Author manuscript, published in "Constraint Programming in Music (2011) 133"

hal-00662257, version 1 - 23 Jan 2012

Chapter 1

Concurrent Constraint Models of Multimedia Interaction

1.1. Introduction

Process calculi (see e.g., [MIL 99, NIE 02, SAR 93]) are well-established formalisms to describe concurrent systems so as to reason rigorously about their properties. Their compositional nature allows a convenient hierarchical system modeling methodology where details can be added incrementally in a controlled way. Moreover, their small number of constructs, expressiveness and precise semantics provide ideal frameworks to build on top of them a variety of languages suitable to various application domains. This is particularly important in music systems where tools must provide relevant intuitions and operations to composers. Because semantic coherence is at the heart of a process calculus, extra features are usually proposed only when the need arises to address very specific new phenomena. For instance, Concurrent Constraint (CCP) calculi where defined to add to concurrent models the ability to compute and synchronize with partial information (i.e. constraints).

In this chapter we follow this "economy of means" way to present several varieties of CCP calculi, starting from a very basic one and building from it by adding new features. A fundamental one for music applications is the ability to represent temporal behavior. This can be introduced within the context of determinate (`tcc`, `utcc`) or non-determinate (`ntcc`) computation. For the determinate case, we show how the addition of a process abstraction feature (`utcc`) allows to model dynamic

---

Chapter written by Carlos OLARTE and Camilo RUEDA and Gerardo SARRIA and Mauricio TORO and Frank D. VALENCIA.

hal-00662257, version 1 - 23 Jan 2012

musical structures in a very simple way. In particular, we model a dynamic version of interactive scores ([ALL 07]). For the nondeterminate case, we use the possibility of defining many alternative computational paths to model an agent following different rhythmic patterns constructed from a given basic one. We then go on to consider a more "metrical" notion of time (`rtcc`) based on uniform ticks used by processes to define their time of execution in a more fine-grained way, or to cause preemption of other processes at more precisely defined points in time. We use these new "real-time" features to describe a simple model of a basic form of musical dissonances.

Two important characteristics of CCP calculi relevant to music modeling are its parametric and declarative natures. The parameterization of CCP in a constraint system provides a very flexible way to tailor data structures to specific domains. In the music improvisation system presented here, for example, a particular constraint system is used to implicitly represent a graph. The declarative nature allows CCP processes to be related to logical formulae, so that desirable properties of a system can be handily verified. This can be very important in musical applications, such as controlled improvisation systems where the appearance of certain musical material is expected, or as support for musical analysis. We show in this chapter how to verify some properties of a `ntcc` model of a particular rhythmic pattern. Finally, we describe the implementation of simulators for `ntcc` and for its stochastic variety.

1.2. Concurrent Constraint Programming

Concurrent constraint programming (CCP) [SAR 93] has emerged as a simple but powerful paradigm for *concurrent systems*; i.e. systems of multiple agents that interact with each other as for example in a collection of music processes (musicians) performing a particular piece. A fundamental issue in CCP is the specification of concurrent systems by means of constraints. A constraint (e.g. $note > 60$) represents partial information about certain system variables. During the computation, the current state of the system is specified by a set of constraints called the *store*. Processes can change the state of the system by *telling* (adding) information to the store and synchronize by *asking* information to the store.

The type of constraints in CCP is not fixed but parametrized by a *constraint system* [SAR 93] which specifies the basic constraints agents can tell or ask during execution. A constraint represents a piece of information (or partial information) upon which processes may act. For instance, in a system with variables $pitch_1$, $pitch_2$ taking MIDI values, the constraint $pitch_1 > pitch_2$ specifies possible values for $pitch_1$ and $pitch_2$ (those where $pitch_1$ is at least a tone higher than $pitch_2$). The constraint system defines also an *entailment* relation ($\vdash$) specifying inter-dependencies between constraints. Intuitively, $c \vdash d$ means that the information d can be deduced from the information represented by the constraint c, e.g., $pitch_1 > 60 \vdash pitch_1 > 42$.

hal-00662257, version 1 - 23 Jan 2012

We can set up the notion of constraint system by using First-Order Logic. Given a signature Σ and a first-order theory Δ over Σ, constraints can be thought of as first-order formulae over Σ. Furthermore, $c \vdash d$ if the implication $c \Rightarrow d$ is valid in Δ. As an example, take the finite domain constraint system (FD) [HEN 98]. In FD, variables are assumed to range over finite domains and, in addition to equality, we may have predicates (e.g., "$\leq$") that restrict the possible values of a variable to some finite set.

