COMPUTING ENGINEERING

A survey of computational calculi used in musical applications

INGENIERIA DE SISTEMAS

Un estudio de los cálculos computacionales usados en aplicaciones musicales

§ Gerardo M. Sarria M.

Departamento de Electrónica y Ciencias de la Computación, Pontificia Universidad Javeriana, Cali-Colombia
§ gsarria@javerianacali.edu.co

(Recibido: 22 de Abril de 2013- Aceptado: 6 de Agosto de 2013)

Abstract
During the last decades, several formal models have been proposed to formalize musical applications, to solve musical and improvisation problems, and to prove properties in music. In this paper, we briefly describe some of those formal models (computational calculi). We provide a description of some applications of these formalisms, and discuss some considerations about each calculus mentioned here remarking strengths and weaknesses.

Keywords: *Computational calculi, formalization, musical applications.*

Resumen
En las últimas décadas muchos modelos formales han sido propuestos para formalizar aplicaciones musicales, para resolver problemas musicales y de improvisación, y para probar propiedades en la música. En este artículo describiremos brevemente algunos de estos modelos formales (los cálculos computacionales); proveeremos una descripción de algunas aplicaciones de dichos formalismos; finalmente discutiremos algunas consideraciones sobre cada cálculo mencionado aquí, resaltando fortalezas y debilidades.

Palabras clave: *Cálculos computacionales, Formalización, Aplicaciones Musicales*

1. Introduction

During the last fifty years, the term formalization has been increasingly used in process modeling. Formalization is a procedure to present scientific theories within the framework of a formal system, and it can be considered a deductive approach from a purely combinatorial point of view. In other words, formalization may be seen as a deductive step leading from a language to a theory.

Musical composition, performance and improvisation are complex tasks. They demand to define and control real-time concurrent activities. Musical objects can be seen as structures with various dimensions. In a horizontal dimension, for example, time becomes a tight notion where musical objects like notes or chords are constrained. The position of each object defines the relative order of musical events with respect to each other, forming rhythmic patterns. In a vertical dimension, event simultaneity and musical objects like voices running in parallel can be perceived, building harmony patterns.

Formalization, in musical terms, consists in clarifying phenomena such as analysis and composition, representing musical processes (human thoughts) in a formal language that can be understood by computers. That means, elaborating models of formal representations of musical concepts that can be transmitted to computers. It is believed that the complexity of musical processes is a challenge to any computational formalism. The development of computational models and tools to be used in musical systems has increased during the last decades. Simple and expressive formal models provide techniques for reasoning musical properties; they are useful in the construction of meaningful musical processes, which are the basis of high-level musical applications.

There are many programming languages for music, musical applications and tools based on mathematical principles, formal theories, and studies in computer science. In this paper, computational calculi that have been proposed to formalize some musical applications to solve musical and improvisation problems, and to prove properties in music are briefly described. Various known musical applications of those formalisms are also described.

2. Calculi

The first time that computer theory was used in western music was in the 13th century when a perforated card was introduced in a musical machine to make it play automatically (Roads, 1985). Late 19th century, as Peter Hanappe mentioned in (Hanappe, 1999), Ada Lovelace realized that the unachieved computing machine designed by Charles Babbage was able to manipulate symbols and numbers; thus, it could become a composing machine for several disciplines including music. Then, in the first decades of the twentieth century, Joseph Schillinger (Schillinger, 1948) predicted the use of computers in musical compositions. After that, the famous musicologists Lejaren Hiller and Leonard Isaacson composed *Illiac Suite for String Quartet* in the 1950s. This composition derived in computational music theories, music research, and engineering for automatic or algorithmic compositions.

The software Musicomp (Music Simulator Interpreter for Compositional Procedures) is perhaps the first software designed for assisting a composition (Assayag, 1998). Robert Baker created it around 1963 with Hiller’s expertise help. Later, technological advances as digital audio, personal computers, graphical interfaces, standards like MIDI and, above all, programming languages, the computational music paradigm was gradually defined.

Recently, research in computer music has focused in providing high-level expressions for music representation synthesis, and real-time control; as a result, languages, tools and computational formalisms were created. The latter provide data abstraction, and control flow paradigms, such as convenient methods for handling time flow, structural organization of musical materials, music data representation, hierarchy, machine improvisation, and those related to style simulation, and concurrency. The following are representative formalisms that have been directly or indirectly used to model musical scenarios.

