  $\rightsquigarrow = \{(ev_q, ev'_q), (ev'_q, ev_q), (\text{pe}(b, \text{ep}(b), \mathit{visibleEndPoint}), ev'_q)\}$
  $\cup \bigcup_{x \in \text{subtrees}(b)} \text{E}(\text{eto}(x)) \times \{ev'_q\}$

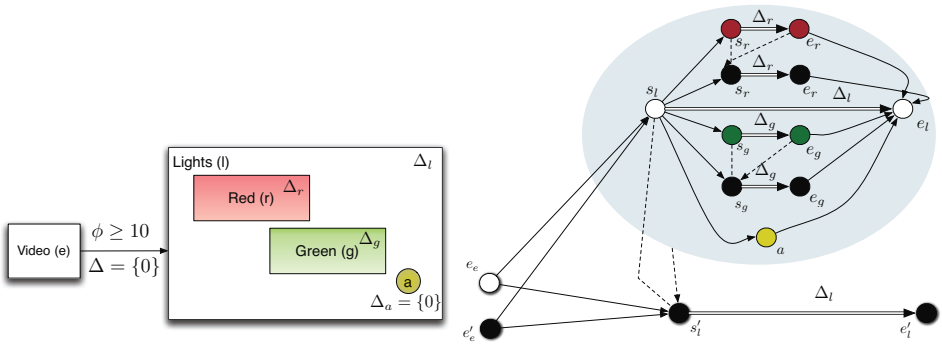

Figure 7.7: Encoding of two hierarchical objects. There is a mutual conflict between s_l and s'_l because the execution of s_l must disable s'_l to forbid s'_l to disable the events representing the children of l when s_l be executed. All the colored events in the gray region are in conflict with event s'_l

7.2.3 Choices

Choices are represented by conflicts.

[1] We recall that if ev_q is an end point, c is always `true`.

Definition 7.2.6 (Choice encoding)**.** *Let s be a score, we recall that set $\mathbb{C}$ contains the sets of points that are in mutual conflict. Function* $\mathrm{co} : \mathbb{P}(P) \to (E \times E)$ *translates a set of points into conflicts of its corresponding events. Let $C \in \mathbb{C}$, point p belong to object a and q to object b, events $ev_{sa}, ev_{sb}, ev_{ea}, ev_{eb}$ be the visible-action events for the start of a, start of b, end of a and end of b, respectively. The encoding* $\mathrm{co}(C)$ *is a conflict relation defined for each pair-wise different pair of points $p, q \in C$:*

$$\begin{aligned}\rightsquigarrow_{=} \{&(ev_{sa}, ev_{sb})(ev_{sb}, ev_{sa}), (ev_{ea}, ev_{sb}), (ev_{eb}, ev_{sa})\} \\ &\cup \bigcup_{x \in \mathrm{subtrees}(b)} \mathrm{E}(\mathrm{eto}(x)) \times \{ev_{sa}\} \\ &\cup \bigcup_{x \in \mathrm{subtrees}(a)} \mathrm{E}(\mathrm{eto}(x)) \times \{ev_{sb}\}\end{aligned}$$

Example 7.2.7. Figures 7.8 and 7.9 are examples of the encoding of choices.

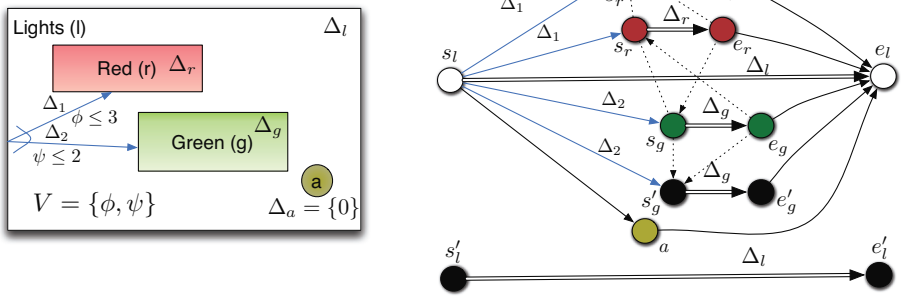

Figure 7.8: Example of a light control. There is a choice between the red and green light given by conditions $\phi \leq 3$ and $\psi \leq 2$, which are not necessarily mutually exclusive.

7.2.4 Interactive scores

The encoding of a score is given by adding the event delays and conflicts from the encoding of the time conditional relations, and the conflicts from the choices to the encoding of the root of the score.

Example 7.2.8. As an example, the encoding of Figure 7.2 is presented in Figure 7.10.

Definition 7.2.9 (Interactive score encoding)**.** *The encoding of an interactive score $s = \langle CS, \mathbb{C}, \mathscr{V}, P, o, R\rangle$ is given by the function* $\mathrm{es} : \mathscr{S} \to \mathscr{E}$ *that translates scores into event structures. Let* $\mathrm{eto}(o) = \langle E, l, \emptyset, \emptyset\rangle$, $\bigcup_{r \in R} \mathrm{etr}(r) = \langle R', \rightsquigarrow\rangle$, *then* $\mathrm{es}(s) = \langle E, l, R', \rightsquigarrow \cup \bigcup_{C \in \mathbb{C}} \mathrm{co}(C)\rangle$, *where $\rightsquigarrow$ is the conflict relation obtained from* $\bigcup_{r \in R} \mathrm{etr}(r)$.

To prove the correctness of the event structures semantics, we will need to prove the following propositions. This is left as future work.

1. There are no conflicts when there are no choices and all the conditions are `true`. In fact, a conditional branching score without choices and with all conditions labeled by `true`, is equivalent to a hierarchical score.

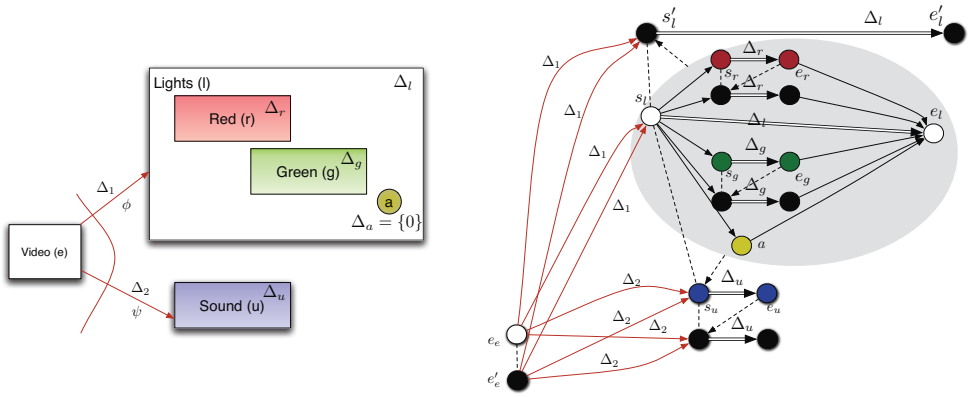

Figure 7.9: Example of a score with a light control and a sound. There is a choice between the turning on the lights and playing a sound given by conditions, which are not necessarily mutually exclusive. All colored events in the gray region are in conflict with s_u and s'_l.

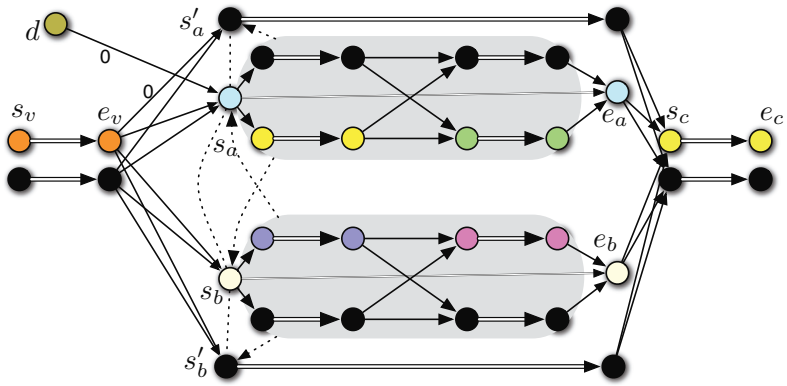

Figure 7.10: Semantics of the score in Figure 7.3. Black events have silent actions. Colored events in the upper gray region are in conflict with s_b and s'_a and colored events in the lower gray region are in conflict with s_a and s'_b. For simplicity, some events are not labeled.

2. If there is a relation $\langle p, \Delta, c, q \rangle$, where c is not `true`, and q is a start point of temporal object a, there is at least one trace where a and its children's events do not appear, and at least one trace where their events appear.

3. No more than one of the events obtained by encoding the start points that are in a mutually exclusive choice appears in a trace of execution.

4. The temporal constraints of the traces of $\text{es}(s)$, replacing the invisible-labeled events of a score by visible-labeled events when the visible-labeled events do not appear in the event trace, correspond to the temporal constraints of s.

7.3 Towards Operational Semantics without Loops

In this section, we sketch some ideas on how to define operational semantics for scores without loops. We believe that the ntcc model presented in Chapter 6 can be extended with the concept of invisible events, visible events and conflicts among events. As an example, we can use the constraint $\bigwedge_{e_i \in \text{Pr}(j)} (launched(e_i) \vee disabled(e_i))$ to check that all the predecessors of an event have been launched or disabled. In addition, instead of using ev_i to represent a user event, we can use a more general definition $condition_i$ that can be a user event, a constraint representing the condition, or a combination of both.

Variables in each temporal object could be modeled as local variables with a cylindrical constraint system, using existential quantification over constraints.

Example 7.3.1. In Figure 7.11, we present the normal form of Figure 7.10.

The concept behind the normal form is to collapse together events that happen at the same time, but are not in conflict. The normal form of the event structure semantics of a score with conditional branching is essential to define operational semantics, as we have shown in Chapter 6. Nonetheless, the normalization rule would be far more complex than the one for hierarchical scores; this is left as future work.

It is also left as future work how to define visible actions that add constraints to the environment and the life span of the variables.

7.4 Structural Definition with Loops

To define scores with loops, we remove the restriction that forbids loops in the structural definition of conditional branching interactive scores without loops, and we add the concept of the *interpretation of the condition*. Another difference is that temporal constraints cannot be defined in the same manner because the problem will be overconstraint. We left as future work a new definition of the temporal constraints of a score with loops.

7.4.1 Temporal objects

A temporal object is defined as in scores without loops introduced in Def. 7.1.1.

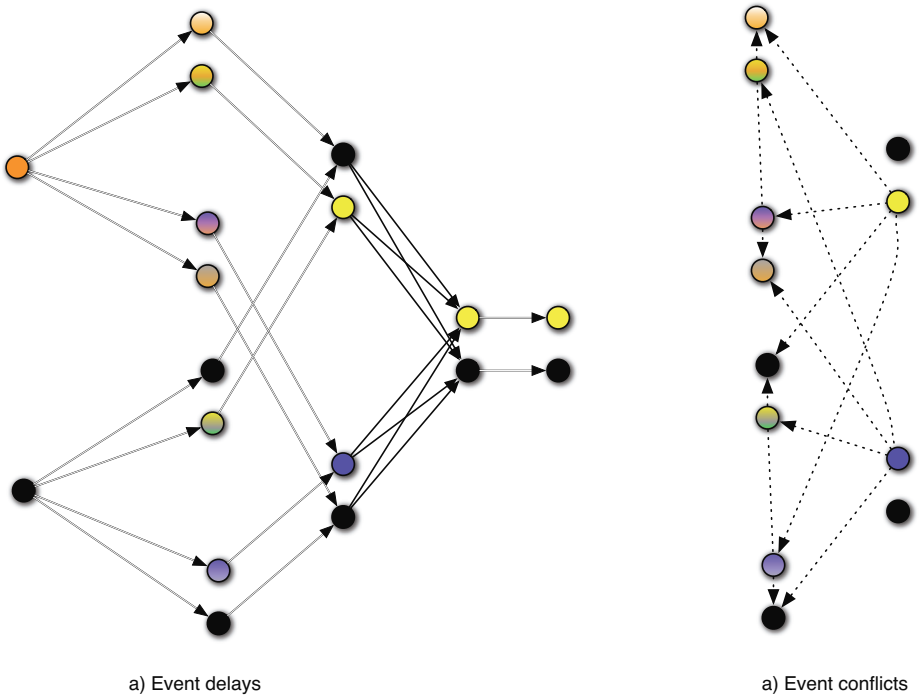

Figure 7.11: Normal Form of the score in Figure 7.10.

7.4.2 Time conditional relations

We upgrade the definition of time conditional relations without loops, introduced in Def. 7.1.2, with the *interpretation of the condition*. This is useful not only for loops, but for conditional branching in general. There are two possible values for the interpretation of the condition in a relation $\langle p, \Delta, c, q \rangle$. (1) *when* means that if c holds, the control jumps from p to q. (2) *unless* means that if c does not hold or its value cannot be deduced from the environment; for instance, when the condition is $a > 0$ and the environment has no information on a, the control jumps from p to q.

7.4.3 Interactive scores

We recall that in conditional branching scores without loops all branches have the same duration; however, that could be inconsistent with the definition of loops because the duration of a loop is always $[0, \infty)$. If a branch has a loop, and we cannot predict the duration of the loop, both branches have the same set of possible durations.

Example 7.4.1. Figure 7.12 describes a score with a loop. We assume that each time unit lasts one second. During the execution, the system plays a silence of one second. After the silence, it plays the sound b during three seconds and simultaneously it turns on the lights d for one second. After the sound b, it plays a silence of one second, then it plays video c. If the variable $finish$ becomes true, it ends the scenario after playing the video c; otherwise,

it jumps back to the beginning of the first silence after playing the video c. Object a is composed by points sp_a and ep_a, its children are b, c and d and its local variable is $finish$. The time conditional relation between points sp_a and sp_b, is labeled with a condition `true`, its interpretation is *when* and its duration is one second.

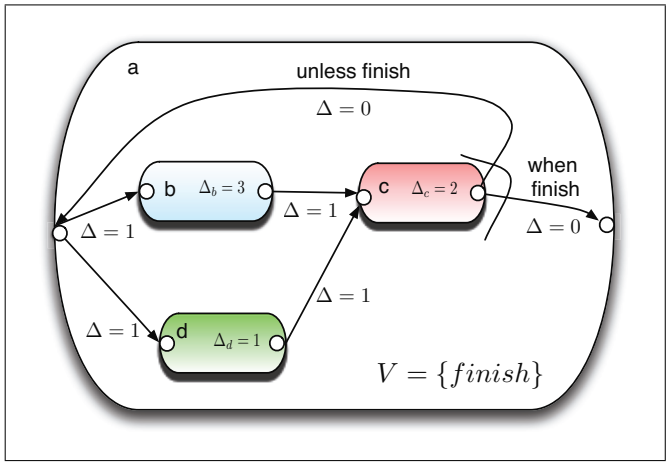

Figure 7.12: A score with a user-controlled loop.

7.5 Towards an Operational Semantics with Loops

Although we presented operational semantics and a prototype in [Toro 2010b, Toro 2010c], such a semantics is based on a recursive structural definition of temporal objects, Allen's relations, has no normal form, has no dispatchable form and allows branches with different durations, thus it needs to be revisited. In addition, it does not take into account multiple instances of a temporal object nor the strategies to stop a loop preserving a content coherence. We believe such issues need to be studied in detail before providing a complete and correct operational semantics. In what follows, we discuss possible behaviors to deal with multiple instances of a temporal object and strategies to stop a loop. Finally, we study a multimedia installation and we show how the behavior of that installation can be modeled in interactive scores.

7.5.1 Multiple instances of an object

Some scenarios require multiple instances of the same temporal object executing concurrently. This must be treated in a special way because not all the processes associated to a temporal object accept that sort of *polyphony*. Consider, for instance, a temporal object that controls a curtain in a theater performance, there may not be a defined behavior for more than one instance of the object at the same time. We propose in Figure 7.13 four behaviors to manage concurrent instances of the same temporal object: splitting them, delaying them, cancelling them or allowing them. This behavior must be selected by the designer of the scenario according to the nature of the temporal object.

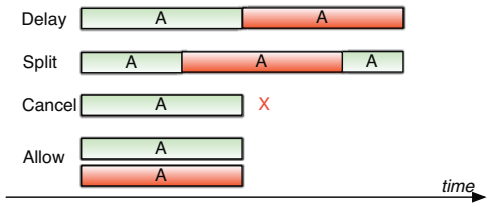

Figure 7.13: Multiple instances of a temporal object simultaneously. The top-colored box is one instance of a and the bottom-colored box is the other instance of a.

7.5.2 Stoping an object including a fade out

In interactive scores, it is common to temporally relate an interactive object to the end point of a static object. When this happens, we say that the interactive object stops the execution of the static object. This raises several problems in terms of content coherence; for instance, when we are playing a sound or a melody that cannot be stopped abruptly. This problem gets worse when we have loops, which also can be stopped by interactive objects.

We believe that defining intermediate points in a temporal object, besides its start and end points, will allow the designer to foresee a special behavior that will be executed when the user wants to interactively end the temporal object during execution: We call this behavior a *fade out*. In Figure 7.14, we show two possible cases to end a temporal object. In Figure 7.15, we explain how an object with a fade-out behavior can be modeled by the means of conditional branching. Finally, in Figure 7.16 we extend this approach to objects that have children. These behaviors need to be studied before defining an operational semantics.

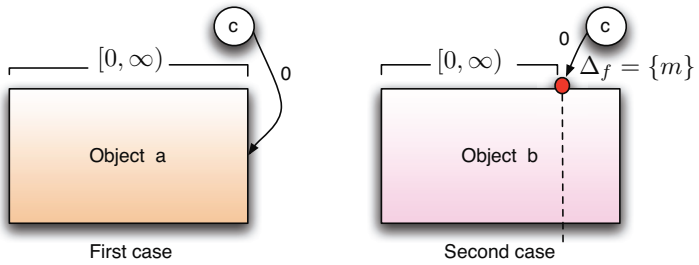

Figure 7.14: Two different cases, combining interactive objects and intermediate exit points. In the first case, the duration of the object is $[0,\infty)$ when there is an interactive point at the end. In the second case, it is possible to choose between stopping the execution of an object playing a short fade out or executing the object as expected. Colored point is an intermediate exit point, c is an interactive object, $\Delta_f = \{m\}$ represents the duration of the fade-out behavior.

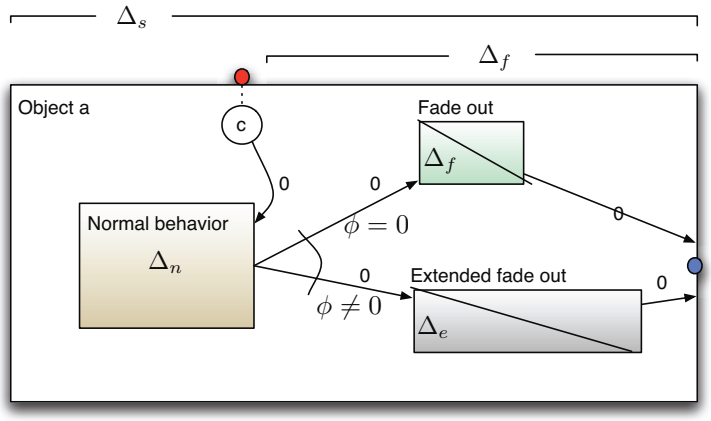

Figure 7.15: An object with a intermediate exit point. An intermediate exit point is controlled by a condition, thus it can be executed or not. The object has a total duration of Δ_s, which is usually an interval. Its minimum duration is given by $\Delta_n = \{n\}$, which is the duration of its normal behavior. After the normal behavior, a extended fade out is executed unless the exit point is launched. If the exit point is launched, a short fade-out is executed. The final point of the object a waits for the first object that transfers the control to it. Internally, the colored exit point is connected to an interactive object c.

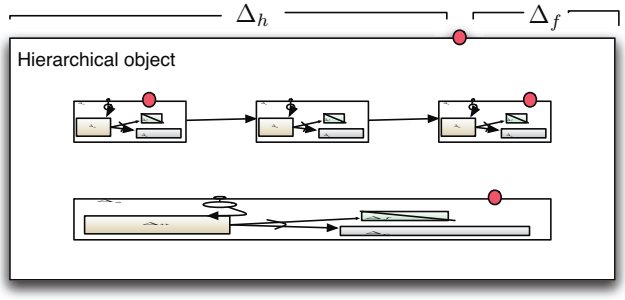

Figure 7.16: An object that contains other objects and has an intermediate exit point. An object that has children does not execute a process of its own: It serves as a container for its children. Each object inside the object may have an intermediate exit point. When the intermediate exit point of the father is called, the intermediate exit points of its children being executed are called. The execution of its children is abruptly ended if they do not haven an intermediate exit point. The fade-out duration of the father Δ_f has to be bigger than all the fade-out durations of its children.

7.5.3 Study Case: Mariona

Mariona[2] (french acronym for automatic machine with memory, iconography, oniric, narrative and acoustic) is a multimedia installation capable to generate images, analyze the movements of the users and produce sounds. Its control is described by three temporal objects that interact concurrently: *global*, *speed* and *aléatoire* (random). In its implementation, these objects are written in Max/MSP.

Figure 7.17 describes the main temporal objects and object *bug*. Figure 7.18 describes some temporal object contained in *global*. The notation used in the figures is from the score provided by it designer, Pol Perez. In Mariona –as we may see in figures 7.17 and 7.18–, there is choice, trans-hierarchic relations, random durations, and finite and infinite loops. In what follows, we give some examples about the relation between Mariona and the models of conditional branching interactive scores with loops.

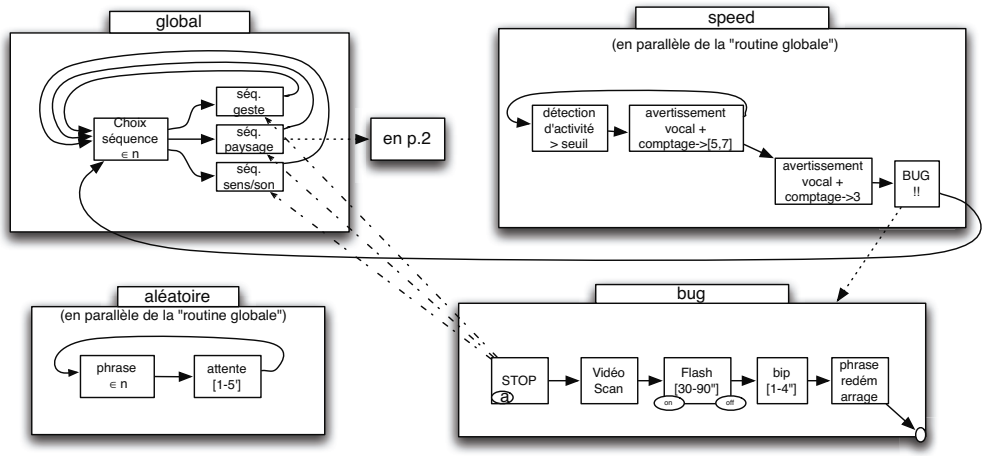

Figure 7.17: Score of Mariona provided by its designer. We present temporal objects *global*, *speed*, *aléatoire* and *bug*.