The Language of CCP Processes. In the spirit of process calculi, the language of processes in CCP is given by a small number of primitive operators. Following the notation in [NIE 02], we present the syntax of CCP in the following definition.

Definition 1 (CCP Processes) *Processes $P, Q, \ldots$ in CCP are built from constraints in the underlying constraint system by the following syntax:*

$$P, Q := \mathbf{skip} \mid \mathbf{tell}(c) \mid \mathbf{when}\ c\ \mathbf{do}\ P \mid P \parallel Q \mid (\mathbf{local}\ \vec{x}; c)\ P$$

The process $\mathbf{skip}$ does nothing. It represents *inaction*. The process $\mathbf{tell}(c)$ adds the constraint c to the store. The process $\mathbf{when}\ c\ \mathbf{do}\ P$ *asks* if c can be deduced (entailed) from the store. If so, it behaves as P. In other case, it remains blocked until the store contains at least as much information as c. This way, ask processes define a synchronization mechanism based on entailment of constraints. The process $P \parallel Q$ denotes the parallel composition of P and Q, i.e., P and Q running in parallel and possibly "communicating" via the common store.

Local variables are declared via the process $(\mathbf{local}\ \vec{x}; c)\ P$. This process behaves like P, except that all the information on the variables $\vec{x}$ produced by P can only be seen by P. This local information corresponds to the constraint c representing a *local store*. We write $(\mathbf{local}\ \vec{x})\ P$ as a shorthand for $(\mathbf{local}\ \vec{x}; \mathtt{true})\ P$.

1.2.1. *Timed CCP*

In CCP once a constraint is added it cannot be removed, i.e., information grows monotonically. This condition is relaxed by considering temporal extensions of CCP such as Timed CCP (`tcc`) [SAR 94]. In `tcc`, processes evolve along a series of *discrete time intervals* (or time-units). Each interval contains its own store and information is not automatically transferred from one interval to another. In music, the duration of each time-unit is related to the minimal metric value used in the piece.

Definition 2 (Temporal Constructs) *The `tcc` processes result from adding in Definition 1 the following constructs:*

$$P, Q \quad := \quad \mathbf{next}\ P \mid \mathbf{unless}\ c\ \mathbf{next}\ P \mid !\, P$$

hal-00662257, version 1 - 23 Jan 2012

The constructs above allow the CCP processes to have effect along the time-units. The unit-delay $\mathbf{next}\, P$ executes P in the next time interval. The process $\mathbf{unless}\, c\, \mathbf{next}\, P$ (*negative ask*) is also a unit-delay but P is executed in the next time-unit iff c *is not* entailed by the final store at the current time interval. This can be viewed as a (weak) time-out (or weak preemption): It waits one time-unit for a piece of information c to be present and if it is not, it triggers activity in the next time interval.

Finally, the *replication* $!\,P$ means $P \parallel \mathbf{next}\, P \parallel \mathbf{next}^{\,2} P ...$, i.e. unboundedly many copies of P but one at a time.

As we said before, processes in `tcc` evolve along a series of discrete time intervals. In each time interval, a CCP processes receives a stimulus (i.e. a constraint) from the environment as an input. It executes with this stimulus as the *initial store*. When it reaches its resting point, i.e., no further evolution is possible, it responds to the environment with the final store (the output of the system). Furthermore, the resting point determines a residual process, which is then executed in the next time interval. Roughly, the residual process is obtained by "unfolding" the sub-terms within "next" and "unless" expressions (see [SAR 94, NIE 02] for a formal account of the operational semantics of `tcc`).

This view of reactive computation is particularly appropriate for programming reactive systems in the sense of Synchronous Languages [BER 92], i.e., systems that react continuously with the environment at a rate controlled by the environment.

In spite of its simplicity, the `tcc` extension to CCP is far-reaching. Many interesting temporal constructs can be expressed. For example, `tcc` allows processes to be "clocked" by other processes. This provides meaningful pre-emption constructs and the ability of defining *multiple forms of time* instead of having a unique global clock.

In what follows we present different applications of CCP-based calculi in the specification and verification of multimedia interaction systems. Some of these applications require the CCP model to be extended with constructs to express, for instance, probabilistic and nondeterministic behavior. In each case, we briefly explain the extension of the calculus and its application.