2.1 λ-Calculus

A λ-calculus (Church, 1985) is a calculus for describing functions that compute values from arguments. It was introduced by Alonzo Church and Stephen Cole Kleene as part of a research on David Hilbert's Entscheidungsproblem, who aimed to find a general algorithm in which given a formal language and a mathematical statement in a language, the output would become True if the statement was true; otherwise, it would become False). The typed λ-calculus is a variant of the λ-calculus that makes the intended types of all expressions explicit. The type of an expression is determined from the types of its sub-expressions. The λ&-calculus (Castagna, 1998) is an extension of the λ≤-calculus (Cardelli and Abadi, 1996), which is the simple typed λ-calculus with a subtyping relation. The λ&-calculus is a formal model for the Common Lisp Object System, CLOS, (Steele, 1990). The following are programming languages based on or formalized with the λ-calculus or its variants.

Yann Orlarey et al., at Grame, have studied the λ-calculus (and functional programming) in order to show its capability to express musical functions and operations. In Orlarey et al. (1994), a music calculus was proposed by introducing abstraction and application concepts from λ-calculus to a descriptive language. The result is an approach to formalize composition activity, and it led to a visual musical language made based on the λ-calculus, called ***Elody*** (Orlarey et al., 1997). This language is a visual environment written in Java where musicians can construct musical objects and assemble them with other objects to create a composition. The main concept in Elody is the visual constructor, which is an interface to build new musical objects. The basic elements are notes and silences (musical expressions are built from them). Windows with boxes represent the visual constructors (arguments and a result). This application is appropriate for introducing programming principles to musicians and non-programmers in general. Programs can be seen as the combination of abstractions and applications (both λ-calculus concepts).

The λ&-calculus has been used in Agon (1998) and Assayag et al. (1998) to formalize ***OpenMusic*** (OM) (Assayag et al., 1999), a visual and object-oriented programming language based on CLOS developed by Gérard Assayag and Carlos Agon at Ircam. OpenMusic is a general-purpose application providing an environment to support musical composition by implementing a set of musical and computational objects symbolized by icons that can be dragged and dropped all around, and it came after PatchWork (Laurson, 1996). Programs in OpenMusic (called patches) are graphical algorithms constituted by boxes (icons that represent functions, classes, instances, etc.), and connections among them. Each patch has an associated Lisp code; that is, there is a flowchart within a patch, which graphically describes Lisp code accomplishing a specific function. Additionally, OpenMusic defines an original notion called Maquettes. They are OpenMusic entities for representing patches and scores in the same object (Agon, 2004). Inside a maquette, musical structures can be organized in a time line together with temporal relations, constraints and hierarchies.

Arctic (Dannenberg, 1984) is a high-level computational language developed by Roger Dannenberg. It synthesized ideas from functional programming to specify real-time control systems (real-time systems are modeled as black boxes with inputs and outputs). A program in Arctic is a higher-order function from a set of inputs to a set of outputs. The formal model on which Arctic is based lacks time, concurrency and synchronization. This problem is solved by borrowing details from other languages. For example, time is represented as functions that execute statements.

The ***Canon Score Language*** (Dannenberg, 1989) is another language developed by the same author, which combines Arctic and MIDI concepts. It takes primitive operators to create scores from them and make score transformations. Canon uses its declarative programming style, and its ability to define an abstract behavior to make compositions. As in Arctic, Canon can specify and manipulate time and synchronization.

Common Music (Taube, 1990) is an object-oriented music composition environment. Heinrich Taube created it for describing sound and its higher-level structure to compose. A composition process is divided in three different levels: developing musical ideas, translating ideas to real world, and understanding how to conceive these ideas. Common music provides collections to translate high-level information introduced by the user into lower-level information understandable by a synthesizer.

In Dannenberg et al. (1991), Roger Dannenberg worked on another language to synthesize sound and music composition called ***Fugue***. It extends the traditional approach to synthesize sound, using functional programming concepts. Fugue is used to design instruments by combining functions (similar to orchestra languages of Music-N family (Mathews et al., 1969)). These new instruments are used in expressions to generated sounds, and the expressions are combined into complex ones to create a whole composition. Fugue extended Canon to manipulate digital audio.

In Hudak et al. (1996), Paul Hudak proposed ***Haskore***, an algebraic formalism to describe music and compositions in Haskell programming language. It is a collection of musical modules (data types) to express music. A score and its components are defined separately from their performance (which is a temporally ordered sequence of musical events): some Music data types (such as notes and their combination) become the score, and various functions can be defined to interpret it to produce a performance.