Choice is the main phenomenon that we can see in Mariona; for instance, in the *global* object, the system makes a choice among three different objects: *séq. geste, séq. paysage, séq. sens/son*. In the *aléatoire* (random) temporal object, we may observe loops and a random delay from one to five seconds (*attente [1'-5']*). There is also a random duration in the *flash* and *bip* (beep) objects, which are inside the *bug* object. Note than in interactive scores, we cannot model random durations.

[2] http://www.gmea.net/activite/creation/2007_2008/pPerez.htm

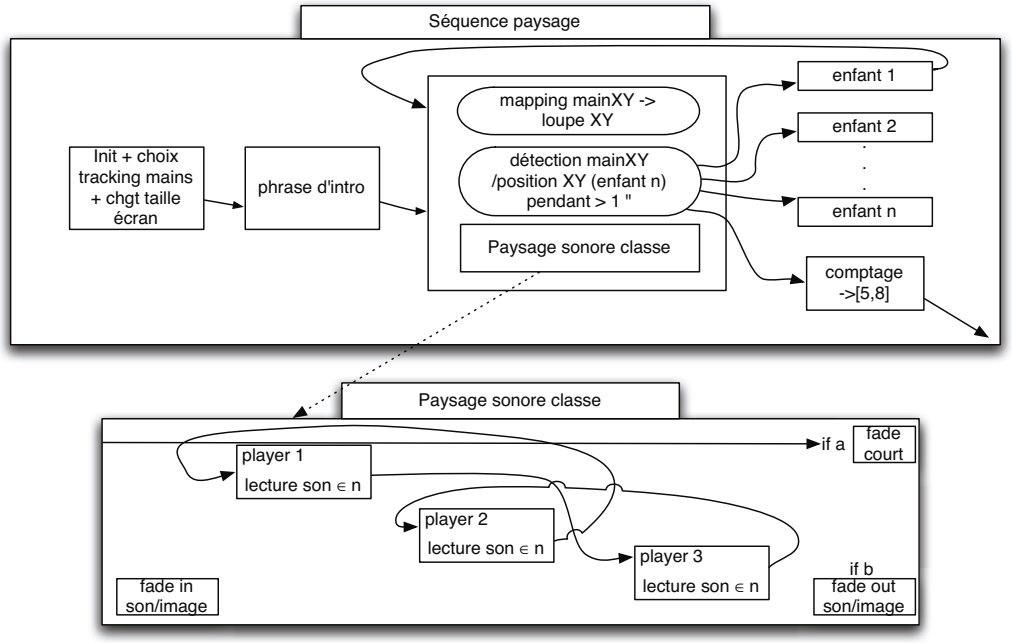

Figure 7.18: Score of Mariona provided by the author. We present temporal objects *séquence paysage* and *paysage sonore classe* temporal objects.

7.6 Summary and Discussion

We presented two models of conditional branching interactive scores. The first model does not include loops. We defined a structural definition based on the idea of constraint system. Afterwards, we gave event structures semantics and we argued that the hierarchical scores, described in Chapter 6, can be encoded into this new model, and, additionally, it can express choice and conditions. In the first models of conditional branching published in [Toro 2010b, Toro 2010c], we allowed branches starting in the same point have different durations. We left aside such an approach because it makes many scores incoherent and unplayable.

The second model of conditional branching includes loops. We slightly modified the structural definition of the score to allow loops. We do not define event structures semantics for the conditional branching model with loops. Such a semantics is not easily defined because events can only be executed once; therefore, to define semantics we need an infinite number of events, as proposed in [Langerak 1992].

We gave some ideas on how to define operational semantics for both models. We believe that ntcc is appropriate to define operational semantics. An advantage of ntcc with respect to languages such as Max or Pure Data (Pd) is that conditions can be declaratively represented by the means of constraints. Complex conditions, in particular those with an unknown number of parameters, are difficult to model in Max or Pd. To model generic

conditions in Max or Pd, we would have to define each condition either in a new patch or in a predefined library. In the Petri nets, we would have to define a net for each condition.

CHAPTER 8

Scores with Signal Processing

Contents

In this chapter, we present an extension of interactive scores with signal processing. In particular, we are interested in modeling the macroform of multimedia content and the microform of sound. We are not interested in the microstructure of image nor video.

To have control over the microform of sound during performance, we need two new types of relations: *high-precision temporal relations* that can be expressed in the order of samples, milliseconds and microseconds, and *dataflow relations* to connect the sound output of a temporal object to the input of another.

To be able to control the microform and the macroform during performance, we use a combination of two computational paradigms: (1) a reactive system based on discrete time units and (2) a synchronous language. As we stated previously in Part I, we chose the non-deterministic timed concurrent constraint (ntcc) calculus to model the reactive system. Ntcc is in charge of the macroform of multimedia content: synchronizing the start and end points of temporal objects that are related with a partial order defined by the macro-temporal relations, as we defined in Chapter 5. In addition, we use Faust for audio processing. Faust is in charge of microdelays in the order of samples, milliseconds and microseconds that go beyond the time scale that the real-time interpreter of ntcc, Ntccrt, can manage. Faust is also in charge of dataflow, connecting the input and output of each Faust process associated to a temporal object.

In this chapter, as we stated in Chapter 1, we propose a system to declare temporal constraints among multimedia processes that aims to ensure all temporal relations between events in the macroform and the microform of the scenario. We present results of the execution of a score modeled with this extension in Chapter 9. In what follows we explain how ntcc and Faust can interoperate together.

Faust and ntcc interoperability. Ntcc can send constraints to Faust, but, currently, Faust cannot send information to ntcc because it requires subsampling. The constraints sent from ntcc cannot be partial information, such as $pitch > 3$ or $gain < 1$ because such information cannot be processed by Faust. Constraints must be equalities of the form $variable = constant$. Using Pure Data, we can communicate those values from ntcc to Faust by the means of number fields. Note that ntcc cannot take decisions based on the samples of an audio signal because ntcc is not mean to handle audio signals, and Faust cannot take decision based on absence of information or partial information. Therefore, they complement each other's strengths.

Example 8.0.1. We present a possible interoperation between Faust and ntcc in Figure 8.1. On the one hand, ntcc can receive a user input each discrete time unit. If the value of the input is 1, ntcc communicates to Faust that the gain is 10; otherwise, if the user gives no input, ntcc communicates to Faust that the gain is $1/10$. On the other hand, Faust takes an audio signal an multiplies by the gain value given by ntcc. In addition, Faust multiplies the signal by 2 if the current sample of the audio input is less than 3.

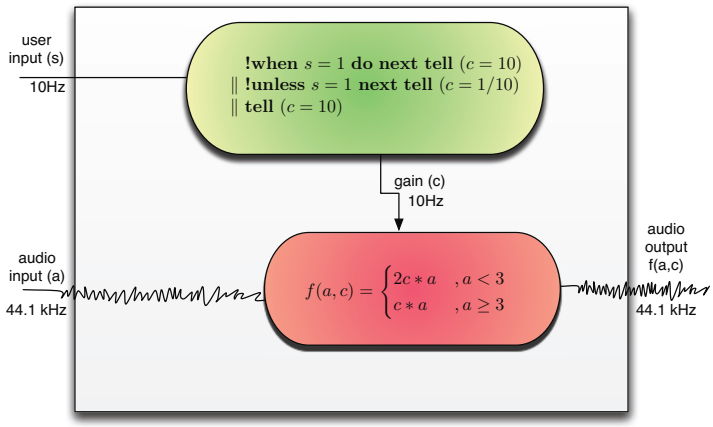

Figure 8.1: Example of interaction between Faust and ntcc.

As we argued in Chapter 4, *graphical user interface* (GUI) objects in Faust can be defined in the same way as other signals. Therefore, we can control buttons, check boxes and integer inputs –originally designed for users– from Ntccrt.

Structure of the Chapter. In what follows we describe the structural definition of a score: temporal objects, temporal relations and dataflow relations. Afterwards, we present several applications of this extension of interactive scores, already presented in [Toro 2012a]. Finally, we present a summary and a discussion.

8.1 Structural Definition

Scenarios in interactive scores are represented by *temporal objects*, *temporal relations for micro and macro controls*, and *dataflow relations*.

8.1.1 Temporal Objects

Temporal objects can be triggered by interactive objects (usually launched by the user) and several temporal objects can be active simultaneously. The duration of a temporal object is given by an interval of natural numbers (which may include ∞), and a temporal object which may contain other temporal objects, as defined in Chapter 6.

In this extension, objects that do not have children may have a sound synthesis process. A process is a Faust program that is active during the execution of the object. These processes include at least one input signal: to control its start and end.

8.1.2 Temporal Relations

In this chapter, we consider temporal relations, as the ones described in Chapter 5. The relations between the start or end of two temporal objects are labeled with an interval of integers that represents the possible duration between the two points. Using ∞ in such intervals, it is possible to represent the relations $<, >, \leq, \geq$ and $=$ with their usual interpretation.

In this chapter, we also include *high-precision temporal relations*. Such a new type of temporal relations between sound objects are meant to have higher precision and they are controlled by Faust. High-precision temporal relations are labeled by an integer n, where n represents, for instance, a number of samples or microseconds. Nonetheless, we can also use these relations to represent durations in seconds. We represent graphically such relations with dashed arrows.

8.1.3 Dataflow Relations

A *dataflow relation* between objects a and b means that the audio outputs of a are connected to the audio inputs of b. If a has more outputs than b inputs, they are merged; if a has less outputs than b, they are split. The control inputs of a Faust subprocess are connected automatically depending on the dataflow, and the micro and macro controls.

Example 8.1.1. As an example, the reader may see the *dataflow view* of a scenario in Figure 8.2. A sound is recorded by the acquisition object, then the stream is passed to a delay object, and then is passed to a filter that adds gain. Finally, the stream is passed to an object that sends two copies of the stream to the output.

8.1.4 Interactive scores

A score is represented by two views of the scenario: (1) the *dataflow view* to describe the dataflow relations among objects, and (2) the *temporal view* to describe the macro-temporal relations and the high-precision temporal relations (represented by dashed-lines).

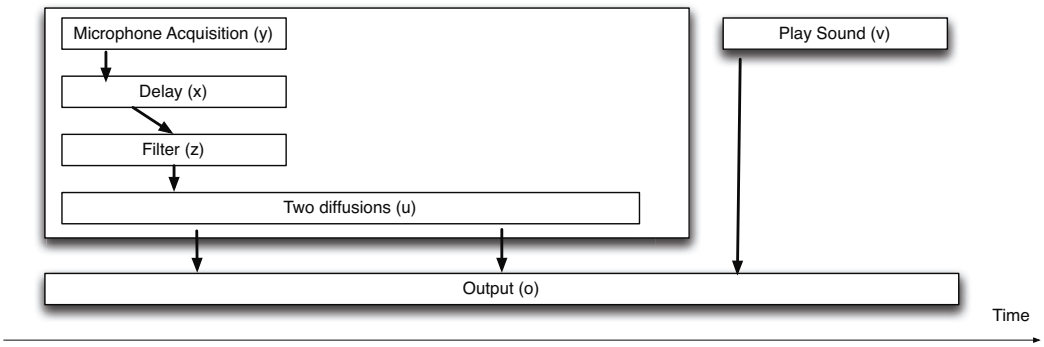

Figure 8.2: *Dataflow view* of a scenario. Thick arrows represent the flow of data through time from one temporal object to another.

In what follows we model an arpeggio based on *Karplus-Strong*. Karplus-Strong is an algorithm to generate metallic plucked-string sounds, which can be described in a few lines of Faust. In the Faust implementation of Karplus, presented by Orlarey *et al.* in [Orlarey 2004], a button triggers the sound. In this chapter, we connect such a button to a control signal sent from ntcc to the Faust plugin at the beginning of the temporal object. We also add another button to stop the sound of the string. In Pure Data (Pd), such buttons can be represented by *bang* or *toggle* objects that send messages to the plugin. We use Pd for simplicity, but Pd is not required to integrate ntcc with Faust.

Example 8.1.2. Figure 8.3 is a scenario that models an arpeggio of three strings using Karplus-Strong. The dataflow is simple: all audio outputs are sent to a single stereo output. There are two types of temporal relations: some labeled with intervals in the order of seconds that will be handled by ntcc, and high precision ones in the order of samples that will be handled by Faust.

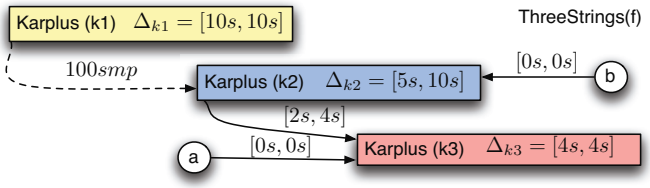

Figure 8.3: An arpeggio with three strings. The durations in the temporal relations are labeled with seconds (s) and in the durations in the high-precision temporal relations with samples (smp). Circles are interactive objects.

As we explain in Chapter 6, the temporal constraints of the scenario are obtained from the duration of each temporal object, the hierarchy and from the temporal relations. For each temporal object, we add two the constraints: (1) "the start time of the object plus its duration is equal to the end time of the object" and (2) "the object starts after its father and

ends before its father". For each temporal relation, we add the constraint "the time of the first point plus the duration in the relation is the time of the second point".

Example 8.1.3. Figure 8.4 is the event structures semantics of the scenario in Figure 8.3. For simplicity, in Figure 8.5, two points linked with a zero duration are represented as a single point in normal form, as introduced in Chapter 6. The ntcc model is parametric on the dispatchable normal form of the score. For that reason, high precision relations are represented as zero durations in the normal form because they are controlled by Faust and not by ntcc, even if the duration in the relations is given in seconds.

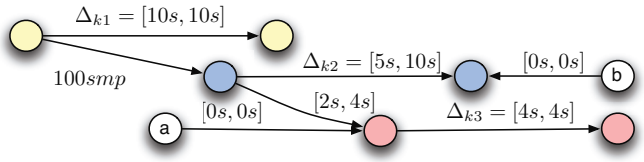

Figure 8.4: The event structures semantics of the scenario in Figure 8.3. The durations in the temporal relations are labeled with seconds (s) and in the high precision temporal relations with samples (smp). Circles are interactive objects.

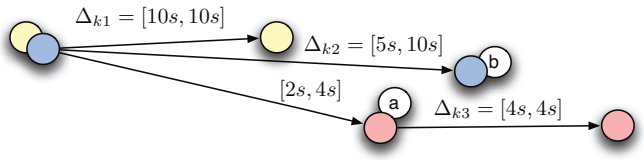

Figure 8.5: The normal form of the event structures semantics of the scenario in Figure 8.4. Intervals are given in seconds (s).

Example 8.1.4. Figure 8.6 is the block diagram for the Faust program in charge of sound processing. The inputs are controlled by ntcc. For simplicity, to avoid downsampling, control signals are replaced by Faust GUI buttons. The audio output of each Karplus block is added together.

8.2 Applications

In what follows we present some multimedia scenarios modeled in the extended formalism of interactive scores, presented in [Toro 2012a].

8.2.1 The macro structure of an arpeggio sequence

In Figure 8.7, we duplicate an arpeggio three times. The macroform is respected: The duration of each arpeggio is 10 seconds, but the start date and the durations of some notes

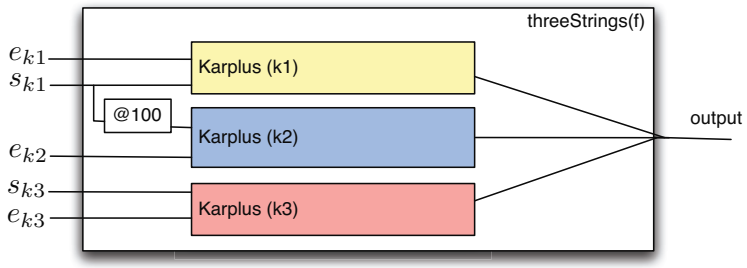

Figure 8.6: Block diagram representing the Faust process in charge of signal processing and the micro controls of the sound processors of the scenario. Signal processor @100 adds a delay of 100 samples to the signal s_{k1} (the start of the first string).

can be controlled by the user with the freedom described in Figure 8.3.

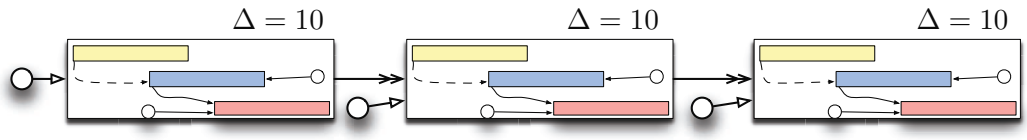

Figure 8.7: Three repetitions of a temporal object containing an arpeggio of three strings (described in Figure 8.3). The double-headed arrow represents an inequality ($\leq$) and a white-headed arrow represents an equality relation ($=$).

8.2.2 An arpeggio without "clicks"

There is a problem with the example in Figure 8.3: Interrupting abruptly the execution of the Karplus-Strong DSP causes perceptible "clicks". A solution to this problem is to gradually decrease the volume (or increase the attenuation parameter) before stoping the DSP, as shown in Figure 8.8. The value of 0.5 seconds is arbitrary, but it is fixed in the scenario, allowing us to know precisely the macroform of the scenario; for instance, its total duration.

8.2.3 Changing the sound source perception

Small delays between the start of two temporal objects are usually not perceptible; however, in some cases –such as the example in Figure 8.9–, a small delay of 500 μs[1] between a sound played on the left channel and the same sound played on the right channel can change the way on which we perceive the sound source[2].

[1] In Faust we represent this delay as 22 samples at 44.1 kHz sampling rate.
[2] http://buschmeier.org/bh/study/soundperception/

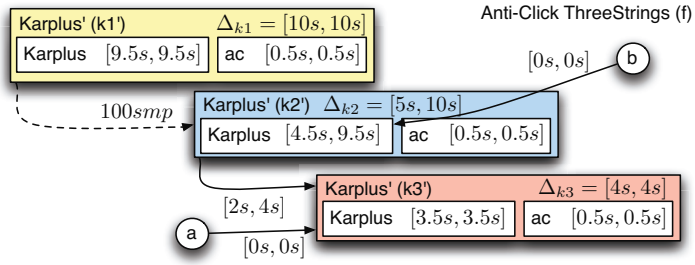

Figure 8.8: A modification of the scenario presented in Figure 8.3 to remove "clicks". The Karplus objects simulate plucked-strings and the *ac* objects change the attenuation parameter of the strings gradually. The macroform is preserved intact.

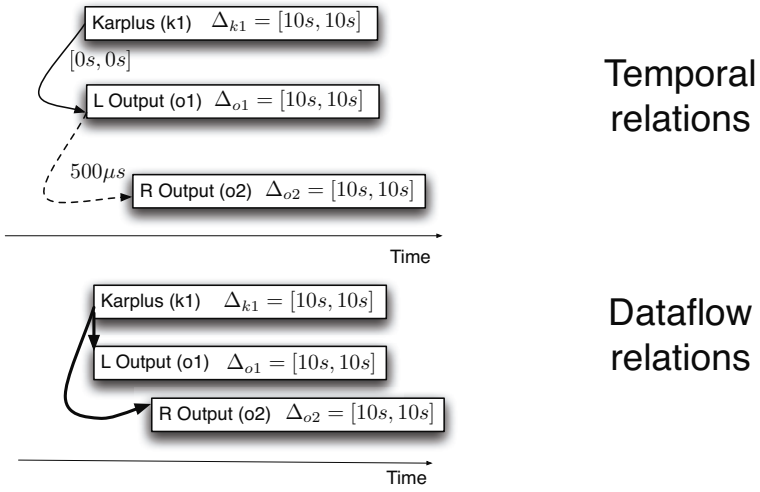

Figure 8.9: A scenario with a high-precision temporal relation. First output is the left channel and second output is the right channel. First view is *temporal relations* and the second view shows the *dataflow relations*.

8.3 Summary and Discussion

In this chapter, we extended the formalism of interactive scores with sound processing and micro controls for sound processors. Time scales are related and available in the same tool. During execution micro controls are managed by Faust and the macro controls by ntcc; however, it is possible to express, for instance, that an object starts 500 microseconds after another and it will end one second before another object.

A perspective of this work is to allow multiple points inside a temporal object; instead, of just start and end points, as usual. Janin has already detailed the advantages of such an approach to model rhythmical structures [Janin 2012].

We also believe that any Faust program could be translated into ntcc based on the results obtained in [Rueda 2005b]. Rueda *et al.* translated the Karplus-Strong Faust program into ntcc. Although it is clear that the execution of a Ntccrt simulation cannot be done at sound processing sampling frequency, such translation could be used to verify properties of correctness of a scenario where ntcc and Faust interact (e.g., playability).

Part III

Implementation

CHAPTER 9

Simulation

Contents

We introduced operational semantics for different models of interactive scores in Part II. To complete our framework, we need to execute and verify properties of those models. In this chapter we focus on execution. A program to simulate the behavior of a score must be able to interact with a user in real-time. Since the operational semantics are given in the non-deterministic timed concurrent constraint (ntcc) calculus, we need an interpreter for ntcc capable of real-time multimedia interaction.

There are three interpreters for ntcc: *Lman* [Muñoz 2004] used as a framework to program Lego^{TM} robots, *NtccSim*[1] used to model and verify properties of biological systems, and *Rueda's interpreter* [Rueda 2006] for multimedia interaction. Nonetheless, they are not suitable for real-time interaction because they are not able to interact with the users without letting them experience noticeable delays in the interaction.

Ntccrt[2] is a framework to execute ntcc models developed by Toro *et al.*, a real-time capable interpreter for ntcc [Toro 2009]. In what follows we describe Ntccrt, then we show how to execute the conditional branching extension and the signal processing extension of interactive scores. Afterwards, we show some quantitative results, and we give a summary and discussion.

9.1 Ntccrt: A Real-Time Capable Interpreter for ntcc

In Ntccrt, we can write a ntcc model on either Lisp, Openmusic or C++. To execute a ntcc model, Ntccrt programs can be compiled into stand-alone programs or as binary plugins (known as external objects) for Pure Data (Pd) or Max/MSP, as shown in Figure 9.1.