1.3. Dynamic Interactive Scores

An interactive score [ALL 07] is a pair composed of *temporal objects* and Allen temporal relations [ALL 83]. In general, each object is comprised of a start-time, a duration, and a procedure. The first two can be partially specified by constraints, with different constraints giving rise to different types of temporal objects, so-called *events* (duration equals zero), *textures* (duration within some range), *intervals* (textures without procedures) or *control-points* (a temporal point occurring somewhere within an

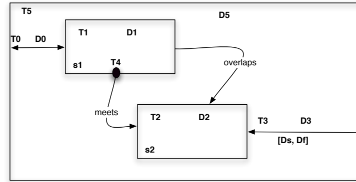


Figure 1.1. *Interactive score*

interval object). The procedure gives operational meaning to the action of the temporal object. It could just be playing a note or a chord, or any other action meaningful for the composer. Figure 1.1, based on one from [ALL 07], shows an interactive score where temporal objects are represented as *boxes*. Objects are T_i, durations D_i. Object T_4 is a control point, whereas T_0 and T_3 are intervals. Duration D_3 should be such that $D_s \leq D_3 \leq D_f$. The whole temporal structure is determined by the hierarchy of temporal objects.

Suppose that, as a result of the information obtained by the occurrence of an event, object T_2 should no longer synchronize with a control-point inside T_1 but, say, with a similar point inside T_5. This very simple interaction cannot be expressed in the standard model of interactive scores [ALL 07]. Another example is an object waiting for some interaction from the performer within some temporal interval. If the interaction does not occur, the composer might then determine to probe the environment again later when a similar musical context has been defined. This amounts to moving the waiting interval from one box to another. For this, it is necessary to provide mechanisms to reconfigure dynamically the communication structure of the processes during execution. This is, it is required to express mobile behavior, e.g, to move/communicate the links of the boxes. In the next section we present a model for dynamic interactive scores originally proposed in [OLA 09] where the expressiveness of *Universal Timed CCP* (`utcc`) to model mobile behavior is central.

1.3.1. *Mobile behavior in* `tcc` *and a Model of Dynamic Interactive Scores*

The `utcc` calculus [OLA 08] aims at modeling mobile reactive systems, i.e., systems that can change their communication structure while interacting with their environment. The basic move from `tcc` to `utcc` is to replace the process $\mathbf{when}\ c\ \mathbf{do}\ P$ by a *temporary parametric ask* constructor of the form $(\mathbf{abs}\ \vec{x}; c)\ P$. This process can be viewed as an *abstraction* of the process P on the variables $\vec{x}$ under the constraint (or with the *guard*) c. Intuitively, $(\mathbf{abs}\ \vec{x}; c)\ P$ performs $P[\vec{t}/\vec{x}]$ in the current time interval for *all the terms* $\vec{t}$ s.t $c[\vec{t}/\vec{x}]$ is entailed by the store. The abstraction construct in `utcc` has a pleasant duality with the local operator. From a programming language

hal-00662257, version 1 - 23 Jan 2012

$$
\begin{array}{lcl}
BoxOperations & \stackrel{\texttt{def}}{=} & (\mathbf{abs}\ id, d; \texttt{mkbox}(id,d)) \\
 & & \qquad (\mathbf{local}\ s)\ \mathbf{tell}(\texttt{box}(id,d,s)) \\
 & & \parallel (\mathbf{abs}\ x,y; \texttt{before}(x,y))\ \mathbf{when}\ \exists_z(\texttt{in}(x,z) \wedge \texttt{in}(y,z))\ \mathbf{do} \\
 & & \qquad \mathbf{unless}\ \texttt{play}\,(y)\ \mathbf{next}\ \mathbf{tell}(\texttt{bf}(x,y)) \\
 & & \parallel (\mathbf{abs}\ x,y; \texttt{into}(x,y))\ \mathbf{unless}\ \texttt{play}\,(x)\ \mathbf{next}\ \mathbf{tell}(\texttt{in}(x,y)) \\
 & & \parallel (\mathbf{abs}\ x,y; \texttt{out}(x,y))\ \mathbf{when}\ \texttt{in}(x,y)\ \mathbf{do} \\
 & & \qquad \mathbf{unless}\ \texttt{play}\,(x)\ \mathbf{next}\ (\mathbf{abs}\ z; \texttt{in}(y,z))\ \mathbf{tell}(\texttt{in}(x,z)) \\
Constraints & \stackrel{\texttt{def}}{=} & (\mathbf{abs}\ x,y; \texttt{in}(x,y))\ (\mathbf{abs}\ d_x, s_x; \texttt{box}(x,d_x,s_x)) \\
 & & \qquad (\mathbf{abs}\ d_y, s_y; \texttt{box}(y,d_y,s_y)) \\
 & & \qquad \mathbf{tell}(s_y \leq s_x) \parallel \mathbf{tell}(d_x + s_x \leq d_y + s_y) \\
 & & \parallel (\mathbf{abs}\ x,y; \texttt{bf}(x,y))\ (\mathbf{abs}\ d_x, s_x; \texttt{box}(x,d_x,s_x)) \\
 & & \qquad (\mathbf{abs}\ d_y, s_y; \texttt{box}(y,d_y,s_y))\ \mathbf{tell}(s_x + d_x \leq s_y) \\
Play(x,t) & \stackrel{\texttt{def}}{=} & \mathbf{when}\ t \geq 1\ \mathbf{do}\ \mathbf{tell}(\texttt{play}(x)) \parallel \mathbf{unless}\ t \leq 1\ \mathbf{next}\ Play(x, t-1)
\end{array}
$$