Finally, ***BOOMS*** (Balaban et al., 2002; Barzilay, 1996) is a computer-music environment developed by Eli Barzilay. It is a general application framework for developing editors supporting a structural and regular editing combination, and an end-user abstraction as a tool to define reusable functions without programming. It is implemented in CLOS and features a sophisticated Windows interface. Although it is a general framework, it was conceived to be instantiated to domains similar to music composition.

2.2 Communicating sequential processes (CSP)

The Communicating Sequential Processes algebra (Hoare, 1978) is a model introduced by C.A.R. Hoare for the formalization and mathematical treatment of concurrent systems. It is supported by a mathematical theory, a set of proof tools, and an extensive literature. CSP permits the description of systems in terms of component processes that operate independently and interact with each other through message-passing communication. There are two types of primitives: events and processes. Processes are independent self-contained entities with particular interfaces through which they interact with environment. Events (or actions) are central elements of interaction and communications among processes or between a process and an environment.

MAX is a programming language considered as a graphical and musical environment for developing real-time musical applications by connecting boxes, which represent a particular treatment of sound. It was developed by Miller Puckette, and proposed as “the Patcher” in (Puckette, 1988). The fundamental element in MAX is the patch, which is a set of objects (boxes) interconnected by lines. These objects send messages among them and respond by taking actions. Since there were no documents explaining the theoretical foundations of MAX, this programming language was formalized in (Seleborg, 2004) using CSP. Then, a Patch is defined as a network of reactive objects (or reactive system): A patch receives some values from an external device (software or hardware), treats these values, and then, returns other data calculated from the original values to an external device (the same or another). This makes MAX an environment builder for musical reactive systems.

2.3 PiCO

The AVISPA research group in Colombia was founded to develop models for integrating Object Oriented and Concurrent Constraint Programming into a Visual Language to have a programming environment sustained in rich semantics to facilitate computer music application development. The group defined PiCO (Alvarez et al., 1998; Rueda et al., 2001), a calculus integrating objects and constraints. The π+-calculus (Diaz et al., 1999) extends π-calculus (Milner et al., 1992) with a constraint notion. PiCO adds π+ the notion of objects and messages synchronized by constraints. Constraints and concurrent objects are primitive notions at a calculus level (the objects are located in constraints and sending messages is defined by delegation).

This calculus is a foundation for developing a computational model which is suitable for constructing music composition tools. ***Cordial*** (Quesada et al., 1998) is a high-level visual programming language integrating object-oriented and constraint programming intended for musical applications. Its semantics is based on PiCO. Cordial is an iconic language in the spirit of OpenMusic. The basic elements of a program are those of object-oriented programming, such as classes, objects and methods. The solution of a music composition problem is given by a visual, concurrent, object-oriented and constraint programming.

2.4 The MWSCCS calculus

The Calculus of Communicating Systems (CCS) (Milner, 1980) is a process algebra proposed by Robin Milner when he noticed that concurrent processes have an algebraic structure; that is, if there were two processes, P and Q, already built; a new process can be built by combining them sequentially or concurrently, and the new process behavior depends on that of P and Q, and the operation used to combine them. A process algebra based on CCS, called MWSCCS, was introduced in Ross (1995) as an extension of WSCCS (Tofts, 1990), a probabilistic version of the synchronous CCS. In MWSCCS, the basic atomic event is called a particle. Each particle represents either an output communication (denoted by an over barred letter) or an input communication (denoted by a non-over barred letter). Particles are used to construct actions. Actions are events occurring at one moment in time. This calculus permits assigning relative frequencies and priorities to processes.

The Musical Weighted Synchronous Calculus of Communicating Systems (MWSCCS) was developed to design stochastic automata to model complex stochastic musical systems. Particles are notes in a chord, inputs are “hearing” actions and outputs are “playing” actions.

2.5 Block-Diagram algebra

Yann Orlarey proposed the Block-Diagram Algebra in (Orlarey et al., 2002) as an algebraic approach to construct block diagram. It was design as an alternative to classical graph approach inspired by dataflow models. This algebra gives an explicit formal semantics to dataflow inspired music languages by means of high-level construction operations combining and connecting block diagrams and rules associated to each construction operation.

Faust (Gaudrain and Orlarey, 2003) is a programming language designed for real-time sound processing and synthesizing. It combines two programming models: functional programming (the name Faust, Functional AUdio Stream, comes from this approach) and block-diagram composition. This language was developed by Grame helping programmers and musicians build audio stream processors. In fact, Faust is an implementation of the block-diagram algebra; that is, it can be thought as a structured block diagram language with a textual syntax. A Faust block-diagram denotes a signal processor transforming signals. This language provides primitives similar to C/C++ operators such as arithmetic, comparison, bitwise, constants, casting, tables, and interface elements.