[1] http://avispa.javerianacali.edu.co
[2] ntccrt.sourceforge.net/

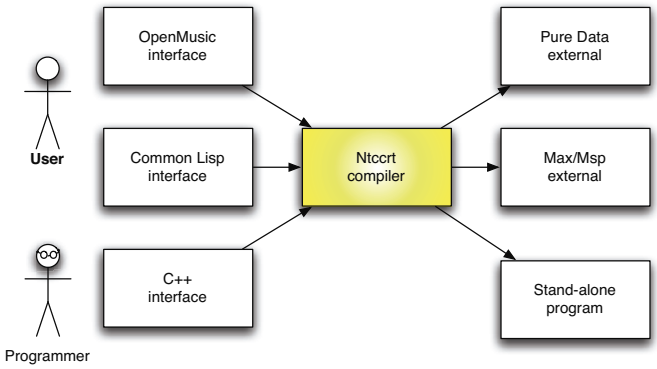

Figure 9.1: Interfaces to program and execute ntcc models in Ntccrt [Toro 2009].

We can use the message passing API provided by Pd and Max to communicate any object with the Ntccrt plugin. We can also control all the available objects for audio and video processing defined in those languages using Ntccrt. Ntccrt can also control a Faust plugin compiled for Pd or Max. Ntccrt uses *Gecode* as its constraint solving library because Gecode [Tack 2009] was carefully designed to efficiently support the finite domain constraint system. Ntccrt relies on propagation of finite domain constraints.

9.1.1 Implementation of Ntccrt

Ntcc interpreters work on a simple, but powerful principle: To simulate ntcc, they do not have to solve a *constraint satisfaction problem* each time unit; instead, using a constraint solving library, ntcc interpreters use *constraint propagation* to calculate the output of each time unit. Propagators are narrowing operators reducing the set of possible values for some variables. A library for constraint propagation executes propagators in an event-driven manner, and chooses the order to call the propagators according to several criteria such as time complexity, type of propagation, variables involved, among others.

Example 9.1.1. Process **tell** $(a = b)$ can be represented by an equality propagator $a = b$, which removes from the domain of a and b values incoherent with the equality constraint.

Ntccrt was designed for multimedia interaction and it is capable of real-time interaction. It is different from its predecessors because it is based on a different theoretical model: an event-driven model instead of concurrency by threads. The problem with threads is that constraint solving libraries are not usually *thread-safe*, thus in order to use threads, it is required to compute an stable constraint store each time an agent wants to know if a constraint can be deduced from the store, making thread implementations inappropriate for real-time interaction.

To represent *when* processes in Ntccrt as propagators, we need constraint deduction. A simplified constraint deduction is easily modeled using *constraint reification*.

Definition 9.1.2 (Constraint Reification)**.** *"The reification of a constraint c with respect to a variable x is the constraint* $(c \leftrightarrow x = 1)$, $x \in \{0,1\}$, *where it is assumed that x does not*

occur free in c. The operational semantics of a propagator for the reification of a constraint c with respect to x is given by the following rules: If the constraint store entails $x = 1$, the propagator for the reification reduces to a propagator for c. If the constraint store entails $x = 0$, the propagator for the reification reduces to a propagator for c. If a propagator for c would realize that the constraint store entails c, the propagator for the reification tells $x = 1$ and ceases to exist. If a propagator for c would realize that the constraint store is inconsistent with c, the propagator for the reification tells $x = 0$ and ceases to exist." [3]

A process **when** c **do** P is represented by two propagators: (1) $c \leftrightarrow b$, a reified propagator for the constraint c; and (2) the *when propagator*, a propagator defined in Ntccrt. This propagator checks the value of b. If the value of b is `true`, it calls the propagator that represents P; otherwise, it does not take any action. If the condition b is `false`, its work is over. This propagator is equivalent to the *synchronized execution propagator* introduced in Gecode 3[4]. Therefore, using a library for constraint propagation, the ordering on how to execute ntcc processes in managed by the library.

Representing *when* processes as propagators works well for models in which propagation is enough to deduce a constraint from another. In the general case, constraint deduction requires constraint search, which is well-known to be a NP problem.

In Ntccrt, non-determinism is solved with a pseudo-random choice. In Ntccrt, tuples $\langle c_i, P_i \rangle$, in a process $\sum_{i \in I}$ **when** c_i **do** P_i, are re-arranged pseudo-randomly, using a C++ random number generator library. The constraint guards are represented by reified propagators $c_i \leftrightarrow b_i$, where b_i is a boolean variable. The *when propagator* is extended. The new propagator is called the *sum propagator*. This propagator calls the propagator that represents P_i when the first constraint c_i is deduced. If all constraints c_i are deduced as `false`, the work of this propagator is over.

Ntccrt is written in C++ and uses *Flext*[5] to generate binary plugins for either Max or Pd, and Gecode for constraint propagation and concurrency control. Although Gecode was designed to solve combinatorial problems, Toro found out in [Toro 2008] that by writing the non-deterministic sum process as a propagator, Gecode can manage all the concurrency needed to represent ntcc.

9.2 Simulation of Interactive Scores

In what follows we explain how to simulate the execution of interactive scores with conditional branching and interactive scores with signal processing.

9.2.1 Simulation of time conditional branching scores

We implemented the example in Figure 7.12 using Ntccrt and Pd, as shown in Figure 9.2. We replaced the *User* process with a user input that assigns the variable *finish*. We

[3] `http://www.ps.uni-saarland.de/alice/manual/cptutorial/node43.html#SECTION00011110000000000000`

[4] `http://www.gecode.org/doc-latest/reference/group__TaskModelIntExec.html`

[5] http://puredata.info/Members/thomas/flext/

generated a Ntccrt plugin for Pd based on a ntcc model. This results are fully detailed in [Toro 2010b].

The plugin has two inputs: one for the clock ticks and one for the value of *finish*. The input for the clock ticks can be connected to a *metronome* object to force time units to have a fixed duration during the performance. The Ntccrt plugin outputs a boolean value for each point, indicating whether it is active or not. Using such values, users can control the start and end of *b*, *c* and *d*, which are programs defined in Pd.

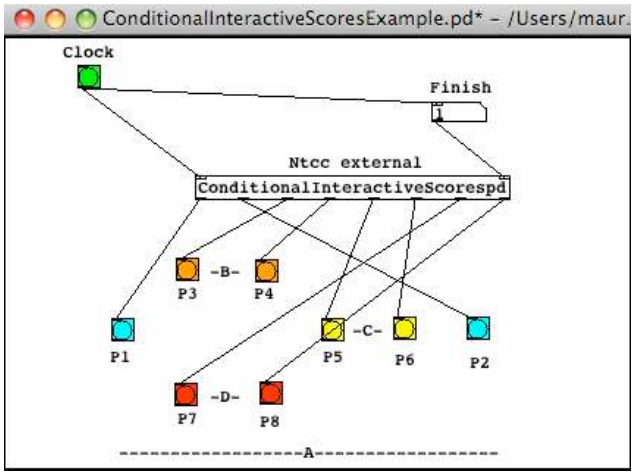

Figure 9.2: Executing the Score in Figure 7.12 in Pure Data.

9.2.2 Simulation of scores with signal processing

We recall from Chapter 1 that a problem may emerge at runtime when we want to synchronize the execution of multiple multimedia processes under high CPU load because a system interruption at this point can often lead to desynchronization. Usually, these eventualities are not considered by developers of multimedia interaction tools, as the quality of systems is evaluated according to an average performance. Nonetheless, during performance, it is desired that the system works well even under high CPU load.

The novelty of our extension of interactive scores is using the constraints sent from ntcc to control Faust. We tested our examples in Pd, although they could also be compiled for Max or as a standalone program since both Faust and ntcc can be translated into C++ and Max. In fact, the final goal of our research is to develop a standalone program.

Using Faust and Ntccrt, we achieved an efficient and real-time capable performance the score in Figure 8.3. In what follows, we present the results on the execution of the conditional branching model and the signal processing model. Figure 9.3 is the Pure Data patch representing the scenario.

9.2.3 Results

In what follows we present quantitative results on the simulation of scores with conditional branching and scores with signal processing.

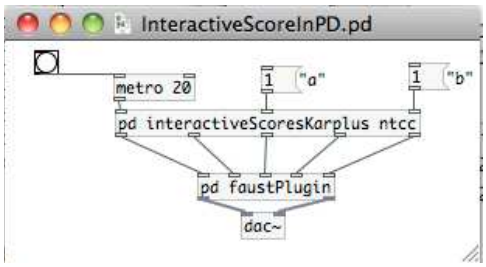

Figure 9.3: Pure Data patch representing the scenario in Figure 8.3. The Ntccrt plugin has only five outputs because the start of the second Karplus-Strong object ($k2$) is controlled directly from Faust. Interactive objects are represented by messages. In this example, the internal clock of Ntccrt is controlled by a Pd *metronome* object with a period of 20ms.

Conditional branching. Performance of the score in Figure 7.12 poses no problem for real-time interaction; however, it is such a small example, thus we need a stress test. We built automatically a conditional branching interactive score with a number of points and temporal relations in the order of 2^n. We tested scores with different values of n: from two to ten. The score is exemplified in Figure 9.4. We ran each score 100 times as a stand-alone program. The duration of each time unit is determined by the time taken by Ntccrt to calculate the output, not by an external clock. The tests were performed on an iMac 2.6 GHz with 2 Gb of RAM under Mac OS 10.5.7. Ntccrt was compiled with GCC 4.2 and liked to Gecode 3.2.2.

Pachet *et al.*, authors of the *Continuator* [Pachet 2002], argued that a multimedia interaction system with a response time less than 30 ms is able to interact in real-time even with a very fast guitar jazz player. Therefore, our results, summarized in Figure 9.5, are acceptable for real-time interaction with a guitarist for up to 1000 points (around 500 temporal objects). We conjecture that a response time of 20 ms is appropriate to interact with even a very fast percussionist. In that case, we can have up to 400 temporal objects. These results were already published in [Toro 2010b].

Signal processing. In addition to the tests with conditional branching, we implemented the score in Figure 8.3. We tested three implementations of the Karplus-Strong in Pd: one from Colin Barry[6] that uses an instruction to define blocks of one sample (object *block~* 1), one from Johannes Kreidler[7] that uses one-sample delays (object *z~* 1), and one from Albert Gräf using a Faust plugin generated with Pd-Faust[8]. The interactive points are launched automatically (at the latest possible time). These results were published in [Toro 2012a].

For each test, we played each arpeggio four times with a CPU load of 3% and four times with a load of 85%. We repeated each test ten times. The tests were performed in a 3.06 GHz Intel Core i3 processor on an iMac with a RAM memory of 4 Gb 1333 MHz DDR3, under Mac OS 10.6.8, using Pure Data extended 0.42 and Faust 0.9. To increase the

[6] www.loomer.co.uk
[7] www.pd-tutorial.com
[8] http://docs.pure-lang.googlecode.com/hg/faust2pd.html

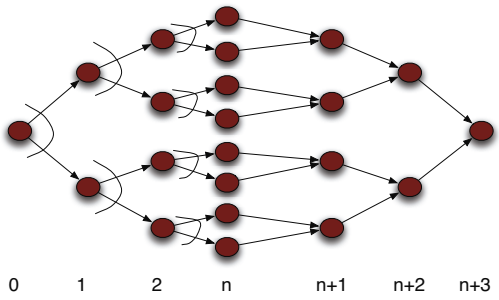

Figure 9.4: A scalable-size score with $3.2^n - 2$ points. In this case, $n = 8$. Points are presented by circles. Arrows represent temporal relations labeled by $[0, \infty)$ and `true` conditions. Arcs crossing two arrows represent mutually exclusive choices. These choices are controlled by the computer because conditions are always `true`.

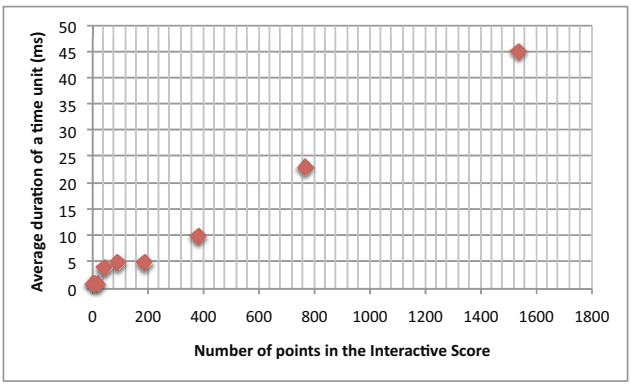

Figure 9.5: Performance of the simulation fo the score in Figure 9.4.

CPU load, we ran several video processing operations from the *graphics environment for multimedia* (GEM) plugin for Pd. The CPU load values are approximatively and they were obtained using Mac OS X's *activity monitor*.

We calculated the average relative jitter of the micro-temporal structure of the scenario: the average time difference between the expected starting time of each string, with respect to the first string of the arpeggio, and the time obtained during execution. The average relative jitter using Faust is 500 μs with both a CPU load of 3% and 85%; on the contrary, the implementation from Colin Barry has a jitter of 7991 ms with a CPU load of 85% and the implementation from Johannes Kreidler has a jitter of 9231 ms with a CPU load of 85%. These values are very big and make the listening of the arpeggio incomprehensible. The average relative jitter was calculated using Matlab.

The Pd implementations of Karplus-Strong have also a limitation for high frequencies: They work well until 2000 Hz and Faust works well until 3000 Hz. Although this last result is the perception of the authors of [Toro 2012a], they argue that the upper fundamental frequency limit may be due to the "chunk-sized" buffer delay in the feedback loop in Pd.

9.3 Summary and Discussion

In this chapter, we argued that Ntccrt is the most appropriate interpreter for ntcc to run our models because it is real-time capable. We explained Ntccrt design and implementation, which is based on constraint propagation and uses Gecode as constraint solving library.

We simulated two models, one with conditional branching and one with signal processing. The conditional branching score was a stress test in which we found out that we can run models with up to 500 temporal objects using Ntccrt under low CPU load. Pachet *et al.*, authors of the *Continuator* [Pachet 2002], argued that a multimedia interaction system with a response time less than 30 ms is able to interact in real-time even with a very fast guitar jazz player.

For a signal processing score, we made another stress test in which we found out that the combination of Ntccrt and Faust behaves outstanding under a high CPU load and equivalent programs in Pure Data have a very high average relative jitter, where are using Ntccrt and Faust the jitter is almost imperceptible.

Another advantage of Faust is that the control signals in Faust can be delayed at sample level, whereas it is not possible to add sample delays to messages in Pd. In Pd, we need to delay the audio output instead of the control signals to produce such result. Finally, using Faust, sound processors could be automatically parallelized, improving its performance in many cases [Orlarey 2010].

We believe that using the graphical paradigm provided by Max or Pd is difficult and is time-demanding to synchronize processes depending on complex conditions. On the contrary, using Ntccrt, we can model such systems with a few graphical boxes in OpenMusic or a few lines in Lisp, representing complex conditions by constraints, and we can create a stand-alone program, or a Pd or Max plugin.

Unfortunately, there are two problems when executing interactive scores with Ntccrt. (1) To compute the event structures semantics, its normal dispatchable form by hand is time demanding. This limits the usability of Ntccrt to execute interactive scores to small examples. In the future, such transformations should be done automatically. (2) Ntcc recursive definition cannot be translated directly to Ntccrt because their encoding is based on nested non-deterministic choices hard to simulate. In the future, they should be treated differently; for instance, using variables that can change value from a time unit to another one.

One may argue that although we can synchronize Ntccrt with an external clock (e.g., a metronome object provided by Max or Pd), this does not solve the problem of simulating models when the clock step is shorter than the time necessary to compute a time unit. To solve this problem, Sarria proposed to develop an interpreter for the *real time concurrent constraint (rtcc)* [Sarria 2008] calculus, which is an extension of ntcc capable of modeling time units with fixed duration.

In the future, we also propose to extend the Ntccrt to handle audio files efficiently. Libaudiostream[9] is an audio library, developed at the french research institute *Grame*[10],

[9] http://libaudiostream.sourceforge.net/
[10] http://www.grame.fr/

to manipulate audio resources through the concept of streams using Faust programs. Including Libaudiostream in our framework, it will be possible to design a scenario where a temporal object loads a sound file into memory, Faust filter it, and then, Faust plays the sound at the appropriate time. Precision is guaranteed because the time to load the file and process it is foreknown in the scenario. Currently, we have to rely on third-party programs, such as Pd, to do handle audio files, and to communicate the control signals from Ntccrt to Faust.

CHAPTER 10

File Format

Contents

Applications for designing and executing interactive scores should provide persistence of the scenarios. A good solution is to define a *markup language* because it is simple to understand and easy to parse. In fact, there are many available tools to parse and process markup language files because they are standardized.

The *standard generalized markup language* (SGML)[1] is an ISO-standard technology for defining generalized markup languages for documents. Markup should be declarative: It should describe a document's structure and other attributes, rather than specify the processing to be performed on it. Markup should be rigorous so that the techniques available for processing databases can be used for processing documents as well. *Extensible markup language* (XML) and *extensible hypertext markup language* (XHTML) are both examples of SGML-based languages[2].

Another example is *music* XML[3], a portable file format to write music scores; and *music markup language* (MML[4])[Steyn 2004], a language to describe audio synthesis, control physical devices and perform live music. Unfortunately, both music XML and MML are limited to sound and cannot be easily extended for multimedia.

Fortunately, there are also SGML formats for multimedia. As an example, *synchronized multimedia integration language* (SMIL)[5] is a *world wide web consortium* recommended XML markup language to describe multimedia presentations. It defines markup for timing, layout, animations, visual transitions, and media embedding, among other things. SMIL allows presenting media items such as text, images, video, audio, links to other SMIL presentations, and files from multiple web servers[6].

[1] SGML is defined in ISO 8879, 1986.
[2] http://en.wikipedia.org/wiki/SGML
[3] http://www.recordare.com/musicxml
[4] http://www.musicmarkup.info/
[5] http://www.w3.org/TR/SMIL/
[6] http://en.wikipedia.org/wiki/Synchronized_Multimedia_Integration_Language

SMIL allows us to represent temporal relations, a hierarchy, and several types of multimedia and interaction; however, its structure is different than an interactive score, which makes it a good candidate to be exported from an interactive score application, but not for the file format of an interactive score itself.

There is a far more complex XML format for multimedia: *hypermedia/time-based structuring language (HyTime)*[7], another markup language based on SGML. HyTime defines a set of hypertext-oriented element types that allows authors to build hypertext and multimedia presentations in a standardized way.

HyTime allows to specify temporal constraints and interactive behavior. In fact, Erfle has shown how to model in Hytime relative time and Allen's relations without disjunction [Erfle 1993]. We recall that Allen's relations are used in Allombert's interactive scores, but they are less expressive than point-to-point relations when they do not contain disjunction.

It is possible to express relative time and Allen's relations in HyTime, but can we use HyTime to describe interactive scores? HyTime does not provide any direct mechanisms to express interaction. There are generic mechanisms that can be used for modeling interaction: Wirag *et al.* describe several interaction types such as start, stop, pause, jump, reverse and selection (choice) [Wirag 1995]. HyTime could be used to translate not only hierarchical interactive scores, but also scores with conditional branching; however, such translation will not be easy, for instance, to reconstruct an interactive score based on a HyTime file because their structure is rather different and HyTime does not allow to express temporal constraints to define a partial order, thus HyTime is not an appropriate file format for interactive scores.

10.1 An Existing File Format for Interactive Scores

Allombert introduced in [Allombert 2008a] a XML file format for interactive scores. This new file format allows to easily encode an interactive score into an XML, thus providing persistence. Allombert's XML file format includes information of the external applications controlled from the interactive score. It also specifies the temporal objects, temporal relations and interactive objects of the score.

Unfortunately, Allombert's XML does not explicitly encode the hierarchy, thus it will be difficult to reconstruct the hierarchy of the score. Objects in Allombert's file format only contain a nominal duration, thus it is not possible to describe the set of possible durations of a temporal object. Finally, the temporal relations are represented by Allen's relations which does not allow to represent quantitative relations between two objects easily; for instance, object a occurs 3 time units after object b. It is neither possible to say "the start of object a is before the end of object b". Finally, it is not explicitly described what fields are required and what fields are optional in the different entities, nor their data types.

Example 10.1.1 (Example of Allombert's format)**.** The following example presents how to encode two temporal objects, called *boxes* in Allombert's format. Boxes include a name, the application that they control, the positions in the cartesian plane to draw the objects and a nominal duration. Boxes also include the interaction points associated to it.

[7] HyTime is defined in ISO/IEC DIS 10744, 1992

```
<boxes>
 <box id="bx1" name="tempobj1" id-application="app1" posy="53"
               start-date="8349" duration="11333" posx="14"
  <start-message>
     /note-1/start
  </start-message>
  <end-message>
     /note-1/end
  </end-message>
  <interaction-points>
     <interaction-point event = "0">
  </interaction-points>
 </box>
 ...
</boxes>
```

To overcome the limitations of Allombert's format, we developed a new SGML file formal on the lines of Allombert's. In what follows we explain how to write a *document type definition* (DTD), then, in section 10.3, we define a new DTD to describe hierarchical interactive scores, and in section 10.4, another DTD to describe interactive scores with conditional branching.

10.2 Document Type Definition

It is possible to define the entities and fields of a SGML using a *document type declaration*, which is is an instruction that associates a particular SGML or XML document (e.g., a web page) with a *document type definition* (DTD); for instance, the formal definition of a particular version of HTML.

A DTD defines the structure of a document. The *doctype* defines the name of the document type and *elements* are the main building blocks. The elements can have children (other elements contained in them), and we can specify that there is only one occurrence of an element `<!ELEMENT father (child)>`, zero or more occurrences `<!ELEMENT father (child*)>`, one or more occurrences `<!ELEMENT father (child+)>`, or zero or one occurrences `<!ELEMENT father (child?)>`.

The word *attlist* is used to define an extra information about the element. The attributes have a data type: *pcdata* means parsed character data (text that can contain entities and markup), *cdata* is character data that will not be parsed, and *id* is a unique identifier. In addition, we can state that an attribute is mandatory (#REQUIRED) or optional (#IMPLIED).