Figure 1.2. *A `utcc` model for Dynamic Interactive Scores*

perspective, the variables $\vec{x}$ in $(\mathbf{local}\,\vec{x}; c)\,P$ can be seen as the local variables of P while $\vec{x}$ in $(\mathbf{abs}\,\vec{x}; c)\,P$ as the formal parameters of P.

Definition 3 (`utcc` Processes) *The `utcc` processes result from replacing in the syntax in Definition 2 the expression* $\mathbf{when}\ c\ \mathbf{do}\ P$ *with* $(\mathbf{abs}\,\vec{x}; c)\,P$.

The extra expressiveness of `utcc` allows us to model dynamic music systems where the composer can dynamically change the hierarchical structure of the score according to the information derived from the environment. In Figure 1.2 we present an excerpt of the model proposed in [OLA 09] and we explain it in the following.

The process $BoxOperations$ may perform the following actions:

– $\texttt{mkbox}(id, d)$: defines a new box id with duration d. The start time is defined as a new (local) variable s whose value will be constrained by the other processes.

– $\texttt{before}(x, y)$: checks if x and y are contained in the same box. If so, the constraint $\texttt{bf}(x, y)$ is added.

– $\texttt{into}(x, y)$: dictates that the box x is into the box y if x is not currently playing.

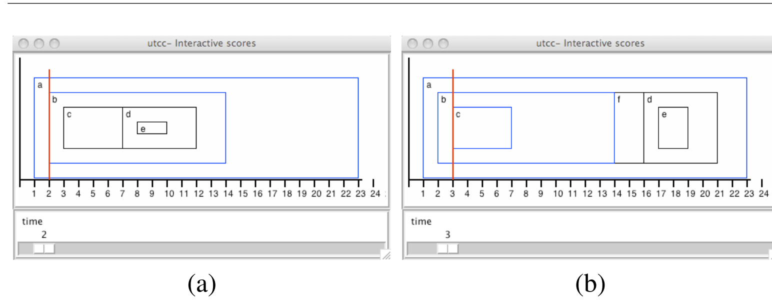
– $\texttt{out}(x, y)$: takes the box x out of the box y if x is not currently playing.

Process $Constraints$ adds the necessary constraints relating the start times of each temporal object to respect the hierarchical structure of the score. For each constraint of the form $\texttt{in}(x, y)$, this process dictates that the start time of x must be less than the one of y. Furthermore, the end time of y (i.e. $d_y + s_y$) must be greater than the end time of x. The case for $\texttt{bf}(x, y)$ can be explained similarly.

The process $Play(x, t)$ adds the constraint $\texttt{play}(x)$ during t time-units. This is an acknowledgment to the environment that the box x is currently playing.

hal-00662257, version 1 - 23 Jan 2012

hal-00662257, version 1 - 23 Jan 2012



(a) (b)

Figure 1.3. *Example of an Interactive Score Execution*

The whole system consists in the parallel composition of the above mentioned processes, some auxiliary processes not depicted in Figure 1.2 for the sake of readability, and a process defining the specific boxes model of the user:

$$
\begin{array}{rcl}
UsrBoxes & \stackrel{\text{def}}{=} & \mathbf{tell}(\texttt{mkbox}(a,22) \wedge \texttt{mkbox}(b,12) \wedge \texttt{mkbox}(c,4)) \parallel \\
& & \mathbf{tell}(\texttt{mkbox}(d,5) \wedge \texttt{mkbox}(e,2)) \parallel \\
& & \mathbf{tell}(\texttt{into}(b,a) \wedge \texttt{into}(c,b) \wedge \texttt{into}(d,b) \wedge \texttt{into}(e,d)) \parallel \\
& & \mathbf{tell}(\texttt{before}(c,d)) \parallel \\
& & \mathbf{whenever}\ \texttt{play}(b)\ \mathbf{do\ unless}\ \texttt{signal}\ \mathbf{next} \\
& & \qquad\qquad \mathbf{tell}(\texttt{out}(d,b) \wedge \texttt{mkbox}(f,2) \wedge \texttt{into}(f,a)) \parallel \\
& & \qquad\qquad \mathbf{tell}(\texttt{before}(b,f) \wedge \texttt{before}(f,d))
\end{array}
$$

This process defines the hierarchy in Figure 1.3(a). When b starts playing, the system asks if `signal` is set (i.e., if it was provided by the environment). If it was not, the box d is taken out from the context b. Furthermore, a new box f is created such that b must be played before f and f before d as in Figure 1.3(b). Notice that when the box d is taken out from b, the internal box e is still into d preserving its structure.