2.6 The *tcc calculi

Vijay A. Saraswat, in Saraswat (1993), has proposed concurrent constraint programming (CCP) as a model for specifying concurrent systems in terms of constraints. An extension of this calculus is the TCC calculus (Saraswat et al., 1994), aimed at programming and modeling timed, reactive systems. Frank Valencia and Catusia Palamidessi proposed the temporal concurrent constraint programming calculus, ntcc, in (Palamidessi and Valencia, 2001; Valencia, 2002). NTCC extends TCC with the notions of asynchrony and no determinism. Another extension of TCC, proposed in (Olarte et al., 2007; Olarte, 2009) by Carlos Olarte, Frank Valencia and Catusia Palamidessi is called UTCC. This calculus increases TCC expressivity allowing infinite behavior and mobility by introducing the abstraction notion. One of the main purposes of this model is to verify security protocols. Jorge Perez and Camilo Rueda proposed a timed concurrent constraint process calculus with probabilistic and non-deterministic choices as a description language in Perez and Rueda (2008). This calculus, called PNTCC, is a TCC calculus for analyzing reactive systems involving constraints, explicit time, probabilities and non-determinism. Finally, RTCC, an extension of NTCC to model real-time behavior, was proposed by Gerardo M. Sarria M. and Camilo Rueda in Sarria & Rueda (2008). This formalism extends NTCC in three directions: it introduces resources as a native notion of the calculus, it enriches the time notion by thinking time as a discrete sequence of minimal units, and it adds constructs to interrupt and delay processes. The semantics of the *tcc calculi described above and its application in music is well-explained in Olarte et al. (2011).

Camilo Rueda and Frank Valencia have proposed NTCC as a model for expressing temporal music processes and applications like rhythm patterns and controlled improvisation (Rueda and Valencia, 2001). In Rueda and Valencia (2002) some musical properties were formally proved using the linear temporal logic of NTCC. NTCC was also proposed to model an audio processing system. In Rueda and Valencia (2005) this calculus was used to describe a framework for audio processing able to model higher-level musical structures and to build formal proofs of properties for a given audio process. On the other hand, musical scores involving static and interactive events, which are bound by some logical properties (like Allen's relations (Allen, 1983)), called interactive scores (Desainte-Catherine and Allombert, 2004), were represented by using NTCC (Allombert et al., 2006) and UTCC (Olarte and Rueda, 2009). A computational model for musical dissonances was proposed in Perchy and Sarria (2009), using the RTCC calculus. Finally, in Assayag and Dubnov (2004) a model based on the Factor Oracle algorithm (Allauzen et al., 1999) was proposed for machine improvisation and related style simulation. Later, in Olarte and Rueda (2009), Perez and Rueda (2008) and Rueda et al. (2006), the factor Oracle was modeled using NTCC, PNTCC and UTCC to be used in learning, improvisation and performance situations.

3. Results and discussion

Some considerations for each mentioned formal model are presented, remarking strengths and weaknesses (those characteristics that are important and have musical significance, and those crucial in musical environments, which are not part or are not native in those formalisms).

The λ-calculus was created several years ago; hence, one of the main advantages of this calculus is its maturity and robustness. It has a countless number of applications in different areas of knowledge. However, it lacks native notions as time and constraint. Without these elements, it is difficult to express different musical temporal aspects, calculations involved, representation of objects, constraints in different applications, and partial information. Applications based on λ and λ& calculi do not have any base supplying insights to define new concepts as primitive temporal entities, either to express repetition and eventuality notions or to provide different models to organize objects in time (OpenMusic and Elody, for instance, provide some of these characteristics, but as an implementation out of the formal model capabilities). Furthermore, there is no formal notion of a constraint system in λ-calculus, which limits the possibilities of applying constraints, and prevents a formal statement of what exactly valid manipulations of temporal objects are, and what the visual representation of musically significant temporal constraints stands for (although OpenMusic has been enhanced with a constraint library called "Situation" (Rueda and Bonnet, 1998), but this is not native in the formalism). Additionally, notions as interaction and concurrency, quite typical in music, are not a priority in λ-calculus. This limits the possibility of expressing parallel composition of processes, synchronization of musical pieces, musicians' dynamics, etc.