10.3 File Format for Hierarchical Interactive Scores

In our DTD of an interactive score, a scenario is an entity that contains several applications, points, temporal relations and a single temporal object (the root). The applications are

external programs that are controlled by the interactive scores; for instance, to control video, sound or light. Temporal objects are represented as a hierarchy, thus they are all contained in the root. Temporal relations contain a minimum and maximum duration; for simplicity we do not include an arbitrary set of possible durations.

```
<?xml version="1.0"?>
<!-- By Mauricio TORO -->
<!DOCTYPE SCENARIO [

<!ELEMENT SCENARIO (APPLICATIONS POINTS TO TRS)>
<!ATTLIST SCENARIO Title CDATA #REQUIRED>
<!ATTLIST SCENARIO Composer CDATA #IMPLIED>
<!ATTLIST SCENARIO Date CDATA #IMPLIED>
<!ATTLIST SCENARIO Osc-receiving-port CDATA #REQUIRED>

<!ELEMENT APPLICATIONS (APPLICATION+)>

<!-- An application is an external program controlled by the
interactive score -->
<!ELEMENT APPLICATION>
<!ATTLIST APPLICATION Name ID #REQUIRED>
<!ATTLIST APPLICATION NameString CDATA #IMPLIED>
<!ATTLIST APPLICATION IP CDATA #REQUIRED>
<!ATTLIST APPLICATION PORT CDATA #REQUIRED>

<!ELEMENT POINTS (POINT+)>

<!-- A point is the start or end of a temporal object -->
<!ELEMENT POINT>
<!ATTLIST POINT Name ID #REQUIRED>
<!ATTLIST POINT XPosition CDATA #REQUIRED>
<!ATTLIST POINT YPosition CDATA #REQUIRED>
<!ATTLIST POINT Application-Id CDATA #IMPLIED>
<!ATTLIST POINT Message CDATA #IMPLIED>

<!-- Interactive Objects -->
<!ELEMENT iTO >
<!ATTLIST iTO Name ID #REQUIRED>

<!-- Static Temporal Objects -->
<!ELEMENT TO (CHILDREN?)>
<!ELEMENT CHILDREN (TO+ iTO*)>
<!ATTLIST TO Name ID #REQUIRED>
<!ATTLIST TO NameString CDATA #IMPLIED>
```

```
<!ATTLIST TO NominalStartTime CDATA #IMPLIED>
<!ATTLIST TO NominalDuration CDATA #IMPLIED>
<!ATTLIST TO StartPointID ID #REQUIRED>
<!ATTLIST TO EndPointID ID #REQUIRED>
<!ATTLIST TO MinDuration CDATA #REQUIRED>
<!ATTLIST TO MaxDuration CDATA #REQUIRED>

<!ELEMENT TRS (TR+)>

<!-- Temporal relations -->
<!ELEMENT TR>
<!ATTLIST TR StartPointID ID #REQUIRED>
<!ATTLIST TR EndPointID ID #REQUIRED>
<!ATTLIST TR MinDuration CDATA #REQUIRED>
<!ATTLIST TR MaxDuration CDATA #REQUIRED>


]>
```

Example 10.3.1. In what follows, we present the XML file to describe the example in Figure 6.1. We have two applications: one to control the sound and one to control the lights. Points' actions are either turn on a light, change its color, play a song, stop a song or do nothing.

```
<?xml version="1.0"?>
<!DOCTYPE SCENARIO "SCENARIO.dtd">
<SCENARIO>
 <APPLICATIONS>
    <APPLICATION Name = "lights" NameString = "Scenario front lights"
                 IP = "localhost" Port ="3300"> </APPLICATION>
    <APPLICATION Name = "sound" NameString = "Background song"
                 IP = "localhost" Port ="3301"> </APPLICATION>
 </APPLICATIONS>
  <POINTS>
    <POINT Name = "sc" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "ec" XPosition = "1000" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "sl" XPosition = "100" YPosition = "30"
           Application-Id = "lights" Message ="/turn-on"> </POINT>
    <POINT Name = "el" XPosition = "700" YPosition = "30"
           Application-Id = "lights" Message ="/turn-off"> </POINT>
    <POINT Name = "sr" XPosition = "300" YPosition = "10"
           Application-Id = "lights" Message ="/red-on"> </POINT>
```

```
    <POINT Name = "er" XPosition = "400" YPosition = "10"
           Application-Id = "lights" Message ="/red-off"> </POINT>
    <POINT Name = "sg" XPosition = "380" YPosition = "40"
           Application-Id = "lights" Message ="/green-on"> </POINT>
    <POINT Name = "eg" XPosition = "580" YPosition = "40"
           Application-Id = "lights" Message ="/green-off"> </POINT>
    <POINT Name = "su" XPosition = "380" YPosition = "70"
           Application-Id = "sound" Message ="/sound-on"> </POINT>
    <POINT Name = "eu" XPosition = "580" YPosition = "70"
           Application-Id = "sound" Message ="/sound-off"> </POINT>
 </POINTS>
 <TO Name = "Scenario" StartPointID = "sc" EndPointID = "ec"
     MinDuration = "0" MaxDuration = "100">
   <CHILDREN>
      <TO Name = "Lights" NameString = "Scenario frontal lights"
          StartPointID = "sl" EndPointID = "el"  MinDuration = "0"
          MaxDuration = "70" NominalStartTime = "20" NominalDuration = "60">
        <CHILDREN>
           <TO Name = "Red" StartPointID = "sr" EndPointID = "er"
               MinDuration = "0" "MaxDuration = "30" NominalStartTime = "5"
               NominalDuration = "30" > </TO>
           <TO Name = "Green" StartPointID = "sg" EndPointID = "eg"
               MinDuration = "0" "MaxDuration = "30" NominalStartTime = "25"
               NominalDuration = "25" > </TO>
           <iTO Name = "a">
        </CHILDREN>
      </TO>
      <TO Name = "Sound" NameString = "Background Song" StartPointID = "su"
          EndPointID = "eu" MinDuration = "0" EndDuration = "40"
          NominalStartTime = "35" NominalDuration = "40"> </TO>
      <iTO Name = "b"> </iTO>
      <iTO Name = "d"> </iTO>
   </CHILDREN>
 </TO>
 <TRS>
   <TR StartPointID = "b" EndPointID = "sl" MinDuration = "0"
       MaxDuration = "0"> </TR>
   <TR StartPointID = "sg" EndPoint = "su" MinDuration = "0"
       MaxDuration = "0"> </TR>
   ...
 </TRS>
</SCENARIO>
```

10.4 File Format for Conditional Branching Interactive Scores

In what follows, we extend the format defined for hierarchical interactive scores. We include a new element called *variable* to define the variables of each temporal object and the variables of the score. We also include a condition in the temporal relations and choices among points. A condition is a text field, thus virtually anything, but more semantic rules are needed to specify which conditions are valid in a score. This will depend on the constraint system and variables of the score.

```
<?xml version="1.0"?>
<!-- By Mauricio TORO -->
<!DOCTYPE CBSCENARIO [

<!ELEMENT CBSCENARIO (APPLICATIONS VARIABLES? POINTS TO TCRS CHOICES?)>
<!ATTLIST CBSCENARIO ConstraintSystem CDATA #Required>

<!-- Applications, interactive objects and points are described
     as in hierarchical interactive scores's DTD -->

<!-- Variables -->
<!ELEMENT VARIABLES (VARIABLE+)>
<!ELEMENT VARIABLE>
<!ATTLIST VARIABLE Name ID #Required>

<!-- Temporal Object -->
<!ELEMENT TO (CHILDREN? VARIABLES?)>
<!ELEMENT CHILDREN (TO+ iTO*)>
<!ATTLIST TO Name ID #REQUIRED>
<!ATTLIST TO NameString CDATA #IMPLIED>
<!ATTLIST TO NominalStartTime CDATA #IMPLIED>
<!ATTLIST TO NominalDuration CDATA #IMPLIED>
<!ATTLIST TO StartPointID ID #REQUIRED>
<!ATTLIST TO EndPointID ID #REQUIRED>
<!ATTLIST TO MinDuration CDATA #REQUIRED>
<!ATTLIST TO MaxDuration CDATA #REQUIRED>

<!ELEMENT TRS (TR+)>

<!-- Temporal relations -->
<!ELEMENT TR>
<!ATTLIST TR StartPointID ID #REQUIRED>
<!ATTLIST TR EndPointID ID #REQUIRED>
<!ATTLIST TR Condition CDATA #REQUIRED>
<!ATTLIST TR MinDuration CDATA #REQUIRED>
```

```
<!ATTLIST TR MaxDuration CDATA #REQUIRED>

<!-- Choices -->
<!ELEMENT CHOICES (CHOICE+)>
<!ELEMENT CHOICE>
<!ATTLIST CHOICE PointIDS CDATA #REQUIRED>

]>
```

Example 10.4.1. In what follows, we describe the XML file to describe the example in Figure 7.12. Points are represented in the same way as in the example in Section 10.3. Temporal objects are slightly different: objects may contain variables, as it is the case of the temporal object *main*. The conditions in the time conditional relations are represented by a string. The points that are in mutual conflict are also represented by a string.

```
<?xml version="1.0"?>
<!DOCTYPE SCENARIO "CBSCENARIO.dtd">
<CBSCENARIO>
 <APPLICATIONS>
    <APPLICATION Name = "sound" NameString = "Sound controller"
                 IP = "localhost" Port ="3301"> </APPLICATION>
 </APPLICATIONS>
  <POINTS>
    <POINT Name = "sm" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "em" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "sa" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "ea" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "sb" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "eb" XPosition = "0" YPosition = "50"
           Application-Id = "none" Message ="none"> </POINT>
    <POINT Name = "sch" XPosition = "0" YPosition = "50"
           Application-Id = "sound" Message ="/chorus/play"> </POINT>
    <POINT Name = "ech" XPosition = "0" YPosition = "50"
           Application-Id = "sound" Message ="/chorus/play"> </POINT>
    <POINT Name = "sw" XPosition = "0" YPosition = "50"
           Application-Id = "sound" Message ="/verse2/play"> </POINT>
    <POINT Name = "ew" XPosition = "0" YPosition = "50"
           Application-Id = "sound" Message ="/verse2/play"> </POINT>
    <POINT Name = "sc" XPosition = "0" YPosition = "50"
```

```
            Application-Id = "sound" Message ="/chorus/play"> </POINT>
    <POINT Name = "ec" XPosition = "0" YPosition = "50"
            Application-Id = "sound" Message ="/chorus/play"> </POINT>
    ...
  </POINTS>
  <TO Name = "main" StartPointID = "sm" EndPointID = "em"
      MinDuration = "0" MaxDuration = "10">
   <VARIABLES>
    <VARIABLE Name = "shi"> </VARIABLE>
    <VARIABLE Name = "psi"> </VARIABLE>
    <VARIABLE NAME = "phi"> </VARIABLE>
   </VARIABLES>
   <CHILDREN>
      <TO Name = "a" StartPointID = "sa" EndPointID = "ea"
          MinDuration = "0" MaxDuration = "10">
          <CHILDREN>
             <TO Name = "chorus(ch)" StartPointID = "sch"
                 EndPointID = "ech" MinDuration = "0"
                 MaxDuration = "5"> </TO>
           ...
          </CHILDREN>
      </TO>
      <TO Name = "chorus(c)" StartPointID = "sc" EndPOintID = "ec"
          MinDuration = "0" MaxDuration = "5"> </TO>
      <iTO Name = "d"> </iTO>
   ...
   </CHILDREN>
  </TO>
  <TCRS>
   <TCR StartPointID = "ev" EndPointID = "sa" MinDuration = "0"
        MaxDuration = "0" Condition = "phi>0 & psi <= 5"> </TCR>
   <TCR StartPointID = "ev" EndPointID = "sb" MinDuration = "0"
        MaxDuration = "0" Condition = "shi>0 & psi <= 3"> </TCR>
   <TCR StartPointID = "sa" EndPointID = "sch" MinDuration = "0"
        MaxDuration = "infinity" Condition = "true"> </TCR>
   ...
  </TCRS>
  <CHOICES>
    <CHOICE PointIDS = "sa,sb"> </CHOICE>
  </CHOICES>
</CBSCENARIO>
```

10.5 Summary and Discussion

There are several markup languages for music such as music XML and *music markup language* (MML), and for multimedia such as *synchronized multimedia integration language* (SMIL) and *HyTime*; nonetheless, it is not easy to translate an interactive score into those formats if possible. To solve that problem, Allombert developed a XML file format for interactive scores. This file format is currently used in Virage and Acousmoscribe; however, it does not allow to represent the hierarchy, point-to-point temporal relations nor a set of possible durations of a temporal object (not even an interval).

To cope with the disadvantages of Allombert's XML, we developed a new XML file format for hierarchical interactive scores, with support for a set of possible durations (represented as an integer interval), point-to-point temporal relations. We also presented a *document type definition*. Finally, extended our XML format to encode interactive scores with conditional branching and choices. In the future, we want to translate files from *music* XML and *mml* to the interactive scores XML format. We also want to represent scores with signal processors in the XML format

CHAPTER 11

Verification

Contents

We recall from Chapter 1 that multimedia interaction systems usually do not have formal semantics. In this thesis, we defined several models of interactive scores with formals semantics based on event structures and the *non-deterministic timed concurrent constraint (ntcc)* calculus. One of the main advantages of such semantics is that they allow to verify properties of the system. We want to develop a verification tool for ntcc because ntcc gives a concrete description of the system behavior and a verification tool for ntcc can be used to verify properties of other multimedia interaction systems already modeled with ntcc.

Ntcc has been used since its beginnings to prove properties of multimedia interaction systems. Ntcc is a powerful formalism because it allows to simulate the behavior of a model and also to verify properties of the model. As an example, ntcc was used to verify properties of a musicological problem of western-african music [Rueda 2002]. The reader may also look at [Rueda 2004] and [Rueda 2001] for other examples of verification of multimedia interaction systems.

A disadvantage of ntcc is the lack of automatic verification tools available. This limits the applicability of the verification techniques to small problems. We claim for the urgent need of a verification tool for ntcc. First, because ntcc has been widely used to model reactive system and verify properties about them, but the verification had to be done by hand. Second, because there are not many frameworks to model and verify multimedia interaction systems, and ntcc has been proved to be successful in that field.

In what follows we describe related work to verification techniques for *concurrent constraint programming (*CCP*)* and other languages, we present a model checking tool *ntcc*MC[1] and its implementation, and we summarize and discuss the results.

[1] http://sourceforge.net/projects/ntccmc/

11.1 Related Work

There are several automatic verification techniques: the one mostly used is model checking. A model checker is a procedure that decides whether a given structure satisfies a logical formula [Müller-Olm 1999]. In what follows, we describe some model checkers commonly used for reactive systems and we explain why we cannot use them directly to verify ntcc models. Afterwards, we explain existing verification techniques for ccp and ntcc, why we cannot implement them as defined, and what modifications were needed to develop ntccMC.

11.1.1 Model checking for reactive systems

A well-known model checker for reactive systems is *Spin* [Holzmann 1997]. Spin is a model checker for the *Promela* language to verify *linear-time logic* (LTL) [Pnueli 1977] properties. Note that LTL properties are discrete-time properties. On the contrary, *Kronos* [Bozga 1998] and UPPAAL [Behrmann 2001] are model checkers for *timed automata* [Alur 1994] that verify not only LTL properties but also continuous-time logic properties. Although Kronos and UPPAAL are often used to verify real-time and reactive systems, they use timed automata. Timed automata model checking is far more complex and limited than verification of finite state automata. Since ntcc is a concurrent formalism for discrete time, a translation of a ntcc process into timed automata may produce one complex automaton or several automata synchronized, thus we discard Kronos and UPPAAL.

We also discard translating ntcc processes to some LTL (or, more precisely, CLTL) model-checker such as Spin. Such a translation exists for an interesting fragment of ntcc, as described in [Valencia 2005], but we conjecture that the price to pay would be an undue increase in the size of the formulae involved.

Example 11.1.1. As an example, consider the formula "eventually there is a store where the constraint c can be deduced". A translation to LTL could be "eventually we will reach state $1, 2, 3 \ldots$ or n". Note that the complexity of the classic model checking algorithm for LTL depends exponentially on the size of the formula.

Another option is to discard ntcc as operational semantics for interactive scores, use the existing Petri nets semantics and provide automatic verification for Petri nets. To handle complex synchronization patterns and to predict the behavior of interactive scenarios, as we discussed in Chapter 3, *hierarchical time stream Petri nets (*HTSPN*)* has been also used to formalize interactive scores [Allombert 2008d] and multimedia streaming systems [Sénac 1995]. There are several model checkers for time Petri nets such as *Roméo* [Gardey 2005] and TAPAAL [Byg 2009]. In addition, there are translations from time Petri nets to timed automata, which allows to verify time Petri nets in existing tools for timed automata [Cassez 2008]. Unfortunately, none of them are suitable for the hierarchical time stream extension of Petri nets. In fact, it was proved in [Boyer 1999] that HTSPN cannot be translated into time Petri nets. There are no automatic verification tools for HTSPN; in addition, the prototype mentioned in [Sénac 1995] has not been used in 17 years and is no longer functional.

In conclusion, we discard model checkers based on timed automata because ntcc is discrete time, Spin because the translation of constraints into Promela might explode exponentially the size of the model and the formula, and Petri nets because there is little work on HTSPN verification techniques and existing techniques for other time Petri nets cannot be applied directly in such Petri net extension. Therefore, a good alternative is to look at existing verification techniques for CCP that may suit better ntcc.

11.1.2 Verification techniques for CCP

There is a different alternative to model checking for verification of CCP , currently explored by Aristizábal *et al.* [Aristizábal 2010, Aristizabal 2012]. They explore a definition of *bisimilarity* for CCP . Intuitively, two systems are bisimilar if they match each other's moves. In this sense, each of the systems cannot be distinguished from the other by an observer. Aristizábal *et al.* extend the concept of bisimulation used in other process calculi for CCP . Nonetheless, the fragment they consider is very reduced, and yet not applicable to music systems; for instance, recursion and non-determinism are not allowed.

Another option is to translate a process into an automaton and use verification tools for automata. It has been shown by Saraswat *et al.* that it is possible to translate a *timed concurrent constraint (tcc)* program into a *finite state automata* (FSA) [Saraswat 1994]. This is possible because tcc is finite state; fortunately, ntcc is also finite state.

The translation of a tcc process into a FSA cannot be extended to infinite state calculi such as *time default concurrent constraint (tccp)* [de Boer 2000], thus a different approach is needed for such calculi. Falaschi *et al.* developed a bounded-time model checking algorithm for tccp [Falaschi 2006]. In addition, such a technique has also been used for a real-time extension of tccp [Alpuente 2006], and a similar approach has been used by Falaschi *et al.* in [Falaschi 2007] for ntcc. Nonetheless, there is not an algorithm nor the intuition on how to implement this technique for ntcc.

We discard a translation of ntcc into tccp to take profits of existing tools for tccp. The reason is that ntcc is finite state, whereas tccp is infinite state, thus their semantics are very different; therefore, if such translation exists, it might not be expressed compositionally and it might result in an exponential number of tccp process to represent a ntcc process. We do not apply Falaschi *et al.*'s algorithm to ntcc because semantics of ntcc and tccp are quite different, for instance, information cannot be forgotten in tccp and variables are infinite streams, thus the algorithm may need several modifications.

In conclusion, bisimilarity techniques are quite promising, but they are not yet mature enough to be applicable to ntcc nor to interactive systems because they currently do not allow non-determinism nor recursive behavior. A translation of ntcc into tccp is not appropriate because the size of the process may explode exponentially. Therefore, in what follows we study alternative verification techniques developed for ntcc.

11.1.3 Verification techniques for ntcc

The *strongest post-condition* in ntcc is a set of sequences that a process can output interacting with any possible environment. The strongest post-condition is defined in the

denotational semantics of ntcc, in Section 4.2. There is an inference system to prove that a CLTL property satisfies the strongest post-condition of a process [Nielsen 2002]. This approach can provide a semi-automatic (or even automatic) approach for verification. Up to our knowledge, there is no research on how to implement this technique, although it has been used in several handmade proofs [Rueda 2002, Rueda 2004, Rueda 2001].

Another possible approach for verification comes from the fact that it was proven by Valencia *et al.* in [Valencia 2005] that the strongest post-condition of a process, whose guards do not depend on local variables, can be translated into a Büchi automaton, and a *constraint linear-time logic (*CLTL*)* formula can be translated into a process, as we described in Section 4.2. We recall that Büchi automata are an extension of finite state automata for infinite-length input. Such automata accept exactly those runs in which at least one of the infinitely often occurring states is an accepting state.

There is a disadvantage in using the Büchi translation to make a model checker based on the classic LTL model checking algorithm presented by Schimpf [Schimpf 2009]. Computing the complement of Büchi automata is intractable, as proved by Safra [Safra 1988]. Note that the intersection also depends on the complement and the existing encoding of ntcc into Büchi heavily relies on the complement and intersection of Büchi automata.

Büchi translation cannot be exploited because the space complexity of computing the complement and intersection of Büchi automata is $2^{O(n.log(n))}$, where n is the number of states [Safra 1988]. We propose a translation of a ntcc process into a FSA inspired on the existing translation to Büchi. The space complexity of computing the complement of a non-deterministic FSA is $2^{O(n)}$, which is lower than the space complexity of computing the complement of Büchi, and the space complexity of computing the complement of a deterministic FSA is $O(n)$.

11.2 NtccMC: A Bounded-time Model Checker for ntcc

We propose a bounded-time model checking procedure for ntcc. The procedure is based on an encoding of a ntcc process and a CLTL formula into deterministic finite state automata. Nonetheless, some restrictions must be made to processes and formulae in order to define such an encoding. As an example, there is no way to represent bang's infinite behavior into FSA, thus only bounded-time model checking is possible with FSA. We recall that process *bang (!)* represents infinite behavior in ntcc, it is analogous to replication in the *pi-calculus* [Milner 1999].

11.2.1 Encoding a ntcc process into a FSA

The encoding of a ntcc process into a FSA is based on the encoding of the strongest post-condition of a ntcc process into a Büchi automaton. Nonetheless, some difficulties arises with processes bang and star.

The process *bang* (**!**) cannot be encoded in the same way as in the Büchi encoding. Following the same principle of the previous encoding, the bang could be codified as the *Kleene closure* (well-known in regular expressions). Such encoding has a problem, it recognizes sequences of any length, which is not the intended meaning of the bang. As an

example, consider $P =^{def}$ **next** $!R$ and $Q =^{def}$ $!R$. If we codify the bang as the Kleene closure, process Q can recognize sequences of any length, whereas process P recognizes sequences of at least one element. As a consequence, we remove $!P$ and use instead the bounded-time version $!_{[a,b]}P$, presented in Section 4.2.

Process $P =^{def} *Q$ cannot be defined either. We could define it as a process that eventually recognizes the sequences of the automaton of Q, but since there is no bang, this will result in a process that eventually recognizes the sequences of Q within a bounded-time, which is not the behavior of P. For that reason, we also remove $*P$ and we replace it by its bounded-time version $*_{[a,b]}P$, presented in Section 4.2.