1.4. Nondeterminism and Verification of Musical Properties

One of the salient features of process calculi is that they provide a runnable specification of the modeled systems. This is, the model can be used to both simulate the behavior of the system and to formally verify properties of it. In this section we describe the `ntcc` calculus [NIE 02], an extension of the `tcc` model to express nondeterminism and asynchrony. Furthermore, we show how the declarative interpretation of `ntcc` processes as formulae in Linear Temporal Logic (LTL) [MAN 91] can be exploited to verify musical properties. For this aim, we shall model some patterns used in the repertoires of Central African Republic studied in [CHE 07]. We shall prove that some patterns satisfy the *Rhythmic Imparity Property* that we explain later on. We start with a simple model of the system and then, we refine it to be able to capture more interesting behaviors and properties.

hal-00662257, version 1 - 23 Jan 2012

1.4.1. *The* `ntcc` *calculus*

The `ntcc` calculus [NIE 02] is a CCP formalism for modeling temporal reactive systems. In `ntcc`, processes can be constrained by temporal requirements such as delays, time-outs and pre-emptions. Thus, the calculus integrates two dimensions of computation: a horizontal dimension dealing with partial information (e.g., $note > 60$) and a vertical one in which temporal requirements come into play (e.g., a process must be executed at any time within the next ten time-units).

The above integration is remarkably useful for modeling complex musical processes, in particular for *music improvisation*. For example, for the vertical dimension one can specify that a given process can nondeterministically choose any note satisfying a given constraint. For the horizontal dimension one can specify that the process can nondeterministically choose the time to play the note subject to a given time upper bound. This nondeterministic view is particularly suitable for processes representing a musician's choices when improvising. Similarly, the horizontal dimension may supply partial information on a rhythmic pattern that leaves room for variation while keeping a basic control.

The `ntcc` calculus is obtained by adding guarded-choices for modeling nondeterministic behavior and an unbounded finite-delay operator for asynchronous behavior.

Definition 4 (`ntcc` **Processes**) *The* `ntcc` *processes result from adding to the syntax in Definition 2 the following constructs:*

$$P, Q \quad := \quad \sum_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ P_i \ \mid\ \star P$$

The guarded-choice $\sum_{i \in I} \mathbf{when}\ c_1\ \mathbf{do}\ P_i$ where I is a finite set of indices, represents a process that, in the current time interval, must nondeterministically choose one of the P_j ($j \in I$) whose corresponding guard (constraint) c_j is entailed by the store. The chosen alternative, if any, precludes the others. If no choice is possible then the summation remains blocked until more information is added to the store.

The operator "$\star$" allows to express asynchronous behavior through the time intervals. Intuitively, a process $\star P$ represents $P + \mathbf{next}\ P + \mathbf{next}^2 P + ...$, i.e., an arbitrary long but finite delay for the activation of P.

1.4.2. *Logic Characterization of* `ntcc` *processes*

CCP calculi enjoys a *declarative* nature that distinguishes it from other models of concurrency: CCP processes can be seen, at the same time, as both computing agents

hal-00662257, version 1 - 23 Jan 2012

and logic formulae (see e.g., [SAR 93, NIE 02, BOE 97, OLA 08]), i.e., programs can be read and understood as logical specifications. A natural benefit of this alternative view is to provide a language suitable for both the specification and the implementation of programs. Let us elaborate on this ideas in the context of the `ntcc` calculus (see [NIE 02] for details).

Linear temporal logics (LTL) have been extensively used to specify properties of timed systems [MAN 91]. Formulae in this logic are built from the following syntax:

Definition 5 (Logic Syntax) *The formulae A, B in LTL are defined by the grammar*

$$A, B, \ldots := \; c \mid A \dot{\Rightarrow} A \mid \; \dot{\neg} A \mid \; \dot{\exists}_x A \mid \circ A \mid \Box A \mid \Diamond A$$

Here c denotes an arbitrary constraint which we shall refer to as *atomic proposition*. The intended meaning of the other symbols is the following: $\dot{\Rightarrow}$, $\dot{\neg}$ and $\dot{\exists}$ represent linear-temporal logic implication, negation and existential quantification. *These symbols are not to be confused with the symbols $\Rightarrow, \neg$ and $\exists$ of the underlying constraint system*. The symbols $\circ$, $\Box$, and $\Diamond$ denote the temporal operators *next*, *always* and *sometime*. Intuitively $\circ\ A$, $\Diamond\ A$ and $\Box\ A$ means that the property A must hold next, eventually and always, respectively.