An advantage of using CSP in modeling applications in different areas of knowledge is that it is conceptually simple, yet provides an appropriate solution to common synchronization problems. However, as in λ-calculus, time and constraints are not native notions. Time can be modeled by taking into account a discrete variable, which never decreases its value and changes its value in an infinite recursion. Nevertheless, there is no control on change rate, this means, time is logic. The same kind of time model was observed in the NTCC calculus: it is not possible to associate this logic time exactly to physical time; all depends on several other conditions. On the other hand, the behavior of CSP processes depends on its environment. Therefore, it is difficult to assert global properties. Then, there is an absence in terms of the precise specification in system properties, which is natural in a constraint-oriented language.

Avispa research group managed to integrate concurrent objects and constraints using PiCO. This calculus permits representing partially-defined complex objects as musical structures in a compact way, and describing harmony relations easily. Nevertheless, since there is no explicit notion of time in PiCO, some musical problems involving time and synchronization are difficult to express. Moreover, since there is no formal logic associated with the calculus, reasoning about musical processes behavior is hard to accomplish. Given the convenience of graphical representations as Block-Diagrams, in Tavera (2008), an extension of Pico, called GraPico, was proposed as a visual representation of the calculus. An abstract view of musical behavior is possibly the main advantage of the MWSCCS calculus. The intuitive "programming language" feeling and the mathematical foundations of process calculi allow building and analyzing compositions written in MWSCCS. Since this calculus is based on CCS, it has rich semantics, an easy way to handle concurrent activities and the ability to formally model specific domain systems. CCS, CSP, and ACP (Bergstra and Klop, 1984) were the first proposed in calculus process field; nevertheless, all of them have similar disadvantages to those previously mentioned as absence time and constraint notions.

The Block-Diagram Algebra is a visual language, which is more intuitive, takes the user to a higher level of abstraction, and makes program analysis easier. This calculus in an appropriate formalism for visual languages because of the graph representation of a block-diagram, its denotational semantics, which describes a program meaning by denoting what is computed (the mathematical object), and its suitability for formal manipulations: λ calculus, partial evaluation, compilation.

The NTCC, UTCC and PNTCC calculi have proved to be convenient for modeling music problems and proving properties in a musical environment. Their well-defined semantics and logic permit easily expressing and proving temporal properties.

Notwithstanding, they may not be adequate for solving complex musical improvisation problems due to real time requirements in these systems. On the other hand, the RTCC calculus handles this limitation with precise notions of time, resources and useful operators. Unfortunately, RTCC does not have a well-structured associated logic; that is, it is difficult to prove properties in models written in this calculus.

4. Conclusions

Recently, a significant increase of formal models has arisen, particularly computational calculi proposed in many fields. Music is not the exception. With formal theories composers and musicians can use tools based on rigorous principles, rules, equations, theorems, models, and languages to solve specific musical problems, to build musical theories, to synchronize devices in music interaction settings, to build musical programming languages, to prove musical properties, to construct complex musical material, and many others. In this paper, some important calculi used in music, and formalisms used in practical musical situations were presented. Nevertheless, many of these formalisms were not originally intended for being used in music; thus, their applicability in many musical environments is difficult to deal with. Some mentioned models lack explicit notions which are crucial in music such as processes, time, constraints, and concurrency; however they have been widely studied. Temporal concurrent constraint calculi like NTCC and RTCC, designed to model interactive systems, fit better in music applications where processes interact in complex ways. Notwithstanding, their limitations concerning the abstract notion of time or the absence of an associated logic to prove properties were shown.

Many music programming languages and musical software use the expressivity and usefulness of formal models (see Loy and Abbott (1985) to know about the former): logics, for instance, were used as music models in Gibbins (1976) and as a music theory tool in Alan Marsden's MTT (Marsden, 1997). Petri Nets have been applied to model music as a concurrent activity in Haus and Sametti (1991).

Besides, music theory has used formal models. One of the first theoretical writings outlining a mathematically rigorous music theory was proposed by Joseph Schillinger in Schillinger (1941) and Schillinger (1948). Geraint Wiggins in Wiggins (2009) reviewed various computational representations of musical systems. Geometrical spaces have been used to represent chords in Tymoczko et al. (2006). A fractal music development was presented in Wright (1995). Hierarchical structures have been used to represent musical objects in Desainte-Catherine (1996). Trace theory has been applied to music in Chemillier and Timis (1988). A deductive object-oriented approach to formalize jazz piano knowledge was proposed in Hirata (1995). Algebraic structures were introduced in Chemillier (1989) approaching a formalization of musical structures. A formal definition of sound was proposed in Kaper (1999). Finally, a category-oriented framework was presented in Mazzola and Andreatta (2007) to describe the relationship between musical and mathematical activities.