In what follows we explain the encoding of each process into a FSA, namely a function $[\![.]\!] : Process \rightarrow Automata$. We construct a FSA that represents the strongest post-condition of a process from the first to the n^{th} time unit. In Figure 11.1, we present examples of the encoding of some processes. We recall from Chapter 4 that S is the set of revelant constraints of a process.

- Process $P =$**tell**(c) is encoded as an automaton with two states. The first state is an initial state with a transition labeled by d such that $c \models d, c \in S$. The second state is the final state and it has a transition to itself labeled by c, such that $c \in S$.

- The encoding of the parallel composition $[\![P \| Q]\!]$ is the intersection of two automata $[\![P]\!] \cap [\![Q]\!]$. Unlike Büchi automata, computing the intersection of two deterministic FSAs is simple and well-known. The intersection can be encoded as the union of the complement of two automata $F \cap G = F' \cup G'$. The complement can be obtained by exchanging the accepting states by the non-accepting states. The union can be encoded as a new automaton whose states represent pairs of states, one from F and one from G. This holds when the FSA is deterministic. For simplicity, we always encode a process into a deterministic FSA.

- Encoding of process **next** P can be done by adding a new state to the encoding of P. Such state is the new initial state. We add transitions labeled by c, such that $c \in S$, from the new initial state to the previous initial state.

- Encoding of **unless** c **next** P proceeds as the encoding into Büchi. We add a new state. Such state is the new initial state. Such state has transitions d such that $d \nvdash c, d \in S$ to the previous initial state. It is also intercepted with an automaton that accepts a constraint d such that $d \vdash c, d \in S$ and then accepts anything to model the case in which c can be deduced from the store.

- The encoding of $\sum_{i \in I}$ **when** c_i **do** P_i is given by an automaton that recognizes the following language, where C^ω is the set of infinite sequences of constraints:

$$\bigcup_{i \in I} \{d.\alpha | d \vdash c_i, d.\alpha \in [\![P_i]\!]\} \cup \bigcap_{i \in I} \{d.\alpha | d \nvdash c_i, d.\alpha \in C^\omega\}$$

- Process $![0,n]P =^{def} P \parallel$ **next** $P \parallel$ **next**^{2} P ... **next**^{n} P is encoded as the combination of parallel and next. To translate $!P$ into $![0,n]P$, we must choose the appropriate value of n such that all sequences have the same length.

- Process **local** x **in** P is encoded differently than in the encoding Büchi. Under a static analysis, we can replace processes **local** x **in** P by $P[x/fresh]$ where fresh is a "fresh variable" that does not occur free in P. This works because there are no loops, except self-loops, $fresh$ does not appear in such self-loops, and the number of time units is bounded. The validity of replacing x by $fresh$ is based on the results from Olarte's Ph.D. thesis, Section 4.4.1 [Olarte 2009a].

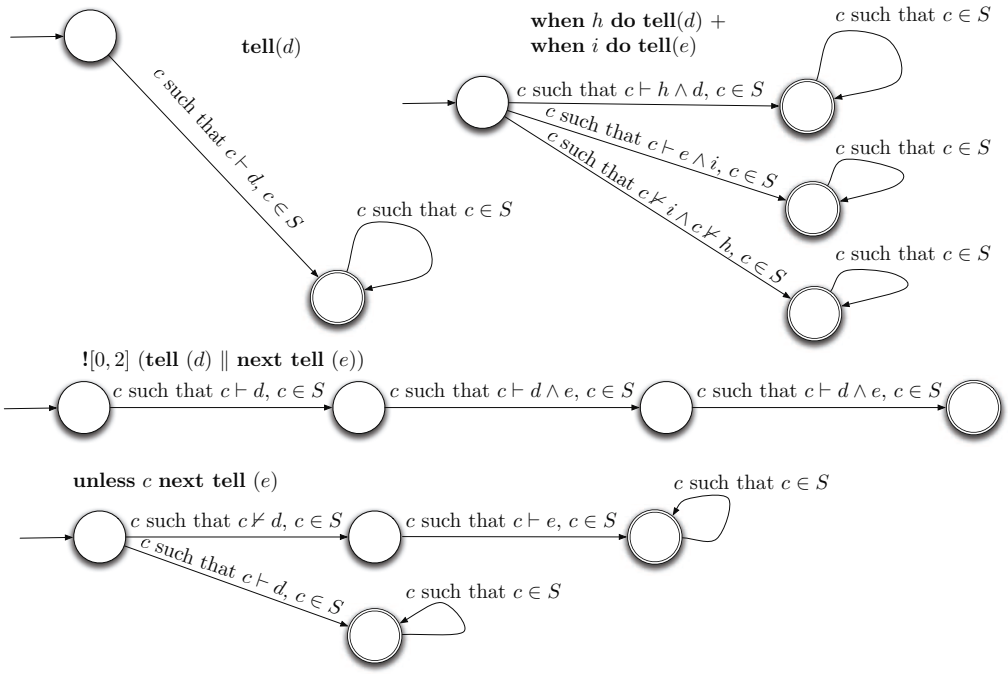

Figure 11.1: Encoding of some ntcc processes into finite state automata.

Encoding a CLTL formula into a FSA. A CLTL formula can be map into a process, as defined in Table 4.6. Nonetheless, for verification purposes we cannot represent translate $\Box F$ into $!h(F)$ and $\Diamond F$ into $*h(F)$, where h is the function to translate formulae into processes, because of the limitation of FSA. Instead, we define bounded-time versions for $\Box F$ and $\Diamond F$, as we did for processes.

11.2.2 Model checking algorithm

The model checking algorithm is based on the classic LTL algorithm for model checking. Figure 11.2 summarizes the algorithms. The algorithm proceeds as follows:

1. Translate the formula into a process.

2. Translate both processes into non-deterministic FSA.

3. Translate both automata into deterministic FSA.
4. Intersect both FSA.
5. Check language emptiness in the resulting FSA.

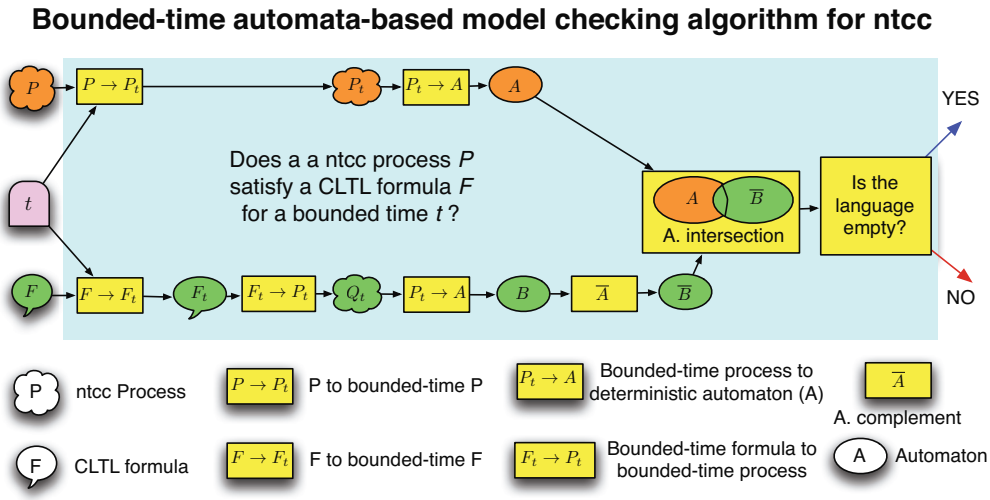

Figure 11.2: Bounded-time FSA-based model checking algorithm for ntcc.

We recall the definitions *Clock* and *sPoint* from Section 6.3.3. In what follows, we encode such processes into FSA that accept sequences of length two (i.e., they are meant to verify the behavior of the process for up to two time units). Process *Clock* is defined as follows and Figure 11.3 shows its encoding. Afterwards, process *sPoint* is defined and Figure 11.4 shows its encoding.

Example 11.2.1. $Clock(k) =^{def}$ **when** $k < n_\infty - 1$ **do** (**tell** $(clock = k)$ || **next** $Clock(k+1)$)

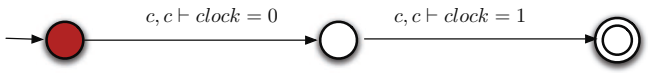

Figure 11.3: Encoding strongest post-condition of process *Clock* for two time units into a finite state automaton. The colored state is the initial state.

Example 11.2.2. $sPoint_{i,\mathrm{Pr}} =^{def}$ **when** $\bigwedge_{j \in \mathrm{Pr}(i)} launched_j$ **do** (
unless $clock + 1 < p_i \vee launched_i$ **next** $Launch_i$
||**when** $clock + 1 < p_i$ **do next** $sPoint_{i,\mathrm{Pr}}$)

$\|$**unless** $launched_i$ **next** $sPoint_{i,\mathrm{Pr}}$

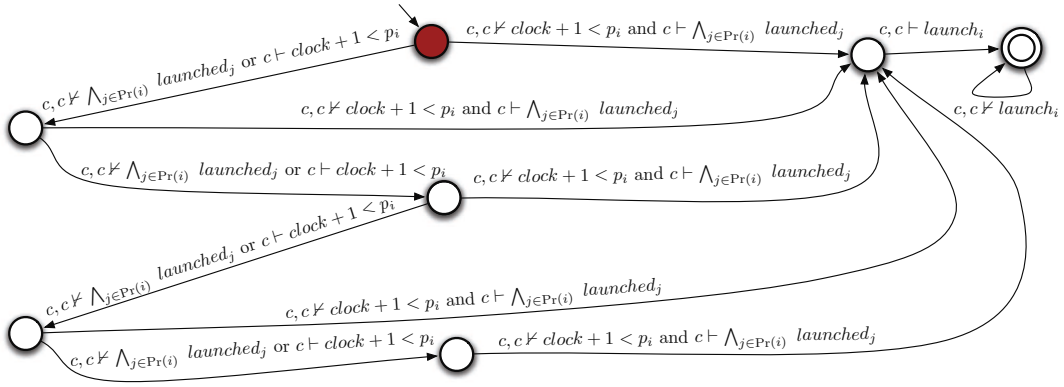

Figure 11.4: Encoding strongest post-condition of process *sPoint* for two time units into a finite state automaton. The colored state is the initial state.

11.3 Implementation of NtccMC

We use *object-oriented programming* to build a reusable and extensible architecture to represent ntcc processes and CLTL formulae. We define abstract classes for processes, formulae and constraints. Concrete classes for processes are capable to produce a syntax tree and a FSA. Concrete classes for formulae are capable to produce its equivalent process. Constraints are equipped with methods to tests whether the constraint can be deduced from another. The implementation is made in C++ and we use four libraries: (1) the constraint solving library *Gecode* [Tack 2009] to test constraint deduction and constraint equivalence using search, (2) the *automata standard library* [2] to handle FSAs efficiently, and (3) *Flex++* and (4) *Bison++*[3] to build a parser. In what follows we explain how to implement constraint deduction, the automata operations and the parser.

Constraint deduction. We use *Gecode* for constraint deduction, based on search. Gecode is a very efficient constraint solving library equipped with state-of-the-art propagation and search algorithm. Propagators narrow the possible values of the variables, but it is often needed to perform search to test if a constraint can be deduced from another, testing over all the possible values of their narrowed domains. *Constraint reification*, described Def. 9.1.2, is useful to model the deduction problem. To achieve a correct construction of the FSA, we need to employ search to compute constraint deduction and constraint equivalence.

[2] http://astl.sourceforge.net/

[3] http://en.wikipedia.org/wiki/Flex_lexical_analyser

To verify if a constraint c can be deduced from a constraint d (i.e, $d \vdash c$), we proceed as follows. First, we add two reified constraints of the form $d \Leftrightarrow b_1$ and $c \Leftrightarrow b_2$, where b_1 and b_2 are two fresh boolean variables. Then, we post a constraint of the form $(b_1 \Rightarrow b_2) \Leftrightarrow b_3$, where b_3 is a fresh boolean variable. Finally, we post a constraint $b_3 = 0$ and we perform search on the set of variables used by the constraints c and d. At this point we need to perform an exhaustive search. If the constraint c can be deduced from d, this problem is unsatisfiable; therefore, no solutions would be found which means that c can be deduced from d.

Automaton operations. The *automata standard library* provides a class to model deterministic FSA with operations for drawing, creation, intersection, union, and complement of deterministic FSA. In addition, it provides a class to define non-deterministic FSA and a procedure to convert non-deterministic into deterministic FSA. There are several implementations for automata: matrix, map, binary search, arrays and hash tables. We found out that using a hash-table implementation speeds up the model checking process because it has almost constant access and edition times; however, it requires, before compilation, to fix the size of the alphabet which is the number of relevant constraints.

Parser for ntcc. We developed a parser on the lines of [Pabón 2003] with extension to represent extended operators of ntcc such as process definition, bounded-time bang and bounded-time star. We used Flex++ and Bison++ to make our parser. Flex++ is a lexical analyzer generator and Bison++ is a is a parser generator, both for C++, for *look-ahead from left to right* (LALR) parsers context-free languages, appropriate for ntcc syntax.

Although the syntax of the ntcc calculus is apparently simple, multimedia interaction systems often require complex constructions such as recursion, cells, bounded bang (![n,m]), sum operator (+), bounded asynchrony (*[n,m]), finite repetition of next ($next^n$) and procedure definitions ($P(a_1 ... a_n) =^{def} Q$). In addition, to avoid the extensive use of parenthesis, we need to define the precedence of its operators and the associativity axioms. Furthermore, the compiler must be able to parse CLTL formulae and finite-domain constraints.

11.4 Summary and Discussion

Although ntcc is finite state and can be encoded into Büchi automata, the existing encoding into Büchi has a space complexity non tractable. In fact, model checking for LTL is, for most cases, intractable. Algorithms known for most fragments of LTL are of exponential time; model checking for only a few fragments of LTL is polynomial [Bauland 2011] and we argue that such fragments are of low interest for music properties. We believe that, for practical purposes, finite state automata (FSA) is better than Büchi. In this chapter we introduced a new model checking tool for ntcc: the bounded-time FSA-based model checker, *ntcc*MC.

Although FSA operations have lower complexity than operations over Büchi, the implementation needs to be improved to be used in bigger examples. The hash-table based

automata class, provided by the *automata standard library*, is parametrized during compilation time by the size of the alphabet which is the number of relevant constraints. In addition, the number of relevant constraint is bounded by $n!$, where n is the number of constraints that appear in the process and the formula. In addition of having a factorial number of constraints, constraint deduction is based on search, as opposed to the case of simulation of ntcc, thus exponential on the size of the domains and the number of variables in the worst case.

A disadvantage of most ntcc tools is the syntax to write the input. Previous attempts to write ntcc processes directly as C++ classes or as Lisp functions has been proven to be insufficiently user-friendly. There is even a visual language to represent ntcc, but we found it insufficiently expressive and not user-friendly [Fernández 2004]. A compiler to parse ntcc into C++ classes is the "missing link" to allow non-programmers to use the real-time capable interpreter for ntcc (*Ntccrt*) and the ntcc time-bounded model checker (*ntccMC*), and could be the base for other CCP tools. In ntccMC we provide a prototype of such parser, but the parser can be improved. As an example, build an efficient representation of the process hierarchy, instead of a directed tree, so that two equivalent processes do not have to be encoded twice into FSA.

In the future, we believe that it is worth to implement a semi-automatic procedure based on ntcc proof system or a model checker based on *abstract interpretation* as another alternative.

CHAPTER 12

Concluding Remarks

Contents

In this chapter we present a summary of this dissertation; afterwards, we present the main conclusions of this thesis; and, finally, we present some future work directions.

12.1 Summary

Technology has shaped the way on which we compose and produce music. Several artistic domains have been benefiting from technology developments; for instance, *Electroacoustic music*, *Experimental music*, *non-linear multimedia* and *interactive multimedia*. In this thesis we focused on interactive multimedia.

As stated in Chapter 1, in *interactive scores*, interactive multimedia scenarios are represented by *temporal objects* and *temporal relations*. Examples of temporal objects are sounds, videos and light controls. A temporal object may contain other temporal objects: this hierarchy allows us to control the start or end of a temporal object by controlling the start or end of its parent. Temporal relations provide a partial order for the execution of the temporal objects; for instance, temporal relations can be used to express precedence between two temporal objects.

In Chapter 3, we discussed several software and formalisms related to interactive multimedia, and existing models and implementations of interactive scores. As an example, we described sequencers which are software for multimedia interaction that are usually based on a fixed timeline or on cue lists. Some software products provide both time models, but they are temporally unrelated. An advantage of interactive scores is to relate temporally both time models and to allow conditional branching and loops.

Process calculi are approaches to formally model concurrent systems. As an example, the *non-deterministic timed concurrent constraint programming (ntcc)* calculus describes partial information by the means of constraints. Ntcc models time as discrete time units, and it allows asynchrony and non-determinism. Ntcc has been used in the past to model interactive scores because it handles naturally temporal constraints. Petri nets is another model of concurrency with an intuitive graphical notation. An extension of Petri nets with

time and hierarchy has also been used to model interactive scores in the past. The Petri nets semantics of interactive scores were implemented in an efficient C++ library called Iscore, and it is currently being used by Virage, Acousmouscribe and i-score.

There exist other models of interactive scores. First models were conceived to control the start and end times of the notes of a score. They also included different temporal relations; for instance, to model two temporal objects that overlaps, by the means of Allen's relations. Later extensions included a Petri nets operational semantics. There are also extensions of interactive scores with conditional branching. In this thesis, we developed new semantics and extensions of interactive scores inspired by these models and implementations.

In Chapter 5, we presented a simple model of interactive scores with temporal objects without hierarchy, equipped with point-to-point temporal relations. We described the structure of the score. We also defined the event structures semantics of a score, and several properties of the score using its event structures semantics. We also showed that the time complexity of the playability of a score is NP-complete, but the problem can be solved in polynomial time when the durations are intervals.

Afterwards, in Chapter 6, we equipped the structural definition of a score presented in Chapter 5, with a directed-tree hierarchy. We extended the event structures semantics of Chapter 5 by adding the hierarchy. We gave operational semantics to this model based on ntcc. We proved that the temporal constraints obtained from the ntcc model are a subset of those from the event structures. Complexity of computing the playability of scores with hierarchy is the same as the complexity of score without hierarchy.

In Chapter 7, we presented two models of conditional branching interactive scores. The first model does not include loops. We defined a structural definition based on the idea of *constraint system*, defined in Chapter 4. Afterwards, we gave event structures semantics and we argued that the hierarchical scores, described in Chapter 6, can be encoded into this new model, and, additionally, it can express choice and conditions. The second model of conditional branching includes loops. We slightly modified the structural definition of the score to allow loops. We gave some ideas on how to define operational semantics for both models. We have not yet study the complexity of computing playability, safeness or any other properties for this extension.

In Chapter 8, we presented another extension of interactive scores. We extended the formalism of interactive scores with sound processing and micro controls for sound processors. We presented an encoding of the scenario into a ntcc model and a Faust program interoperating.

In Chapter 9, we argued that Ntccrt is the most appropriate interpreter for ntcc to run our models because it is real-time capable. We simulated two models: one with conditional branching and one with signal processing. For the conditional branching score, we made a stress test in which we found out that ,using Ntccrt, we can run models with up to 500 temporal objects, under low CPU load. For the signal processing score, we made another stress test in which we found out that the combination of Ntccrt and Faust behaves outstanding even under a high CPU load. Equivalent programs in Pure Data have a very high average relative jitter, whereas using Ntccrt and Faust the jitter is almost imperceptible.

We executed interactive scores, but models were written by hand and there were no

means to assure its persistence; therefore, we also defined a file format for future applications in Chapter 10. We developed a new XML file format for hierarchical interactive scores, with support of a set of possible durations and point-to-point temporal relations. We also presented a document type definition. We also extended our XML format to encode interactive scores with conditional branching and choices.

Finally, to complete our framework, we needed automatic verification. Although ntcc is finite state and can be encoded into Büchi automata, the existing encoding into Büchi has a space complexity non tractable. We believe that, for practical purposes, finite state automata (FSA) is better than Büchi automata. In Chapter 11, we introduced a new model checking tool for ntcc: a bounded-time FSA-based model checker, called *ntcc*MC.

12.2 Discussion

Most scenarios and musical pieces with interactive controls have no formal semantics. Interactive scores is a formalism to describe interactive scenarios based on temporal constraints. In this dissertation, we introduced an event structures semantics of interactive scores, we formalized some properties, and we proved that the event structures semantics complies with the temporal constraints of the score. With the event structures semantics, we expressed several properties about the traces of execution that are difficult to express and prove using constraints.

We introduced the *dispatchable event structures (*DES*)*: event structures whose temporal object durations and temporal distances among objects are integer intervals. DES can be dispatched online by relying only on local propagation: This is achieved by transforming the constraint graph into an *all-pairs shortest-path graph*; however, that drastically increases the number of arcs. In the future, we propose to minimize the number of arcs of such networks, as proposed by Muscettola *et al.* [Muscettola 1998].

Although event structures provide a theoretical background to specify properties and understand the system, there is no difference between interactive objects and static temporal objects in the event structures semantics: such a difference can only be expressed in the operational semantics. This means that the event structure semantics are not *fully abstract* with respect to the operational semantics: Operational equivalence does not always coincides with denotational equality. It is an open issue how to capture the behavior of interactive objects in the event structures semantics.

Operational semantics are based on the dispatchable normal form of the event structures of the score. A score is in normal form when it does not have zero-duration event delays. The computation of the normal form is similar to the algorithm to transform a score into a Petri net proposed by Allombert *et al.* [Allombert 2009]: In Petri nets semantics of interactive scores, points of temporal objects executed at the same time share the same *place* (i.e., state).

Comparison with Allombert *et al.*'s model. We believe that this dissertation extends significantly Allombert *et al.*'s model because it provides a concise operational semantics for interactive scores whose temporal object duration can be any interval of integers. Al-

lombert *et al.* proposed temporal relations with flexible intervals with only $\{0\}$, $[0,\infty)$ and $(0,\infty)$ intervals [Allombert 2006, Allombert 2008b]. In fact, arbitrary integer intervals are not allowed in neither Virage nor i-score, only flexible-time intervals. To handle temporal relations with arbitrary intervals, Allombert proposed in [Allombert 2009] to either build a *Hierarchical colored time stream Petri net*, adding a big number of new places (states), or to use a constraint store that is unrelated to the Petri nets semantics, and the combined semantics of Petri nets interacting with a constraint store are not given.