Let P be a process and A a LTL formulae specifying a given temporal property. One may wonder whether the process P satisfies the specification A, written as $P \models A$. The intended meaning of this assertion is that, regardless the input, every output of P satisfies the temporal formula A.

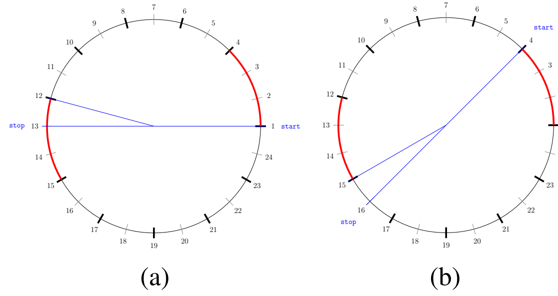
Let us give some examples. Since in every infinite sequence output by $\star\,\mathbf{tell}(c)$ on arbitrary inputs there must be an element entailing c, we have $\star\,\mathbf{tell}(c) \models \Diamond c$. Analogously, $!\,\mathbf{tell}(c) \models \Box c$ since all output of P must entail c. Let $P = \mathbf{tell}(c) + \mathbf{tell}(d)$. We have $P \models (c \,\dot{\vee}\, d)$ as every constraint output by P entails either c or d. Similarly, if $P = \mathbf{tell}(c) \parallel \mathbf{tell}(d)$, then $P \models c \,\dot{\wedge}\, d$ since every output by P entails both c and d. If $P = \mathbf{when}\; c\; \mathbf{do}\; \mathbf{next}\; \mathbf{tell}(d)$ then $P \models c \,\dot{\Rightarrow}\, \circ d$ since the entailment of c in the first time-unit implies the entailment of d in the next one.

1.4.3. *Modeling rhythm patterns*

The following examples were taken from the studies of the repertoires of Central African Republic reported in [CHE 07]. Assume a rhythmic pattern where groups of "2"-unit elements separate groups of "3"-unit elements. For instance, we shall consider sequences such as the one in the following equation:

$$3\; \underbrace{2\,2\,2\,2}\; 3\; \underbrace{2\,2\,2\,2\,2} \tag{1.1}$$

hal-00662257, version 1 - 23 Jan 2012



(a) (b)

Figure 1.4. *Patter of "2" and "3"-unit elements (taken from [CHE 07]).*

where there is a `beat` in the time-units $1, 4, 6, 8, ...$ This pattern can be represented in a circle with 24 divisions, where "2" and "3"-unit elements are placed (see Figure 1.4). In what follows, without loss of generality, we consider only cyclic patterns of 24 time-units.

Given a pattern we start playing the rhythm in an arbitrary position of the sequence. For example, in Figure 1.4 (a), time starts in the first element of the sequence and then, in the first time-unit. In Figure 1.4 (b), time starts in the second element of the sequence and then, in time-unit 4.

The first model we propose for this system considers a fixed sequence of "3" and "2"-units pattern (as in Equation 1.1) and it chooses nondeterministically where the rhythm starts. For this, let $I_1 = \{3, 5, 7, 9, 11, 14, 16, 18, 20, 22\}$ and $I_2 = \{3, 5, 7, 9, 11\}$ and let us define the following processes:

$$\begin{array}{lcl} Beat & \stackrel{\text{def}}{=} & \mathbf{tell}(\mathtt{beat}) \parallel \prod_{i \in I_i} \mathbf{next}^{\,i}\mathbf{tell}(\mathtt{beat}) \\ Start & \stackrel{\text{def}}{=} & \mathbf{tell}(\mathtt{start}) + \sum_{i \in I_2} \mathbf{next}^{\,i}(\mathbf{tell}(\mathtt{start})) \\ Check & \stackrel{\text{def}}{=} & !\,\mathbf{when}\ \mathtt{start}\ \mathbf{do}\ \mathbf{next}^{\,12}(\mathbf{tell}(\mathtt{stop})) \\ System & \stackrel{\text{def}}{=} & Beat \parallel Start \parallel Check \end{array} \tag{1.2}$$

In the above, notation $\prod_{i \in I_i} \mathbf{next}^{\,i} P$ stands for the parallel composition of processes $\mathbf{next}^{\,i} P$ for each $i \in I_1$. The process $Beat$ adds the constraint `beat` in the time-units $1, 4, 6, 8, ...$, this is, in the time-units where a new interval of "2" or "3"-units starts. The process $Start$ signals the time-unit when the rhythm starts (e.g, time-unit 1 and time-unit 4 in Figure 1.4 (a) and Figure 1.4 (b) respectively). The process $Check$ adds the constraint `stop` twelve units after the signal `start` is detected, i.e., it marks the half of the circle. The process $System$ is just the parallel composition of the above mentioned processes.