5. References

Agon, C. A. (1998). *Openmusic: A langage visuel pour la composition musicale assistée par ordinateur*. Ph.D. thesis, Univeristé Paris VI, Paris, France.

Agon, C. A. (2004). Mixing visual programs and music notation. *Perspectives in Mathematical and Computational Music Theory*. Electronic Publishing Osnabruck.

Allauzen, C., Crochemore, M., & Raffinot, M. (1999). *Factor oracle: A new structure for pattern matching*. In Proceedings of the Conference on Current Trends in Theory and Practice of Informatics. p. 295–310.

Allen, J. F. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM 26* (11), 832–843.

Allombert, A., Assayag, G., Desainte-Catherine, M., & Rueda, C. (2006). *Concurrent constraint models for specifying interactive scores*. In Proceedings of the 3rd Sound and Music Computing Conference (SMC'06). Marseille, France.

Alvarez, G., Diaz, J. F., Quesada, L. O., Valencia, F. D., Assayag, G., & Rueda, C. (1998). *Pico: A calculus of concurrent constraint objects for musical applications.* In Proceedings of the European Congress on Artificial Intelligence (ECAI'98), Brighton, England.

Assayag, G. (1998). *Computer assisted composition today*. In 1st Symposium on Music and Computers. Corfu, Grecia.

Assayag, G., Agon, C. A., Rueda, C., & Delerue, O. (1998). *Objects, time and constraints in openmusic.* In Proceedings of the International Computer Music Conference (ICMC'98), ICMA, Ed. University of Michigan, Ann Arbor, USA.

Assayag, G. & Dubnov, S. (2004). Using factor oracles for machine improvisation. *Soft Computing* 8 (9), p. 604–610.

Assayag, G., Rueda, C., Laurson, M., Agon, C. A., & Delerue, O. (1999). Computer-assisted composition at ircam: from patchwork to openmusic. *Computer Music Journal 23* (3), 59–72.

Balaban, M., Barzilay, E., & Elhadad, M. (2002). Abstraction as a means for end-user computing in creative applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans 32* (6), 640–653.

Barzilay, E. (1996). *Booms object oriented music system.* M.S. thesis, Ben-Gurion University of the Negev, Israel.

Bergstra, J. A. & Klop, J. W. (1984). Process algebra for synchronous communication. *Information and Control 60* (1–3), 109–137.

Cardelli, L. & Abadi, M. (1996). *A Theory of Objects.* Springer.

Castagna, G. (1998). *Foundation of object-oriented programming.* Tutorial Notes – Laboratoire d'Informatique de l'École Normale Supérieure – France.

Chemillier, M. (1989). *Structure et méthode algébriques en informatique musicale.* Ph.D. thesis, Univeristé Paris VII, Paris, France.

Chemillier, M. & Timis, D. (1988). *Towards a theory of formal musical languages.* In Proceedings of 14th International Computer Music Conference (ICMC'88). GMIMIK, Kologne, Germany, p. 175–183.

Church, A. (1985). *The Calculi of Lambda* Conversion. Princeton University Press.

Dannenberg, R. B. (1984). *Arctic: A functional language for real-time control.* In Proceedings of the ACM Symposium on LISP and Functional Programming (LFP'84). ACM Press, Austin, Texas, United States, p. 96–103.

Dannenberg, R. B. (1989). The canon score language. *Computer Music Journal 13* (1), 47–56.

Dannenberg, R. B., Fraley, C. L., & Velikonja, P. (1991). Fugue: A functional language for sound synthesis. *IEEE Computer 24* (7), 36–42.

Desainte-Catherine, M. (1996). *The hierarchical structure may improve the resolution of musical problems.* In Proceedings of Journees d'Informatique Musicale (JIM'96). Île de Tatihou, Basse Normandie, France.

Desainte-Catherine, M. & Allombert, A. (2004). *Specification of temporal relations between interactive events.* In Proceedings of the Sound and Music Computing (SMC'04). Ircam, Paris, France.

Diaz, J. F., Rueda, C., & Valencia, F. (1999). A calculus for concurrent processes with constraints. *CLEI Electronic Journal 1* (2), p. 20–33.

Gaudrain, E. & Orlarey, Y. (2003). *A Faust Tutorial.* Grame, Centre National de Création Musicale.

Gibbins, P. (1976). Logics as models of music. *The British Journal of Aesthetics 16* (2), 157–160.

Hanappe, P. (1999). *Design and Implementation of an Integrated Environment for Music Composition and Synthesis*. PhD thesis, Paris VI, Paris, France.

Haus, G. & Sametti, A. (1991). Scoresynth: A system for the synthesis of music scores based on petri nets and a music algebra. *IEEE Computer 24* (7), 56–60.