There is another disadvantage of Allombert *et al.*'s models: Temporal relations are limited to *Allen's relations*. Allen's relations do not allow to represent quantitative relations between two objects easily; for instance, "object a occurs 3 time units after object b". Using Allen's relations, it is neither possible to say "the start of object a is before the end of object b". These kind of relations are easily modeled using point-to-point temporal relations. In fact, recently, i-score has moved forward to point-to-point temporal relations.

A conditional branching extension was presented in [Allombert 2009], but no temporal relations were allowed. We struggled to allow temporal relations and conditional branching in the same model. As an example, in the score of Figure 7.2, it is possible to model conditions and also preserve temporal properties over all the branches, for instance, that $\Delta_b + [2,5] = \Delta_a + [1,4]$.

In our first models of conditional branching, published in [Toro 2010b, Toro 2010c], we allowed branches starting in the same point have different durations. We left aside such an approach because it makes many scores incoherent and unplayable, as discussed in Chapter 7.

An advantage of our extension of interactive scores with conditional branching with respect to previous models of interactive scores, Pure Data, Max and Petri Nets is representing declarative conditions by the means of constraints. Complex conditions, in particular those with an unknown number of parameters, are difficult to model in Max or Pd. To model generic conditions in Max or Pd, we would have to define each condition either in a new patch or in a predefined library. In Petri nets, we would have to define a net for each condition.

A disadvantage of our conditional branching model is that the number of event conflicts increases exponentially with respect to the hierarchy depth. Fortunately, the hierarchy depth is usually not so big, thus we argue that we do not need a formalism that supports hierarchical constructions, such as hierarchical Petri nets or statecharts.

Using timed event structures with conflicts, it is possible to model conditional branching: the possibility to choose among different continuations of the piece based on the preferences of the musician. In addition, Langerak describes in [Langerak 1992] how to encode recursive processes into event structures; in fact, loops in the interactive scores could be encoded with such a technique. Unfortunately, conditional branching drastically increases the complexity of the system; for instance, a score may contain dead-locks. An alternative for automated verification is *constraint programming*; for instance, to verify the playability of a score and calculate the potential time positions of the points of the score. Nonetheless, once again, we argue that, for some properties, the notion of *trace* is more appropriate.

Another advantage of our event structure semantics and our operational semantics is that they can express *trans-hierarchical relations*: temporal relations between objects with

different parents. Trans-hierarchical relations are not possible to model with hierarchical time stream Petri nets used by Allombert *et al.* These relations are useful; for instance, to model temporal relations between videos and sounds that are contained in different temporal objects, allowing to define temporal relations among different media.

A key issue of this dissertation is that we executed interactive scores in a efficient manner. We want to encourage the use of process calculi to develop reactive systems. For that reason, this research focused on developing real-life applications with ntcc and showing that our interpreter *Ntccrt* is a user-friendly tool, providing a graphical interface to specify ntcc models and compiling them to efficient C++ programs capable of real-time interaction in Max and Pure Data (Pd). We argue that using ntcc to model, verify and execute reactive systems decreases the development time and guarantees correct process synchronization, in contrast to the graphical patch paradigm of Max and Pd.

Disadvantages of our models. A disadvantage of most ntcc tools is the syntax to write the input. Previous attempts to write ntcc processes directly as C++ classes, Lisp functions or visual objects has been proven to be insufficiently user-friendly. A compiler to parse ntcc into C++ classes is the "missing link" to allow non-programmers to use the real-time capable interpreter for ntcc (Ntccrt) and the ntcc time-bounded model checker (ntccMC), and could be the base for other CCP tools.

There are some other problems to execute interactive scores with Ntccrt. First, To compute the event structures semantics, its normal form and the dispatchable form by hand is very difficult. In the future, this should be done automatically. Second, ntcc recursive definition cannot be translated directly to Ntccrt because their encoding is based on nested non-deterministic choices hard to simulate. In the future, variables should be treated differently; for instance, using variables that can change value from a time unit to another one. Unfortunately, there are other problems that Ntccrt must overcome. Third, one may argue that although we can synchronize Ntccrt with an external clock (e.g., a metronome object) provided by Max or Pure Data, this does not solve the problem of simulating models when the clock step is shorter than the time necessary to compute a time-unit. To solve this problem, Sarria proposed to develop an interpreter for the *real time concurrent constraint (rtcc)* [Sarria 2008] calculus, which is an extension of ntcc capable of modeling time units with fixed duration. The reader may find a further discussion on executing time units with fixed durations in [Toro 2009].

One may also argue that interactive scores had little applicability because they do not allow to describe signal processors. In this dissertation, we also extended the formalism of interactive scores with sound processing and micro controls for sound processors. We present an encoding of the scenario into a ntcc model –executed using the real-time capable interpreter *Ntccrt*– and a Faust program. Both programs interact during the performance of the scenario. We show how some interesting applications can be easily modeled in the formalism and how they can be executed in Pure Data. Using Faust and Ntccrt, we achieved an efficient and real-time capable performance of a scenario –even under high CPU-load. Nonetheless, our final goal is to integrate Ntccrt and Faust in a standalone program.

Expressiveness of our models. Note, for instance, that the score in Figure 8.3 is difficult to model in the existing tools for interactive scenarios presented in Chapter 3. As an example, Qlab and Live do not allow to model delays of 100 samples. Max and Csound allow to express delays of 100 samples, but it is very hard to synchronize processes whose durations are integer intervals such as $duration \in [5, 10]$. Nonetheless, we need to evaluate interactive scores not only by its efficacy but also by its expressiveness to model multimedia interaction scenarios.

There is an interesting framework to evaluate the expressiveness of interactive multimedia formalisms: *Janin's dimensions*. There are several dimensions in multimedia interaction, according to Janin[1]: *Abstraction* that represents the hierarchy of temporal objects, *time* that represents the causality and can be thought as the *logical implication*, *parallelism* that represents that two (or more) objects can be executed simultaneously and can be though as an *logical and*, *alternative* that represents conditional branching and can be though as a *logical or*. Finally, there are dimensions for *value* that represents, for instance, the value of the pitch, volume or pan. Janin's dimensions are represented in Figure 12.1.

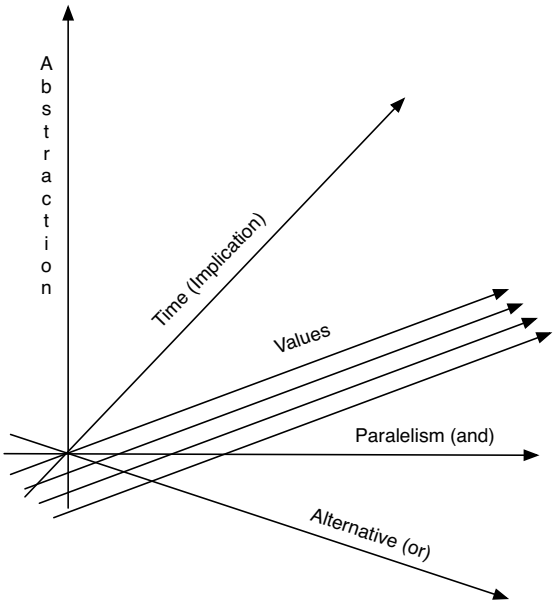

Figure 12.1: Janin's dimensions of interactive multimedia.

The hierarchical model of interactive scores allows us to express abstraction, time and values in the same two-dimensional space. In fact, i-score represents such interactive scores in a two-dimensional space. In the conditional branching model we can express abstraction, time, value and alternative, all in the same two-dimensional space because all branches starting on the same point have the same duration. Finally, in the signal processing extension, we can express time, value and parallelism in one two-dimensional space, and time,

[1] http://www.labri.fr/perso/janin/index_fichiers/Magma.jpg

value and *dataflow* in another two-dimensional space. We argue that the dataflow dimension is missing among Janin's dimensions and should also be considered. The dataflow dimension describes how sound is transferred from one process to another. To represent time, value and dataflow together, we would need a tridimensional space; otherwise, arrows representing dataflow will overlap with those representing temporal relations.

12.2.1 Answers to problem statements

We have identified, in Chapter 1, seven problems with existing software to design multimedia scenarios: (1) there is no formal model for multimedia interaction, (2) multimedia scenarios have limited reusability and difficulties with the persistence of multimedia scenarios, (3) time models (fixed timeline and cue lists) are temporally unrelated, (4) most multimedia interaction software products provide no hierarchy, (5) the different time scales are unrelated, (6) schedulers for multimedia scenarios are not appropriate for soft real-time, and (7) there is no model to combine temporal relations and conditional branching. In what follows we explain how the interactive scores formalism solves those problems.

First, interactive scores is a formalism to model multimedia scenarios. Event structures semantics allows to specify properties over the traces of execution. Ntcc semantics allows to understand the execution of the score and to specify temporal properties as well. Both semantics were proved to be related. Therefore, interactive scores is a formal model for multimedia interaction.

Second, scenarios described in interactive scores can be preserved because they have formal semantics. In addition, signal processors can be specified in Faust, which also has formal semantics. In fact, Faust can be used for preservation of music pieces because it provides formal semantics of all the audio processors used in the music piece [Mihalic 2011, Barkati 2011].

Third, time models are related temporally, for instance, we can specify that an object is executed strictly in the third second of execution, and we can also express that another object is executed between two and five seconds after the end of the previous object. Although, during the execution, micro controls are managed by Faust and macro controls by ntcc, it is also possible to express, for instance, that an object starts 500 microseconds after another, and it will end one second before another object.

Fourth, hierarchy is available in our model and it allows to constrain the execution times of the objects contained in another object.

Fifth, different time scales are available in our tool, but, unfortunately, they are temporally unrelated, as in many tools; for instance, it is not possible to relate the frequency of the clock that controls ntcc discrete time units to the signal processing sampling rate.

Sixth, the system is appropriate, even under high CPU-load, to interact with a human in real-time, as shown in the quantitative results, in Chapter 9. The solution to this problem is relevant for the multimedia interaction domain because, in addition to sound processing, the computer may execute at the same time complex video and image operations. For that reason, we did the evaluation of our system under high CPU-load, obtained by executing several video processing operations concurrently.

Seventh, in interactive scores, it is now possible to combine conditions and intervals

into a new type of relation called *time conditional relations*. In fact, by labeling these relations by `true` conditions, we can also express scores written in the pure temporal model. We managed to combine conditions and temporal relations by making the assumption that all branches starting in the same point have the same duration.

12.3 Future Directions

We propose some directions on the study and applications of interactive scores. Our final goal is to have a complete framework, as shown in Figure 12.2. The translation of conditional branching scores with loops into event structures is missing. In addition, operational semantics of conditional branching scores, for the general case, are missing. The translation of event structures semantics of scores with arbitrary durations into ntcc is also missing. Formal semantics of the integration of ntcc and Faust are missing. Some improvements for the model checker are missing to make it fully usable, and finally stand alone programs are missing to allow different applications of interactive scores, such as applications for music pedagogy. In what follows, we explain in detail some of these issues.

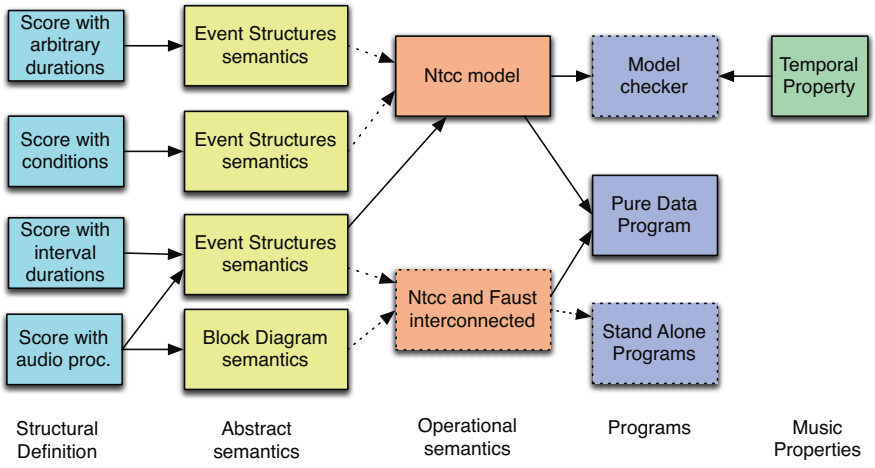

Figure 12.2: Diagram of the complete interactive scores framework. Dashed-arrows and dashed-lines represent translations, semantics and programs that are missing or are incomplete.

Signal processing extension. To improve the expressiveness of interactive scores, we should allow multiple points inside a temporal object, instead of just start and end points, as usual. Janin has already explained the advantages of such an approach to model rhythmical structures [Janin 2012] .

We also propose to extend our implementation to handle audio files efficiently. *Libau-*

diostream[2] is an audio library, developed at the french research institute *Grame*[3], to manipulate audio resources through the concept of streams using Faust programs. Including Libaudiostream in our framework, it will be possible to design a scenario where a temporal object loads a sound file into memory, filter it in Faust, and then, play the sound in Faust at the appropriate time. Precision is guaranteed because the time to load the file and to process it is foreknown in the scenario. Currently, we have to rely on third-party programs, such as Pd, to do handle audio files, and to communicate the control signals from Ntccrt to Faust.

It has been already discussed that Faust can be used to assure the persistence of music pieces with sound synthesis. We believe that such an approach could be used for the extension of interactive scores with signal processing. To solve that problem, Allombert developed a XML file format for interactive scores. This file format is currently used in Virage and i-score; however, it does not allow to represent the hierarchy, point-to-point temporal relations nor a set of possible durations of a temporal object.

In the future, we also want to to translate files from *music* XML and *music markup language* (*mml*) to our interactive scores XML format. We also want to represent scores with signal processors in our XML format.

Conditional branching extension. Event structures semantics for scores with loops is not easily defined because events can only be executed once; therefore, to define semantics we need infinite number of events, as proposed by Langerak in [Langerak 1992]. Afterwards, it will be required to translate such event structures semantics into operational semantics in ntcc with a finite number of processes.

Automatic verification. At the time of this writing, there are no formal semantics of a heterogeneous system that synchronizes concurrent objects, handles global constraints, and controls audio and video streams. Modeling this kind of systems will be useful in other domains such as *machine musical improvisation* and *music video games*. An advantage over the existing implementations of these systems will be verification.

We believe that any Faust program could be translated into ntcc based on the results obtained by Rueda *et al.* in [Rueda 2005b]. Rueda *et al.* translated the Karplus-Strong Faust program into ntcc. Although it is clear that the execution of a Ntccrt simulation cannot be done at the sound processing sampling frequency, such a translation could be used to verify properties of correctness of a scenario where ntcc and Faust interact (e.g., playability).

In the proof system of ntcc, we could prove properties like "10 time units after the event e_A is launched, during the next 4 time units, the stream B is the result of applying a *gain filter* to the stream A"; however, real-time audio processing cannot be implemented in Ntccrt because it requires to simulate 44100 time units per second to process a 44.1 kHz sound. If we replace some ntcc processes by Faust plugins, we can execute such a system efficiently, but we cannot verify that the properties of the system hold. There is one open

[2] http://libaudiostream.sourceforge.net/
[3] http://www.grame.fr/

issue: How to prove that a Faust plugin that replaces a ntcc process obeys the temporal properties proved for the process. We discussed this issue in [Toro 2010a].

A first step to achieve the goal explained above is our model checker for ntcc, ntccMC. In ntccMC, we provide a prototype of a parser for ntcc syntax, but the parser can be improved. As an example, build an efficient representation of the process hierarchy, instead of a directed tree, so that two equivalent processes do not have to be encoded twice.

There is another disadvantage of *ntcc*MC: Although FSA operations have lower complexity than operations over Büchi, the implementation needs to be improved to be used in bigger examples. The hash-table based automata class, provided by the *automata standard library*, is parametrized, during compilation time, by the size of the alphabet which is the number of relevant constraints. In addition, the number of relevant constraints is bounded by $n!$, where n is the number of constraints that appear in the process and the formula. In addition to having a factorial number of constraints, constraint deduction is based on search, thus the domains of the variables should not be too big to be tractable.

Scores whose temporal objects have arbitrary durations. This extension will allow us to represent rhythmical patterns using temporal objects. When the duration of a temporal object can be an arbitrary set of integers, we can model rhythmical patterns; for instance, that a music object should be played at beats one, three or five (but not two nor four). Constraints of this form are found in the improvisation system presented by Rueda and Valencia in [Rueda 2004].

Example 12.3.1. As an example, Figure 12.3 is a score to represent rhythms. Object a's start time could be in the $1st, 3rd, 5th, 9th$ or $12th$ time unit and its duration could be $1, 2, 3$ or 4 time units.

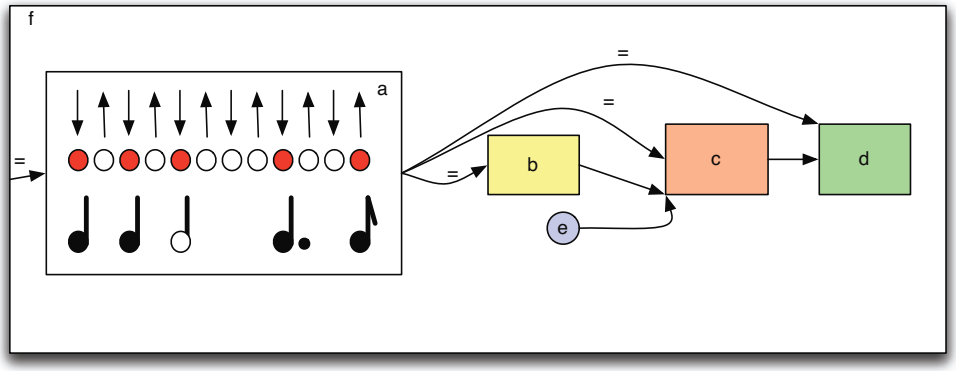

Figure 12.3: A scores whose temporal objects have arbitrary durations.

We have shown, in Chapter 6, that the satisfiability of a score with this kind of temporal constraints is equivalent to a disjunctive temporal problem, which is well-known to be NP-complete. One alternative to cope with this problem is to do a static analysis; for

instance, a space efficient backtrack-free representation for constraint satisfaction problems [Beck 2008]; however, to achieve such as representation, the order on which the temporal objects are going to be executed must be foreknown. Nonetheless, there are some scores, for instance Figure 5.2, in which this is possible, but for many other it is not possible, for instance, the score in Figure 7.3.

Another possibility to cope with this problem in real-time could be an extension of Truchet's approach to solve music constraint satisfaction problems with local search [Truchet 2003]. Nonetheless, her algorithm requires random initialization of the variables and iterative refinements. Such a random initialization could be an incoherent representation of the temporal objects in the timeline; for instance, an end point could be executed before a start point.

Pedagogic applications. There are several possible pedagogic applications that can be developed using interactive scores. One alternative is to use interactive scores for rhythmic exercises for music students, easily modeled by constraints. Anders *et al.* have already discussed this approach [Percival 2008], but we believe that it could be improved by allowing user interactions and temporal relations, which is possible in interactive scores.

Another possibility is using user gestures to generate Electroacoustic music for pedagogical purposes. This was not possible before in interactive scores due to the lack of a signal processing extension. In the future, we could imagine scenarios, as those proposed by Kurtag *et al.* [Kurtag 2007].

Finally, another possibility for future work is to use automatic generated fingering for piano or guitar to generate scores in which only "easy" playable notes (according to a fingering analysis) are played by the user and the "hard" playable notes are played by the computer. Note that automatic generation of piano fingering has been already studied by Robine, who also describes several related work on that subject [Robine 2007].

Bibliography

[Allen 1983] James F. Allen. *Maintaining Knowledge about Temporal Intervals*. Communication of ACM, vol. 26, 1983. (Cited on pages 35, 62 and 78.)

[Allombert 2005] Antoine Allombert and Myriam Desainte-Catherine. *Interactive scores: A model for specifying temporal relations between interactive and static events*. In Journal of New Music Research, 2005. (Cited on page 37.)

[Allombert 2006] Antoine Allombert, Gérard Assayag, M. Desainte-Catherine and Camilo Rueda. *Concurrent Constraint Models for Interactive Scores*. In Proc. of Sound and Music Computing (SMC) '06, Marseille, France, 2006. (Cited on pages 17, 19, 34, 38, 71, 78 and 146.)

[Allombert 2007] Antoine Allombert, Gérard Assayag and Myriam Desainte-Catherine. *A System of Interactive Scores based on Petri Nets*. In Proc. of SMC '07, Athens, Greece, 2007. (Cited on pages 37, 72 and 78.)

[Allombert 2008a] Antoine Allombert. *Un format de partitions interactives*. Document numérique, vol. 11, page 160, 2008. (Cited on page 124.)

[Allombert 2008b] Antoine Allombert, Gérard Assayag and Myriam Desainte-Catherine. *Iscore: a system for writing interaction*. In Proc. of 3rd International Conference on Digital Interactive Media in Entertainment and Arts (DIMEA) '08, pages 360–367, New York, NY, USA, 2008. ACM. (Cited on pages 17, 39 and 146.)

[Allombert 2008c] Antoine Allombert, Myriam Desainte-Catherine and Gérard Assayag. *De Boxes à Iscore: vers une écriture de l'interaction*. In Proc. of Journées d'Informatique Musicale (JIM) 2008, 2008. (Cited on page 39.)

[Allombert 2008d] Antoine Allombert, Myriam Desainte-Catherine, J. Larralde and Gérard Assayag. *A system of Interactive Scores based on qualitative and quantitative temporal constraints*. In Proc. of 4th International Conference on Digital Arts (Artech) '08, Porto, Portugal, 2008. The Artech International Association. (Cited on pages 37, 62 and 134.)

[Allombert 2009] Antoine Allombert. *Aspects temporels d'un système de partitions numèriques interactives pour la composition et l'interprétation*. PhD thesis, Université de Bordeaux, November 2009. (Cited on pages 16, 17, 38, 39, 62, 85, 145 and 146.)

[Allombert 2010] Antoine Allombert, Pascal Baltazar, Raphaël Marczak, Myriam Desainte-Catherine and Laurent Garnier. *Designing an interactive intermedia sequencer from users requirements and theoretical background*. In Proc. of International Computer Music Conference (ICMC) '10, 2010. (Cited on pages 17 and 39.)

[Allombert 2011] Antoine Allombert, Myriam Desainte-Catherine and Mauricio Toro. *Modeling Temporal Constrains for a System of Interactive Score*. In Gérard Assayag and Charlotte Truchet, editors, Constraint Programming in Music, Chapter 1, pages 1–23. Wiley, 2011. (Cited on page 24.)

[Alpuente 2006] María Alpuente, María del Mar Gallardo, Ernesto Pimentel and Alicia Villanueva. *Verifying Real-Time Properties of tccp Programs*. Journal of Universal Computer Science, vol. 12, no. 11, pages 1551–1573, 2006. (Cited on page 135.)