We use now the logical interpretation of ntcc processes as formulae in LTL to prove the asymmetry property of the sequence in Equation 1.1. This property asserts that *if one attempts to break the circle into two parts, it is not possible to have two equals parts*. To verify this property in our model, let us start with the LTL formulae that corresponds to the processes above, denoted as $[\![P]\!]$:

$$\begin{array}{lcl} [\![Beat]\!] & = & \texttt{beat} \,\dot{\wedge} \dot{\bigwedge\limits_{i \in I_1}} \circ^i \texttt{beat} \\ [\![Start]\!] & = & \texttt{start} \,\dot{\vee} \dot{\bigvee\limits_{i \in I_2}} \circ^i \texttt{start} \\ [\![Check]\!] & = & \Box(\texttt{start} \dot{\Rightarrow} \circ^{12} \texttt{stop}) \\ [\![System]\!] & = & [\![Beat]\!] \,\dot{\wedge} [\![Start]\!] \,\dot{\wedge} [\![Check]\!] \end{array} \tag{1.3}$$

One can then verify that $[\![System]\!] \models \Diamond(\texttt{start} \,\dot{\wedge} \circ^{11}(\texttt{beat} \,\dot{\wedge} \circ\texttt{stop}))$. In words, at some point the signal start is given. Then, eleven units later there is a beat and one unit later the constraint stop can be deduced. This means that it is not possible that the constraints beat and stop are present in the same time-unit (i.e, $[\![System]\!] \not\models \Diamond(\texttt{stop} \,\dot{\wedge} \texttt{beat})$). This then shows that the sequence in the Equation 1.1 satisfies the asymmetric property.

First Refinement. The model in the Equation 1.2 can be refined to prove more interesting properties. For example, one may wonder if there is another distribution of the "3" and "2"-units in the circle such that the asymmetry property holds.

The next model we propose, consists in placing the first "3"-unit pattern in the beginning of the circle, and then, nondeterministically choose where to put the second one. Notice that it suffices to choose the position of the second "3"-unit in the first half of the circle since placing it in the second half leads to a cyclic permutation.

Let $Start$ and $Check$ be as in Equation 1.2, $I_3 = \{2,3,4,5,6\}$ and $Beat'$ and $System'$ as follows:

$$\begin{array}{lcl} Beat' & \stackrel{\text{def}}{=} & \textbf{tell}(\texttt{beat}) \parallel \textbf{next}^{\,3} \sum\limits_{i \in I_3} (\textbf{tell}(\texttt{pos} = i) \parallel Beat_Aux(i-1)) \\ Beat_Aux(N) & \stackrel{\text{def}}{=} & \quad \textbf{tell}(\texttt{beat}) \parallel \\ & & \qquad \textbf{when } N = 1 \textbf{ do next}^{\,3} Beat_Aux(0) \\ & & \quad + \textbf{when } N \neq 1 \textbf{ do next}^{\,2} Beat_Aux(N-1) \\ System' & \stackrel{\text{def}}{=} & Beat' \parallel Start \parallel Check \end{array} \tag{1.4}$$

Intuitively, the process $Beat'$ starts with the first "3"-unit and three units later chooses nondeterministically the position of the next "3"-unit interval in the sequence. Assume that the value chosen for $i \in I_3$ is 3 and then, the sequence starts with 3 2 3 2 2 ..... Notice that if this is the case, the variable pos takes the value 3 and

hal-00662257, version 1 - 23 Jan 2012

the procedure $Beat_Aux$ is called with $N = 2$. Hence the next `beat` will take place two units later, i.e., in the time-unit 6. In the next call, $N = 1$ and then, the "3"-unit pattern is chosen and the next `beat` takes place three units later. From this point, N is less than 1 and therefore, "2"-units intervals are chosen from that point to the end.

From the model in Equation 1.4, one can verify the following. If $x = 6$ then, $[\![System']\!] \models \Diamond((\texttt{pos} = x) \Rightarrow \Diamond(\texttt{stop} \dot{\wedge} \circ\texttt{beat}))$. On the contrary, if $x \in \{2,..,5\}$ then $[\![System']\!] \models \Diamond((\texttt{pos} = x) \Rightarrow \Diamond(\texttt{stop} \dot{\wedge} \texttt{beat}))$. This means, that the unique sequence (up to cyclic permutation) with two "3"-unit pattern satisfying the asymmetry property is the sequence in Equation 1.1.