Hirata, K. (1995). *Towards formalizing jazz piano knowledge witha deductive object-oriented approach.* In Proceedings of the Artificial intelligence and Music (IJCAI'95). p. 77–80.

Hoare, C. A. R. (1978). Communicating sequential processes. *Communications of the ACM 21* (8), 666–677.

Hudak, P., Makucevich, T., Gadde, S., & Whong, B. (1996). Haskore music notation - an algebra of music. *Journal of Functional Programming 6* (3), 465–483.

Kaper, H. G. (1999). *Formalizing the concept of sound.* In Proceedings of the International Computer Music Conference (ICMC'99), ICMA, Ed. Beijing, China.

Laurson, M. (1996). *Patchwork: A visual programming language and some musical applications.* Ph.D. thesis, Sibelius Academy, Helsinki, Finland.

Loy, G. & Abbott, C. (1985). Programming languages for computer music synthesis, performance, and composition. *ACM Computing Surveys 17* (2), 235–265.

Marsden, A. (1997). *Mtt - a music theory tool.* In Proceedings of Journees d'Informatique Musicale (JIM'97). Biblioteque de la Part-Dieu, Lyon - France.
Mathews, M. V., Miller, J. E., Moore, F. R., Pierce, J. R., & Risset, J. C. (1969). *The Technology of Computer Music.* The MIT Press.

Mazzola, G. & Andreatta, M. (2007). Diagrams, gestures and formulae in music. *Journal of Journal of Mathematics and Music* 1, 23-46.

Milner, R. (1980). A Calculus of Communicating Systems. *Lecture Notes in Computer Science.* Springer-Verlag.

Milner, R., Parrow, J., & Walker, D. (1992). A calculus of mobile processes, parts i and ii. *Information and Computation* 100 (1), 1–40.

Olarte, C. (2009). *Universal temporal concurrent constraint programming.* Ph.D. thesis, Ecole Polytechnique - France, France.

Olarte, C., Palamidessi, C., & Valencia, F. (2007). Universal timed concurrent constraint programming. Logic Programming. *Lecture Notes in Computer Science 4*670, Springer-Verlag, 464-465.

C. Olarte, C. Rueda, G. Sarria, M. Toro, & F. Valencia. (2011) Concurrent constraints models of music interaction. In: G. Assayag and C. Truchet (editors), *Constraint Programming in Music*. (Chapter 6), p 133-153. Wiley.

Olarte, C. & Rueda, C. (2009). *A declarative language for dynamic multimedia interaction systems.* In Proceedings of the Second International Conference of the Society for Mathematics and Computation in Music (MCM2009). Communications in Computer and Information Science (CCIS), 38. Yale University in New Haven, Connecticut, USA.

Orlarey, Y., Fober, D., & Letz, S. (1997). *L'environnement de composition musicale elody.* In Proceedings of the Journees d'Informatique Musicale (JIM'97). Bibliothèque de la Part-Dieu, Lyon - France.

Orlarey, Y., Fober, D., & Letz, S. (2002). *An algebra for block diagram languages.* In Proceedings of the International Computer Music Conference (ICMC'02), ICMA, Ed. Gothenburg, Sweden, p. 542–547.

Orlarey, Y., Fober, D., Letz, S., & Bilton, M. (1994). *Lambda calculus and music calculi.* In Proceedings of the International Computer Music Conference (ICMC'94), ICMA, Ed. DIEM, Danish Institute of Electroacoustic Music, Denmark, p. 243–250.

Palamidessi, C. & Valencia, F. (2001). *A temporal concurrent constraint programming calculus.* In Proceedings of the Seventh International Conference on Principles and Practice of Constraint Programming. Lecture Notes in Computer Science, 2239. Springer-Verlag, p. 302–316.

Perchy, S. & Sarria, G. (2009). *Dissonances: Brief description and its computational representation in the rtcc calculus.* In Proceedings of the 6th Sound and Music Computing Conference (SMC2009). Casa da Música, Porto, Portugal, p. 53–58.

Perez, J. A. & Rueda, C. (2008). *Non-determinism and probabilities in timed concurrent constraint programming.* In Proceedings of the 24th International Conference on Logic Programming (ICLP 2008). Lecture Notes in Computer Science 5366 (1), p. 677–681.

Puckette, M. (1988). *The patcher.* In Proceedings of the International Computer Music Conference (ICMC'88), ICMA, Ed. GMIMIK, Kologne, Germany.