[Alur 1994] Rajeev Alur and David L. Dill. *A Theory of Timed Automata*. Theoretical Computer Science, vol. 126, pages 183–235, 1994. (Cited on page 134.)

[Aristizábal 2010] Andrés Aristizábal. *Bisimilarity in Concurrent Constraint Programming*. In 26th International Conference on Logic Programming (ICLP) 2010, 2010. (Cited on page 135.)

[Aristizabal 2012] A. Aristizabal, F. Bonchi, L. Pino and F. Valencia. *Partition Refinement for Bisimilarity in CCP*. In Proc. of the 27th ACM Symposium On Applied Computing (SAC 2012), 2012. (Cited on page 135.)

[Assayag 2006] Gérard Assayag, Georges Bloch, Marc Chemillier, Arshia Cont and Shlomo Dubnov. *OMax brothers: a dynamic topology of agents for improvization learning*. In Proc. of the 1st ACM workshop on Audio and music computing multimedia (AMCMM) '06, pages 125–132, New York, NY, USA, 2006. ACM. (Cited on page 28.)

[Baier 1998] Christel Baier, Joost-Pieter Katoen and Diego Latella. *Metric Semantics for True Concurrent Real Time*. In Proc. of International Conference on Automata, Languages and Programming (ICALP) '98, Berlin, Germany, 1998. Springer. (Cited on pages 16, 43, 44, 45 and 46.)

[Baltera 2007] Constance G. Baltera, Sara B. Smith and Judy A. Flanklin. *Probabilistic Interactive Installations*. In Proc. of the Florida Artificial Intelligence Research Society Conference (FLAIRS) '07, pages 553–558, 2007. (Cited on page 7.)

[Barkati 2011] Karim Barkati and Yann Orlarey. *Auto-documentation mathématique pour le traitement du signal avec Faust*. In Proc. of Journées d'informatique musical (JIM), 2011. (Cited on page 149.)

[Bauland 2011] Michael Bauland, Martin Mundhenk, Thomas Schneider, Henning Schnoor, Ilka Schnoor and Heribert Vollmer. *The tractability of model checking for LTL: The good, the bad, and the ugly fragments*. ACM Trans. Comput. Logic, vol. 12, no. 2, 2011. (Cited on page 141.)

[Beck 2008] J. Christopher Beck, Tom Carchrae, Eugene C. Freuder and Georg Ringwelski. *A Space-Efficient Backtrack-Free Representation for Constraint Satisfaction Problems*. International Journal on Artificial Intelligence Tools, vol. 17, no. 4, pages 703–730, 2008. (Cited on page 153.)

[Behrmann 2001] Gerd Behrmann, Alexandre David, Kim G. Larsen, Oliver Möller, Paul Pettersson and Wang Yi. UPPAAL - *Present and Future*. In Proc. of 40*th* IEEE Conference on Decision and Control. IEEE Computer Society Press, 2001. (Cited on page 134.)

[Benveniste 2003] Albert Benveniste, Paul Caspi, Stephen A. Edwards, Nicolas Halbwachs, Paul Le Guernic and Robert De Simone. *The synchronous languages twelve years later*. In Proceedings of the IEEE, 2003. (Cited on page 30.)

[Berthaut 2010] Florent Berthaut, Myriam Desainte-Catherine and Martin Hachet. *DRILE: an immersive environment for hierarchical live-looping*. In Proc. of New Interfaces for Musical Expression (NIME) 2010, 2010. (Cited on page 13.)

[Beurivé 2001] Anthony Beurivé and Myriam Desainte-Catherine. *Representing Musical Hierarchies with Constraints*. In 7th International Conference on Principles and Practice of Constraint Programming, Musical Constraints Workshop, Paphos, 2001. (Cited on pages 17, 36 and 71.)

[Biundo 2004] Susanne Biundo, Thom Frühwirth and Günther Palm. *Mining Hierarchical Temporal Patterns in Multivariate Time Series*. In Advances in Artificial Intelligence '04, pages 127–140, 2004. (Cited on page 36.)

[Bordeaux 2011] Lucas Bordeaux, George Katsirelos, Nina Narodytska and Moshe Y. Vardi. *The Complexity of Integer Bound Propagation*. Journal of Artificial Intelligence Research (JAIR), vol. 40, pages 657–676, 2011. (Cited on pages 47 and 87.)

[Boyer 1999] Marc Boyer and Michel Diaz. *Non Equivalence between Time Petri Nets and Time Stream Petri Nets*. In Proc. of the The 8th International Workshop on Petri Nets and Performance Models, number 98 of 1, Washington, DC, USA, 1999. IEEE Computer Society. (Cited on pages 33 and 134.)

[Bozga 1998] Marius Bozga, Conrado Daws, Oded Maler, Alfredo Olivero, Stavros Tripakis and Sergio Yovine. *Kronos: A model-checking tool for real-time systems*. In Anders Ravn and Hans Rischel, editors, Formal Techniques in Real-Time and Fault-Tolerant Systems, volume 1486 of *Lecture Notes in Computer Science*, pages 298–302. Springer Berlin / Heidelberg, 1998. (Cited on page 134.)

[Bresson 2005] Jean Bresson, Carlos Agón and Gérard Assayag. *OpenMusic 5: A Cross-Platform Release of the Computer-Assisted Composition Environment*. In 10th Brazilian Symposium on Computer Music, Rio de Janeiro, Brazil, 2005. Brazilian Computing Society. (Cited on page 37.)

[Bresson 2011] Jean Bresson, Carlos Agon and Gérard Assayag. *OpenMusic: visual programming environment for music composition, analysis and research*. In Proceedings of the 19th ACM international conference on Multimedia, MM '11, pages 743–746, New York, NY, USA, 2011. ACM. (Cited on page 37.)

[Byg 2009] Joakim Byg, Kenneth Yrke Jørgensen and Jiří Srba. *TAPAAL: Editor, Simulator and Verifier of Timed-Arc Petri Nets*. In Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA) '09, pages 84–89, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 134.)

[Cassez 2008] Franck Cassez and Olivier Roux. *From Time Petri Nets to Timed Automata*. Petri Net, Theory and Applications, vol. 1, pages 1–10, 2008. (Cited on page 134.)

[Cont 2008] Arshia Cont. *ANTESCOFO: Anticipatory Synchronization and Control of Interactive Parameters in Computer Music*. In Proc. of ICMC '08, 2008. (Cited on page 29.)

[Cormen 2001] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest and Charles E. Leiserson. Introduction to algorithms. McGraw-Hill Higher Education, 2nd édition, 2001. (Cited on page 69.)

[Couprie 1999] Pierre Couprie. *Three analysis models for L'oiseau moqueur, one of the Trois rêves d'oiseau François Bayle*. Org. Sound, vol. 4, no. 1, pages 3–14, January 1999. (Cited on page 9.)

[Dahan 2008] Kevin Dahan and Marin Laliberté. *Réflexions autour de la question d'interprétation de la musique électroacoustique*. In Proc. of JIM, 2008. (Cited on page 10.)

[de Boer 2000] F. S. de Boer, M. Gabbrielli and M. C. Meo. *A timed concurrent constraint language*. Journal of Information and Computation, vol. 161, no. 1, pages 45–83, 2000. (Cited on page 135.)

[Dechter 1991] Rina Dechter, Itay Meiri and Judea Pearl. *Temporal Constraint Networks*. Artif. Intell., vol. 49, no. 1-3, pages 61–95, 1991. (Cited on pages 36, 69 and 80.)

[Demri 2007] Stéphane Demri and Deepak D'Souza. *An automata-theoretic approach to constraint LTL*. Inf. Comput., vol. 205, pages 380–415, March 2007. (Cited on page 53.)

[Desainte-Catherine 2003] M. Desainte-Catherine and N. Brousse. *Towards a Specification of Musical Interactive Pieces*. In Proc. of the XIX Colloquium on Musical Informatics (CIM), Firenze, Italy, 2003. (Cited on pages 17 and 36.)

[Desainte-Catherine 2012] Myriam Desainte-Catherine, Antoine Allombert and Gérard Assayag. *Towards a hybrid temporal paradigm for musical composition and performance: The case of musical interpretation*. Computer Music Journal, vol. To appear in fall, 2012. (Cited on page 9.)

[Echeveste 2011] José Echeveste, Arshia Cont, Jean-Louis Giavitto and Florent Jacquemard. *Formalisation des relations temporelles entre une partition et une performance musicale dans un contexte d'accompagnement automatique : Accompagnement musical automatique*. In Journal Européen des Systèmes Automatisés, 2011. (Cited on pages 11 and 29.)

[Eker 2003] Johan Eker, Jorn Janneck, Edward A. Lee, Jie Liu, Xiaojun Liu, Jozsef Ludvig, Sonia Sachs and Yuhong Xiong. *Taming heterogeneity - the Ptolemy approach*. In Proceedings of the IEEE, 91(1):127-144, 2003. (Cited on page 31.)

[Erfle 1993] Robert Erfle. *Specification of temporal constraints in multimedia documents using HyTime*. Electronic Publishing, vol. 6, pages 397–411, 1993. (Cited on page 124.)

[Falaschi 2006] Moreno Falaschi and Alicia Villanueva. *Automatic verification of timed concurrent constraint programs*. Theory and Practice of Logic Program, vol. 6, no. 4, pages 265–300, 2006. (Cited on page 135.)

[Falaschi 2007] M. Falaschi, C. Olarte, C. Palamidessi and F. Valencia. *Declarative Diagnosis of Temporal Concurrent Constraint Programs*. In Véronica Dahl and Ilkka Niemelä, editors, Logic Programming, volume 4670 of *Lecture Notes in Computer Science*, pages 271–285. Springer Berlin Heidelberg, 2007. (Cited on page 135.)

[Fernández 2004] D. Fernández and J. Quintero. Vin: A visual language based on the ntcc calculus (in spanish). Master's thesis, Department of Computer Science and Engineering, Pontifica Universidad Javeriana, Cali, 2004. (Cited on page 142.)

[Forget 2009] Julien Forget. *Un Langage Synchrone pour les Systèmes Embarqués Critiques Soumis à des Contraintes Temps Réel Multiples*. PhD in Computer science, Université de Toulouse, Toulouse, Novembre 2009. (Cited on page 30.)

[Garavel 2008] Hubert Garavel. *Reflections on the Future of Concurrency Theory in General and Process Calculi in Particular*. Electron. Notes Theor. Comput. Sci., vol. 209, pages 149–164, 2008. (Cited on page 34.)

[Gardey 2005] Guillaume Gardey, Didier Lime, Morgan Magnin and Olivier (H.) Roux. *Roméo: A Tool for Analyzing time Petri nets*. In 17th International Conference on Computer Aided Verification (CAV'05), volume 3576 of *Lecture Notes in Computer Science*, pages 418–423, Edinburgh, Scotland, UK, 2005. Springer. (Cited on page 134.)

[Gautier 1987] Thierry Gautier, Paul Le Guernic and Löic Besnard. *SIGNAL: A declarative language for synchronous programming of real-time systems*. In Proc. of FPCA '87, 1987. (Cited on page 30.)

[Gennari 1998] Rosella Gennari. *Temporal Resoning and Constraint Programming - A Survey*. CWI Quaterly, vol. 11, pages 3–163, 1998. (Cited on pages 35 and 61.)

[Gräf 2007] Albert Gräf. *Interfacing Pure Data with Faust*. In Proc. of the 5th International Linux Audio Conference (LAC) '07, 2007. (Cited on pages 30 and 54.)

[Gutiérrez 2007] Julian Gutiérrez, Jorge A. Pérez, Camilo Rueda and Frank D. Valencia. *Timed Concurrent Constraint Programming for Analyzing Biological Systems*. Electron. Notes Theor. Comput. Sci., vol. 171, no. 2, pages 117–137, 2007. (Cited on page 20.)

[Halbwachs 1991] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud. *The synchronous data flow programming language LUSTRE*. In Proceedings of the IEEE, volume 79, pages 1305–1320, September 1991. (Cited on page 30.)

[Halbwachs 1994] Nicolas Halbwachs, Fabienne Lagnier and Christophe Ratel. *Programming and verifying real-time systems by means of the synchronous data-flow language LUSTRE*. IEEE Transaction on Software Engineering - Special issue: specification and analysis of real-time systems, vol. 18, pages 785–793, 1994. (Cited on page 30.)

[Haury 2008] Jean Haury. *La grammaire de l'exécution musicale au clavier et le mouvement des touches*. Documents numériques, vol. 11, no. 3-4, pages 127–148, 2008. (Cited on pages 28 and 37.)

[Holzmann 1997] Gerard J. Holzmann. *The Model Checker SPIN*. Software Engineering, vol. 23, no. 5, pages 279–295, 1997. (Cited on page 134.)

[Janin 2012] David Janin. *Modélisation compositionelle des structures rythmiques: une exploration didactique.* Revue Francophone d'Informatique Musicale, vol. 2. To appear in Fall, 2012. (Cited on pages 111 and 150.)

[Jouvelot 2011] Pierre Jouvelot and Yann Orlarey. *Dependent vector types for data structuring in multirate Faust*. Computer Languages, Systems and Structures, 2011. (Cited on pages 30, 54 and 55.)

[Keil 2006] Geert Keil. *La cause d'un événement. Eléments d'une métaphysique descriptive de la causalité entre événements*. Revue de Philosophie: Causalité, no. 89, pages 1–10, April 2006. (Cited on page 12.)

[Köhler 2003] Michael Köhler, Daniel Moldt and Heiko Rölke. *Modelling mobility and mobile agents using nets within nets*. In Proc. of the 24th international conference on Applications and theory of Petri nets (ICATPN) '03, pages 121–139, Berlin, Heidelberg, 2003. Springer-Verlag. (Cited on page 38.)

[Kurtag 2007] Gyorgy. Kurtag, Myriam. Desainte-Catherine, Jean-Louis Di Santo and Philippe Guillem. *Pédagogie de l'électroacoustique, du geste musical à la composition assistée par ordinateur*. In Proc. of the Journées d'Informatique Musicale (JIM) '07, 2007. (Cited on page 153.)

[Labiak 2004] G. Labiak and P. Miczulski. *UML Statecharts and Petri nets - Model Comparison for System Level Modelling*. Technical report, University of Zielona Góra, Poland, 2004. (Cited on page 34.)

[Labiak 2008] G. Labiak and M. Adamski. *Concurrent processes synchronisation in statecharts for FPGA implementation*. In Design Test Symposium (EWDTS), 2008 East-West, pages 59 –64, oct. 2008. (Cited on page 33.)

[Langerak 1992] Rom Langerak. *Bundle event structures: a non-interleaving semantics for LOTOS*. In Michel Diaz and Roland Groz, editors, Proc. of the Fifth International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE) '92, volume C-10 of *IFIP Transactions*, pages 331–346, Twente, Holland, 1992. North-Holland. (Cited on pages 102, 146 and 151.)

[Laudisa 2006] Federico Laudisa. *Le principe de causalité entre empirisme logique et néokantisme*. Revue de Philosophie: Causalité, no. 89, pages 11–20, April 2006. (Cited on page 13.)

[Lee 2001] Edward A. Lee, C. Hylands, J. Janneck, J. Davis II, J. Liu, X. Liu, S. Neuendorffer, S. Sachs M. Stewart, K. Vissers and P. Whitaker. *Overview of the Ptolemy Project*. Technical report UCB/ERL M01/11, EECS Department, University of California, Berkeley, 2001. (Cited on page 31.)

[Lee 2005] Edward A. Lee and Haiyang Zheng. *Operational Semantics of Hybrid Systems*. In Hybrid Systems: Computation and Control: 8th International Workshop, HSCC, LNCS 3414, Zurich, Switzerland, March 9-11, 2005, 2005. (Cited on page 31.)

[López 2006] Hugo A. López, Catuscia Palamidessi, Jorge Andrés Pérez, Camilo Rueda and Frank D. Valencia. *A Declarative Framework for Security: Secure Concurrent Constraint Programming*. In ICLP, pages 449–450, 2006. (Cited on page 20.)

[Maniatakos 2010] Fivos Maniatakos, Gérard Assayag, Frederic Bevilacqua and Carlos Agón. *On the architecture and formalisms for computer-assisted improvisation*. In Proc. of Sound and Music Computing (SMC), 2010. (Cited on page 28.)

[Marczak 2011] Raphaël Marczak, Antoine Allombert and Myriam Desainte-Catherine. *Real-Time Temporal Control of Musical Processes*. In Proc. of the International Conferences on Advances in Multimedia (MMEDIA) '11, 2011. (Cited on page 39.)

[Martello 1990] Silvano Martello and Paolo Toth. Knapsack problems: algorithms and computer implementations. John Wiley & Sons, Inc., New York, NY, USA, 1990. (Cited on page 68.)

[Meiri 1996] Itay Meiri. *Combining Qualitative and Quantitative Constraints in Temporal Reasoning*. Artificial Inteligence, vol. 87, no. 1-2, pages 343–385, 1996. (Cited on pages 36, 61, 62 and 78.)

[Mihalic 2011] Alexander Mihalic and Laurent Pottier. *Migrer des œuvres avec électronique temps réel vers FAUST*. In Proc. of Journées d'informatique musical (JIM) '11, 2011. (Cited on page 149.)

[Milner 1999] Robin Milner. Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press, 1st édition, June 1999. (Cited on pages 31 and 136.)

[Müller-Olm 1999] Markus Müller-Olm, David Schmidt and Bernhard Steffen. *Model-checking: A tutorial introduction*. In In Proceedings of the 6th static analysis symposium, pages 330–354. Springer, 1999. (Cited on page 134.)

[Muñoz 2004] Pilar Muñoz and Andrés Hurtado. *Programming robot devices with a timed concurrent constraint programming*. In In Principles and Practice of Constraint Programming (CP) '4. LNCS 3258, page 803. Springer, 2004. (Cited on pages 19 and 115.)

[Muscettola 1998] Nicola Muscettola, Paul H. Morris and Ioannis Tsamardinos. *Reformulating Temporal Plans for Efficient Execution*. In Proc. of Principles of Knowledge Representation and Reasoning, pages 444–452, 1998. (Cited on pages 38, 70, 80 and 145.)

[Nielsen 2002] M. Nielsen, C. Palamidessi and F. Valencia. *Temporal Concurrent Constraint Programming: Denotation, Logic and Applications*. Nordic Journal of Computing, vol. 1, no. 9, pages 145–188, 2002. (Cited on pages 16, 35, 47, 49, 52 and 136.)

[Nyman 1999] Michael Nyman. Experimental music: Cage and beyond, Chapter 1, pages 1–50. Cambridge University Press, London, UK, second édition, 1999. (Cited on pages 4 and 10.)

[Olarte 2008] Carlos Olarte, Camilo Rueda and Frank Valencia. *Concurrent constraint programming: Calculi, languages and emerging applications*. In Newsletter of the ALP, volume 21, 2008. (Cited on page 34.)

[Olarte 2009a] Carlos Olarte. *Universal temporal concurrent constraint programming*. PhD in Computer science, École Polytechnique, Palaiseau, September 2009. (Cited on page 138.)

[Olarte 2009b] Carlos Olarte and Camilo Rueda. *A Declarative Language for Dynamic Multimedia Interaction Systems*. In Proc. of Mathematics and Computation in Music, volume 38, Berlin, Germany, july 2009. Springer. (Cited on pages 19, 34 and 38.)

[Olarte 2011] Carlos Olarte, Camilo Rueda, Gerardo Sarria, Mauricio Toro and Frank Valencia. *Concurrent Constraints Models of Music Interaction*. In Gérard Assayag and Charlotte Truchet, editors, Constraint Programming in Music, Chapter 6, pages 133–153. Wiley, Hoboken, NJ, USA., 2011. (Cited on pages 19, 24 and 34.)

[Orlarey 2004] Yann Orlarey, Dominique Fober and Stephane Letz. *Syntactical and semantical aspects of Faust*. Soft Comput., vol. 8, no. 9, pages 623–632, 2004. (Cited on pages 16, 54, 55 and 108.)

[Orlarey 2010] Yann Orlarey, Dominique Fober and Stephane Letz. *Work Stealing Scheduler for Automatic Parallelization in Faust*. In Proc. of Linux Audio Conference, 2010. (Cited on pages 54 and 121.)

[Pabón 2003] M. Pabón, F. Rocha and J. Chalá. *Developing a compiler for Ntcc (in spanish)*. Technical report 2003-1, Department of Computer Science and Engineering, Pontifica Universidad Javeriana, Cali, Colombia, 2003. (Cited on page 141.)

[Pachet 2002] François Pachet. *Playing with Virtual Musicians: the Continuator in Practice*. IEEE Multimedia, vol. 9, pages 77–82, 2002. (Cited on pages 28, 119 and 121.)

[Percival 2008] Graham Percival, Torsten Anders and George Tzanetakis. *Generating Targeted Rhythmic Exercises for Music Students with Constraint Satisfaction Programming*. In Proceedings of the 2008 International Computer Music Conference, Belfast, UK, 2008. (Cited on page 153.)

[Planken 2010] Léon Planken, Mathijs de Weerdt and Neil Yorke-Smith. *Incrementally Solving STNs by Enforcing Partial Path Consistency.* In Proc. of International Conference on Automated Planning and Scheduling (ICAPS) '10, pages 129–136, 2010. (Cited on page 70.)

[Pnueli 1977] A. Pnueli. *The temporal logic of programs*. In Proc. of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77), pages 46-57. IEEE, IEEE Computer Society Press, 1977., 1977. (Cited on page 134.)

[Popova-Zeugmann 1999] Louchka Popova-Zeugmann. *On Liveness and Boundedness in Time Petri Nets*. Technical report, Humboldt-Universität zu Berlin, 1999. (Cited on page 33.)

[Pratt 1992] Vaughan R. Pratt. *The Duality of Time and Information*. In Proc. of CONCUR'92, LNCS 630, pages 237–253. Springer-Verlag, 1992. (Cited on page 14.)

[Puckette 1996] M. Puckette. *Pure Data*. In Proceedings of the International Computer Music Conference. San Francisco 1996, 1996. (Cited on page 7.)

[Puckette 1998] Miller Puckette, Theodore Apel and David Zicarelli. *Real-time audio analysis tools for Pd and Max/MSP*. In Proc. of ICMC '98, Ann Arbor, USA, 1998. (Cited on pages 7, 10 and 29.)

[Puckette 2002] Miller Puckette. *Using Pd as a score language*. In Proc. of ICMC '02, pages 184–187, New York, NY, USA, 2002. (Cited on page 10.)

[Ranaivoson 2009] Nirina Ranaivoson. Réflexion sur la mise en place des structures logiques dans un logiciel de partitions musicales interactives. Master's thesis, Université de Bordeaux, 2009. (Cited on pages 39 and 85.)

[Robine 2007] Matthias Robine. *Analyse automatique du doigté au piano*. In Proceedings of the Journées dInformatique Musicale (JIM) '07, Lyon, France, 2007. (Cited on page 153.)

[Roy 2004] Peter Van Roy. *Multiparadigm Programming in Mozart/Oz*. In Second International Conference (MOZ) '04, volume 3389 of *Lecture Notes in Computer Science*, Charleroi, Belgium, October 2004. Springer. (Cited on page 35.)

[Rueda 2001] Camilo Rueda and Frank D. Valencia. *Formalizing Timed Musical Processes with a Temporal Concurrent Constraint Programming Calculus*. In Proc. of Musical Constraints Workshop in Theory and Practice of Constraint Programming (CP) '01, 2001. (Cited on pages 19, 20, 34, 71, 133 and 136.)

[Rueda 2002] Camilo Rueda and Frank Valencia. *Proving Musical Properties using a temporal Concurrent Constraint Calculus*. In Proc. of the 28th International Computer Music Conference (ICMC) '02, 2002. (Cited on pages 19, 20, 34, 35, 133 and 136.)

[Rueda 2004] C. Rueda and F. Valencia. *On validity in modelization of musical problems by CCP*. Soft Computing, vol. 8, no. 9, pages 641–648, 2004. (Cited on pages 19, 20, 34, 35, 133, 136 and 152.)

[Rueda 2005a] Camilo Rueda and Carlos Olarte. *Using Stochastic NTCC to Model Biological Systems*. In CLEI 2005 (31st Latinoamerican Conference on Informatics), 2005. (Cited on page 20.)

[Rueda 2005b] Camilo Rueda and Frank Valencia. *A temporal concurrent constraint calculus as an audio processing framework*. In SMC '05, 2005. (Cited on pages 19, 34, 112 and 151.)

[Rueda 2006] Camilo Rueda, Gérard Assayag and Shlomo Dubnov. *A Concurrent Constraints Factor Oracle Model for Music Improvisation*. In Proc. of the XXXII Conferencia Latinoamericana de Informática (CLEI) '06, 2006. (Cited on pages 19, 34 and 115.)

[Russel 2006] Bertrand Russel. *Sur la notion de cause*. Revue de Philosophie: Causalité, no. 89, pages 20–30, April 2006. (Cited on page 12.)

[Safra 1988] S. Safra. *On the complexity of omega-automata*. In Proceedings of the 29th Annual Symposium on Foundations of Computer Science (SFCS) '88, pages 319–327, Washington, DC, USA, 1988. IEEE Computer Society. (Cited on page 136.)

[Sangiorgi 2012] Davide Sangiorgi. Introduction to bisimulation and coinduction. Cambridge University Press, 2012. (Cited on page 34.)

[Saraswat 1992] Vijay A. Saraswat. Concurrent constraint programming. MIT Press, Cambridge, MA, 1992. (Cited on pages 34 and 47.)

[Saraswat 1994] Vijay A. Saraswat, Radha Jagadeesan and Vineet Gupta. *Foundations of Timed Concurrent Constraint Programming*. In Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science, pages 71–80. IEEE Computer Press, 1994. (Cited on pages 35 and 135.)

[Sarria 2008] Gerardo Sarria. *Formal Models of Timed Musical Processes*. PhD thesis, Universidad del Valle, Colombia, 2008. (Cited on pages 38, 121 and 147.)

[Schimpf 2009] Alexander Schimpf, Stephan Merz and Jan-Georg Smaus. *Construction of Büchi Automata for LTL Model Checking Verified in Isabelle/HOL*. In Proceedings of the 22nd International Conference on Theorem Proving in Higher Order Logics, TPHOLs '09, pages 424–439, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 136.)

[Schwer 2005] Sylvaine R. Schwer. *Quel modèle mathématique pour la temporalité?* Technical report, Laboratoire d'Informatique de l'Université Paris-Nord, 2005. (Cited on page 11.)

[Sénac 1995] Patrick Sénac, Pierre de Saqui-Sannes and Roberto Willrich. *Hierarchical Time Stream Petri Net: A Model for Hypermedia Systems*. In Proc. of the 16th International Conference on Application and Theory of Petri Nets, pages 451–470, London, UK, 1995. Springer-Verlag. (Cited on pages 33 and 134.)

[Sipser 1996] Michael Sipser. Introduction to the theory of computation, Chapter 7. PWS Publishing Company, United States of America, 1996. (Cited on page 68.)

[Soonhoi 1999] Dohyung Kim Soonhoi, Dohyung Kim and Soonhoi Ha. *Asynchronous Interaction between FSM and Dataflow Models*. In Proc. of the international Conference on VLSI (Very Large-Scale Integration) and CAD (Computer-Aided Design) [ICVC] '99, 1999. (Cited on page 31.)

[Stephens 1997] Robert Stephens. *A survey of stream processing*. Acta Informatica, vol. 34, pages 491–541, 1997. 10.1007/s002360050095. (Cited on page 29.)

[Steyn 2004] Jacques Steyn. *Prerequisites for an XML-based music markup language - an exercise in the design of an XML language*. In Proc. of 4th Open Workshop of MUSICNETWORK: Integration of Music in Multimedia Applications, 2004. (Cited on page 123.)

[T. Sjoland 2001] S. Haridi T. Sjoland Erik Klintskog. *An interpreter for Timed Concurrent Constraints in Mozart*. Technical report, Swedish Institute of Computer Science, 2001. (Cited on page 35.)

[Tack 2009] Guido Tack. *Constraint Propagation - Models, Techniques, Implementation*. PhD thesis, Saarland University, Germany, 2009. (Cited on pages 19, 116 and 140.)

[Teehan 2007] Paul Teehan, Mark R. Greenstreet and Guy G. Lemieux. *A Survey and Taxonomy of GALS Design Styles*. IEEE Design & Test of Computers, vol. 24, no. 5, pages 418–428, 2007. (Cited on page 32.)

[Tini 1999] Simone Tini. *On the Expressiveness of Timed Concurrent Constraint Programming*. In Electronics Notes in Theoretical Computer Science, 1999. (Cited on page 35.)

[Toro 2008] Mauricio Toro. *Exploring the possibilities and limitations of Concurrent Programming for Multimedia Interaction and graphical representations to solve musical CSP's*. Technical report 2008-3, Ircam, Paris.(FRANCE), 2008. (Cited on page 117.)

[Toro 2009] Mauricio Toro, Carlos Agón, Gérard Assayag and Camilo Rueda. *Ntccrt: A concurrent constraint framework for real-time interaction*. In Proc. of ICMC '09, Montreal, Canada, 2009. (Cited on pages 17, 19, 34, 35, 115, 116 and 147.)

[Toro 2010a] Mauricio Toro. *Structured Musical Interactive Scores (short)*. In Proc. of the doctoral consortium in ICLP '10, Edinburgh, Scotland, UK, 2010. (Cited on pages 25 and 152.)

[Toro 2010b] Mauricio Toro and Myriam Desainte-Catherine. *Concurrent Constraint Conditional Branching Interactive Scores*. In Proc. of SMC '10, Barcelona, Spain, 2010. (Cited on pages 18, 22, 24, 85, 98, 102, 118, 119 and 146.)

[Toro 2010c] Mauricio Toro, Myriam Desainte-Catherine and P. Baltazar. *A Model for Interactive Scores with Temporal Constraints and Conditional Branching*. In Proc. of Journées d'Informatique Musical (JIM) '10, May 2010. (Cited on pages 18, 22, 24, 85, 98, 102 and 146.)

[Toro 2012a] Mauricio Toro, Myriam Desainte-Catherine and Julien Castet. *An Extension of Interactive Scores for Multimedia Scenarios with Temporal Relations for Micro and Macro Controls*. In Proc. of Sound and Music Computing (SMC) '12, Copenhagen, Denmark, July 2012. (Cited on pages 11, 22, 24, 106, 109, 119 and 120.)

[Toro 2012b] Mauricio Toro, Myriam Desainte-Catherine and Camilo Rueda. *Formal semantics for interactive music scores: A framework to design, specify properties and execute interactive scenarios*. Journal of Mathematics and Music, vol. To be published., 2012. (Cited on pages 19, 21, 22, 24, 34, 66, 67 and 83.)

[Truchet 2003] Charlotte Truchet, Gérard Assayag and Philippe Codognet. *OMClouds, a heuristic solver for musical constraints*. In Proc. of the Metaheuristics International Conference (MIC) '03, Kyoto, Japan, 2003. (Cited on page 153.)

[Tsamardinos 2001] Ioannis Tsamardinos, Martha E. Pollack and Philip Ganchev. *Flexible Dispatch of Disjunctive Plans*. In European Workshop on Planning, 2001. (Cited on page 72.)

[Valencia 2002] Frank D. Valencia. *Temporal Concurrent Constraint Programming*. PhD thesis, University of Aarhus, 2002. (Cited on pages 47, 51 and 87.)

[Valencia 2005] Frank D. Valencia. *Decidability of infinite-state timed CCP processes and first-order LTL*. Journal of Theoretical Computer Science - Expressiveness in concurrency, vol. 330, no. 3, pages 557–607, 2005. (Cited on pages 52, 53, 134, 136 and 171.)

[Van Roy 2004] Peter Van Roy and Seif Haridi. Concepts, techniques, and models of computer programming. MIT Press, March 2004. (Cited on page 35.)

[Vardi 2001] Moshe Vardi. *Branching vs. Linear Time: Final Showdown*. In Proceedings of the 2001 Conference on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001 (LNCS Volume 2031, pages 1–22. Springer-Verlag, 2001. (Cited on page 14.)

[Vickery 2003] Lindsay Vickery. *Non-linear structures for real-time interactive musical works*. In Proceedings of the Australasian Computer Music Conference (ACMC) '03, 2003. (Cited on pages 5, 18, 38 and 85.)

[Vickery 2004] Lindsay Vickery. *Interactive control of higher order musical structures*. In Proc. of ACMC '04, Victoria University, New Zealand, July 2004. ACMA. (Cited on pages 6 and 15.)

[Vickery 2011] Lindsay Vickery. *The possibilities of novel format structures through computer controlled live performance*. In Proc. of ACMC '11, 2011. (Cited on pages 5 and 6.)

[Virbitskaite 2008] I. B. Virbitskaite and R. S. Dubtsov. *Semantic domains of timed event structures*. Programming and Computer Software, vol. 28, page 3, 2008. (Cited on page 43.)

[Wirag 1995] Stefan Wirag, Kurt Rothermel and Thomas Wahl. *Modelling Interaction with HyTime*. In Proc. of GI/ITG Kommunikation in Verteilten Systemen, pages 188–202, 1995. (Cited on page 124.)

[Xu 2003] Lin Xu and Berthe Y. Choueiry. *A New Efficient Algorithm for Solving the Simple Temporal Problem*. In International Syposium on Temporal Representation and Reasoning, page 212, Los Alamitos, CA, USA, 2003. IEEE Computer Society. (Cited on page 70.)

[Yamauchi 2007] Takuya Yamauchi and Toru Iwatake. *An Interactive Installation through Spatial Sensing*. In Proc. of the 4th International Mobile Music Workshop, May 2007. (Cited on page 7.)

Part IV

Appendices

APPENDIX A

Proofs

In what follows, we present a detailed proof of correctness of the hierarchical model of interactive scores.

A.1 Correctness of the Operational Semantics of Chapter 6

It was proven by Valencia *et al.* in [Valencia 2005] that the strongest post-condition of a process, whose guards do not depend on local variables, can be translated into a Büchi automaton, and a *constraint linear-time logic (*CLTL*)* formula can be translated into a process, as we described in Section 4.2. The reader may find the encoding of the denotation of `ntcc` into Büchi in [Valencia 2005]. The encoding also defines a finite set of constraints (i.e., *the relevant constraints of P*) because the alphabet of a Büchi automaton must be finite.

We also recall from Chapter 4 that the infinite sequences of the denotation of a process can be represented as a Büchi automaton. Büchi automata are an extension of finite state automata for infinite input. Such automata accept exactly those runs in which at least one of the infinitely often occurring states is an accepting state. For simplicity we label the transitions as $c \vdash d$ (i.e., there is a transition for every constraint $c \in C_{finite}$ stronger than d).

As in Chapter 6, we use the notation $T + \Delta =^{def} \{t' | t' = t + \delta, t \in T, \delta \in \Delta\}$ for temporal constraints of duration.

Proof. We shall proceed by induction over the structure of ε^*. We prove the proposition by contradiction. We suppose that there is at least a tuple in $T_1 \times T_2 \ldots \times T_n$ that is not a solution of $\mathrm{tc}(\varepsilon^*)$. We do not consider the process $User_i$ in the proof because its denotation is the set of all the possible sequences of stores (i.e., constraints). The constraint $launch_i$ can only be deduced in a single time-unit because after such time unit, the constraint $\neg launch_i$ is added in all subsequent stores.

1. **A single interactive object.** The encoding of a score with a single interactive object has one event e_i, no intervals, and $\mathrm{tc}(\varepsilon^*) = t_i \in \mathbb{N}$.

   Figure A.1 presents the denotation of $iPoint_{i,\mathrm{Pr}}$ and Figure A.3 the denotation of $Clock(0)$. In what follows we analyze $[\![Score_{I,S,\mathrm{Pr},R}]\!]$, which is the intersection of the denotations of $iPoint_{i,\mathrm{Pr}}, Clock(0)$ and **!tell**$(0 < p_i \leq n_\infty)$. Since e_i has no predecessors, the constraint $\bigwedge_{j \in \mathrm{Pr}} launched_j$ can always be deduced. Note that whenever ev can be deduced (after the first time unit), $launch_i$ can also be deduced. Finally, $clock + 1 > p_i$ can be deduced when $clock = n_\infty - 1$ because $0 < p_i < n_\infty$ can always be deduced, thus if $clock + 1 > p_i$ can be deduced, $launch_i$ can be deduced at last in the position n_∞; therefore, $T_i = [1, n_\infty]$

2. **Two points.** The encoding of a score with one temporal object has two events e_i and e_j, an event delay $e_i \mapsto^{\Delta} e_j$ and $\text{tc}(\varepsilon^*) = t_i \in \mathbb{N} \wedge t_j \in \mathbb{N} \wedge t_j \in t_i + \Delta$.

   (a) **Two interactive objects.** We analyze $[Score_{I,S,\text{Pr},R}]$, which is the intersection of the denotations of $iPoint_{i,\text{Pr}}$, $iPoint_{j,\text{Pr}}$, $Clock(0)$ and !**tell**$(0 < p_i \leq n_\infty)$. Since point i has no predecessors, the constraint $\bigwedge_{j \in \text{Pr}(i)} launched_j$ can always be deduced. Whenever ev_i can be deduced (after the first time unit), $launch_i$ can also be deduced. Analogically, whenever ev_j can be deduced, $launch_j$ can also be deduced, and this can only happen after $launched_i$ can be deduced. The constraint $clock + 1 > p_i$ can be deduced for values of p_i that satisfy the constraints $0 < p_i < n_\infty$ and $p_j = p_i + \Delta$. Finally, $clock + 1 > p_j$ can be deduced when at last $clock = n_\infty - 1$ can be deduced because $0 < p_j < n_\infty$. Therefore, the values of T_i and T_j are given by $T_i \subseteq [1, n_\infty] \wedge T_j \subseteq [1, n_\infty] \wedge T_j \in T_i + \Delta$.

   (b) **Two static points.** We analyze $[\![Score_{I,S,\text{Pr},R}]\!]$ which is the intersection of the denotations of $sPoint_{i,\text{Pr}}$, $sPoint_{j,\text{Pr}}$, $Clock(0)$ and !**tell**$(0 < p_i \leq n_\infty)$. Figure A.2 presents the denotation of $sPoint_{i,\text{Pr}}$.

      i. Since e_i has no predecessors, the constraint $\bigwedge_{j \in \text{Pr}(i)} launched_j$ can always be deduced. The constraint $clock + 1 < p_i$ can be deduced in the first time unit; therefore, $launch_i$ will be deduced in the second time unit.

      ii. Once the constraint $launch_i$ can be deduced in the second time unit, it will be deduced from all subsequent sequences the constraints $launched_i$ and $p_i = 1$, thus the constraint $clock + 1 < p_j$ will be deduced from the store until $clock = 1 + \min(\Delta)$; therefore, the constraint $launch_j$ will be deduced in the time unit $1 + \min(\Delta)$, when $clock + 1 < p_j$ cannot be longer deduced.

      Therefore $T_i = \{1\}$ and $T_j = \{1 + \min(\Delta)\}$.

   (c) **A static point followed by an interactive object.** We analyze $[\![Score_{I,S,\text{Pr},R}]\!]$, which is the intersection of the denotations of $sPoint_{i,\text{Pr}}$, $iPoint_{j,\text{Pr}}$, $Clock(0)$ and !**tell**$(0 < p_i \leq n_\infty)$. The constraint $launch_i$ can be deduced in the first time unit for process $sPoint_{i,\text{Pr}}$, as it was described in part 2(b)i. The positions where $launch_j$ can be deduced are in the set $T_j = [1 + \min(\Delta), n_\infty)$; the reason is that the constraint $clock + 1 < p_j$ cannot be deduced before $clock = 1 + \min(\Delta)$ and the constraint $clock + 1 > p_j$ can only be deduced after $clock = n_\infty$.

   (d) **An interactive object followed by a static point.** We analyze $[\![Score_{I,S,\text{Pr},R}]\!]$, which is the intersection of the denotations of $iPoint_{i,\text{Pr}}$, $sPoint_{j,\text{Pr}}$, $Clock(0)$ and !**tell**$(0 < p_i \leq n_\infty)$. The positions where $launch_i$ can be deduced are the ones described in part 1. Once $launched_i$ can be deduced from a store, the constraint $launch_j$ will be deduced from all stores $\min(\Delta)$ time units after, because $clock + 1 < p_i$ cannot be deduced anymore; thus, $T_i = [1, n_\infty), T_j \in T_i + \min(\Delta)$.

3. **Inductive case.** We must prove that if the proposition holds for an event structure ε^* with k events, it also holds for ε'^*, which is ε^* with one more event. Let e_{k+1} be the new event in ε'^*, $\text{pred}(e_{k+1})$ the event delays between e_{k+1} and its predecessors and $\text{succ}(e_{k+1})$ the event delays between e_{k+1} and its successors. We know by inductive

hypothesis that the preposition holds for a ε^* with k events, we must prove that adding the event e_{k+1} to the events of ε^* and $\text{pred}(e_{k+1})$ and $\text{succ}(e_{k+1})$ to the event delays of ε^*, the proposition also holds. If the new event has no predecessors nor successors, the proof continues as part 1. Otherwise, for each predecessor and each successor, the proof continues as part 2 because the event structure is *dispatchable*, thus we can rely on the neighbors to dispatch an event. Therefore, all tuples in $T_1 \times T_2 ... \times T_{k+1}$ are solutions of $\text{tc}(\varepsilon'^*)$, which is a contradiction.

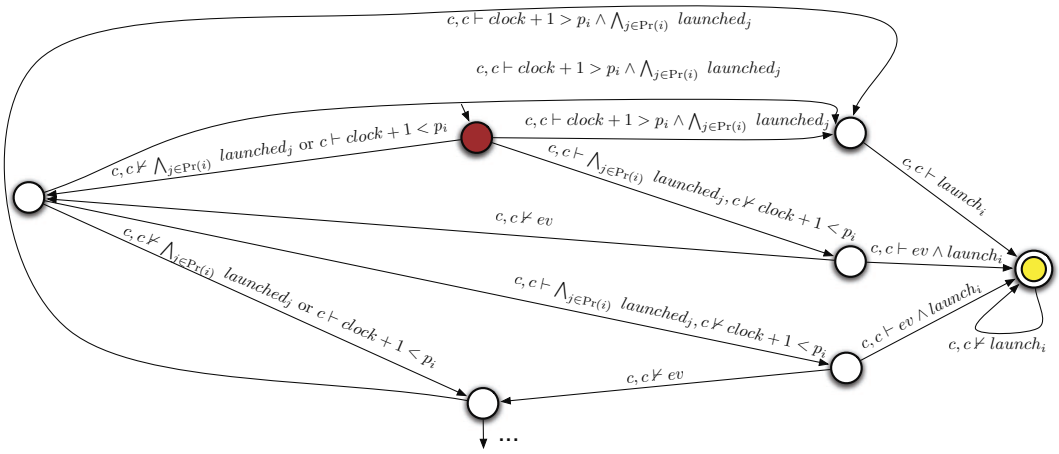

Figure A.1: Denotation of process $iPoint_{i,\Pr}$ represented as a Büchi automaton, $c \in S \subset_{fin} C$. Set S represents the relevant constraints and is a finite subset of C, the set of all possible constraints in the constraint system.

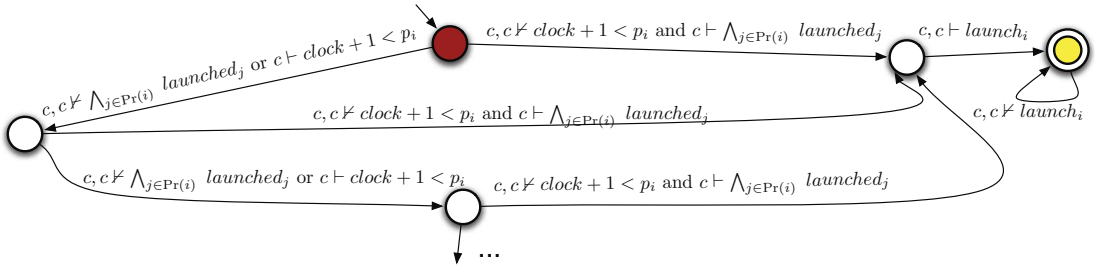

Figure A.2: Denotation of process $sPoint_{i,\Pr}$ represented as a Büchi automaton, $c \in S \subset_{fin} C$. Set S represents the relevant constraints and is a finite subset of C, the set of all possible constraints in the constraint system

□

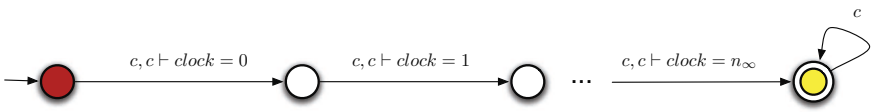

Figure A.3: Denotation of process $Clock(0)$ represented as a Büchi automaton, $c \in S \subset_{fin} C$. Set S represents the relevant constraints and is a finite subset of C, the set of all possible constraints in the constraint system