Quesada, L. O., Rueda, C., & Tamura, G. (1998). *The Visual Model of Cordial.* In Proceedings of the XXIV Conferencia Latinoamericana en Informática (CLEI98), Quito, Ecuador.

Roads, C. (1985). Research in music and artificial intelligence. *ACM Computing Surveys*, 17 (2), 163–190.

Ross, B. J. (1995). *A process algebra for stochastic music composition.* In Proceedings of the International Computer Music Conference (ICMC'95), The Banff Centre for the Arts, Alberta, Canada.

Rueda, C., Alvarez, G., Quesada, L. O., Tamura, G., Valencia, F., Diaz, J. F., & Assayag, G. (2001) Integrating constraints and concurrent objects in musical applications: A calculus and its visual language. *Kluwer Academic Publishers 6* (1), 21–52.

Rueda, C., Assayag, G., & Dubnov, S. (2006). *A concurrent constraints factor oracle model for music improvisation.* In Proceedings of the XXXII Conferencia Latinoamericana en Informática (CLEI2006), Santiago de Chile, Chile.

Rueda, C. & Bonnet, A. (1998). *Un langage visual basee sur les constraintes pour la composition musicale.* Recherches et Applications en Informatique Musicale. Paris: Hermes Science Publications.

Rueda, C. & Valencia, F. (2001). F*ormalizing timed musical processes with a temporal concurrent constraint programming calculus.* Musical Constraints Workshop (CP'2001). Cyprus.

Rueda, C. & Valencia, F. (2002). *Proving musical properties using a temporal concurrent constraint calculus.* In Proceedings of the International Computer Music Conference (ICMC'02), Gothenburg, Sweden.

Rueda, C. & Valencia, F. (2005). *A temporal concurrent constraint calculus as an audio processing framework.* In Proceedings of the 2nd Sound and Music Computing Conference (SMC'05). Salerno, Italy.

Saraswat, V. A. (1993). *Concurrent Constraint Programming.* ACM Doctoral Dissertation Award. The MIT Press, Cambridge, MA, USA.

Saraswat, V. A., Jagadeesan, R., & Gupta, V. (1994). *Foundations of timed concurrent constraint programming.* In Proceedings of the Ninth Annual IEEE Symposium on Logic in Computer Science, Paris, France. p. 71–80.

Sarria, G. (2008). *Formal models of timed musical processes.* Ph.D. thesis, Escuela de ingeniería de sistemas y computación, Universidad del Valle, Cali, Colombia.

Sarria, G. & Rueda, C. (2008). *Real-time concurrent constraint programming.* In Proceedings of the XXXIV Conferencia Latinoamericana en Informática (CLEI2008), Santa Fe, Argentina.

Schillinger, J. (1941). *The Schillinger System of Musical Composition.* Carl Fischer, New York. Reprinted by Da Capo Press in 1978.

Schillinger, J. (1948). *The Mathematical Basis of the Arts.* The Philosophical Library. Reprinted by Da Capo Press in 1976.

Seleborg, C. (2004). *Interaction temps-réel/temps différé: Élaboration d'un modèle formel de max et implémentation d'une bibliothèque osc pour openmusic.* M.S. thesis, Université Aix-Marseille II, Marseille, France.

Steele, G. L. (1990). *Common Lisp The Language.* 2nd Edition. Digital Press.

Taube, H. (1990). Common music: A music composition language in common lisp and clos. *Computer Music Journal 15* (2), p. 21–32.

Tavera, C. (2008). *Diseño, implementación y corrección de grapico: Un cálculo visual, orientado al objeto y por restricciones compilado a PiCo.* Ph.D. thesis, Escuela de ingeniería Eléctrica y Electrónica, Universidad del Valle, Cali, Colombia.

Tofts, C. (1990). *A synchronous calculus of relative frequency.* In Proceedings of Theories of concurrency: unification and extension (CONCUR'90), Lecture Notes in Computer Science 458, Springer-Verlag, p.467-480

Tymoczko, D., Callender, C., & Quinn, I. (2006). The geometry of musical chords. *Science* 313, 72–74.

Valencia, F. (2002). T*emporal concurrent constraint programming.* Ph.D. thesis, University of Aarhus, Dinamarca

Wiggins, G. A. (2009). *Computer-representation of music in the research environment.* In Crawford, T. T. and Gibson, L., editors, Modern Methods for Musicology: Prospects, Proposals and Realities, Digital Research in the Arts and Humanities, p. 7–22. Ashgate, Aldershot, UK.

Wright, P. (1995). G*enerating fractal music.* Ph.D. thesis, University of Western Australia.