Second Refinement. A further refinement consists in finding sequences with a given number N of "3"-unit patterns. Notice that N must be an even number, in other case, the number of time-units cannot be 24.

We propose a new process $Beat$ that nondeterministically places a number N of "3"-unit patterns. As in the previous case, we assume that the sequence starts with a "3"-unit interval.

$$
\begin{array}{lcl}
Beat''(i,j,N) & \stackrel{\texttt{def}}{=} & \textbf{when } i \leq 24 \textbf{ do} \\
& & \quad \textbf{when } j = 1 \textbf{ do tell}(\texttt{beat}) \parallel \\
& & \qquad \textbf{when } 24 - 3 \times N \leq i \textbf{ do } Choose_3(i,N) \\
& & \qquad + \textbf{when } 24 - 3 \times N > i \textbf{ do} \\
& & \qquad\quad \textbf{when } N > 0 \textbf{ do } Choose_3(i,N) \\
& & \qquad\quad + \textbf{when } \texttt{true} \textbf{ do } Choose_2(i,N) \\
& & \quad + \textbf{when } j \neq 1 \textbf{ do next } Beat''(i+1, j-1, N) \\
Choose_3(i,N) & \stackrel{\texttt{def}}{=} & \texttt{pos}_i = i \parallel \textbf{next } Beat''(i+1, 3, N-1) \\
Choose_2(i,N) & \stackrel{\texttt{def}}{=} & \textbf{next } Beat''(i+1, 2, N)
\end{array}
$$

In the process $Beat''$, i represents the number of the current time-unit, j the remaining duration of the current interval and N the number of "3"-units that must be placed in the sequence. This process checks if there are still time-units to verify (i.e., $i \leq 24$). If $j = 1$ the constraint `beat` is added and there is a decision to take: to place a "2" or a "3"-unit pattern. If there are no more time-units to place a number N of the "3"-unit missing, then unavoidable a "3"-unit is chosen. This is verified by the guard $24 - 3 \times N \leq i$. If there is enough space to place a "2"-unit pattern, then the process nondeterministically chooses the next unit to be placed. Notice that in each iteration the parameter i is incremented and j decremented if it is not necessary to take a decision in the current time-unit.

Similarly to the previous models, we can verify properties such as with six "3"-unit intervals, it is possible to find patterns that satisfies the asynchrony property like the sequences "3 3 3 2 3 3 3 2 2" and "3 3 3 2 3 3 2 3 2".

hal-00662257, version 1 - 23 Jan 2012

hal-00662257, version 1 - 23 Jan 2012

1.5. Real Time and Preemption

In many musical systems, constraints, time and concurrency arise. As we showed in Section 1.4, the `ntcc` calculus may to some extent be convenient for this. However time in `ntcc` is logical, that is, each time-unit is defined by the time taken by all processes to make all their internal transitions until no further transition can be done. This is not enough to satisfy quantitative temporal constraints which is a requirement of real-time systems and in music improvisation scenarios.

On the other hand, an essential issue in reactive and real-time systems is process preemption. In [BER 93], this concept is defined as the control mechanism consisting in denying a process the right to be executed, either permanently (abortion) or temporarily (suspension). In music improvisation situations, for instance, there are cases in which the musician must skip some note or play something different to synchronize with other partners, or wait for a signal before continuing.

The `rtcc` calculus [SAR 10] is obtained from `ntcc` by adding constructs for specifying strong preemption and delay declarations, and by extending the transition system with support for resources, limited time and true concurrency.

Definition 6 (`rtcc` Processes) *The `rtcc` processes result from adding in the syntax in Definition 4 the following constructs*

$$P, Q, \ldots \quad := \quad \textbf{\textit{catch}}\ c\ \textbf{\textit{in}}\ P\ \textbf{\textit{finally}}\ Q \ \mid \textbf{\textit{delay}}\ P\ \textbf{\textit{for}}\ \delta$$

The strong time-out process, **catch** c **in** P **finally** Q, represents the interruption of P in the current time interval when the store can entail c; otherwise, the execution of P continues. When process P is interrupted, process Q is executed. If P finishes, Q is discarded. The execution of a process P can be delayed within a given unit. In construct **delay** P **for** δ process P is activated in the current time-unit only at least δ ticks after the beginning of the time-unit (the concept of time-unit is extended as a discrete sequence of minimal units that we call *ticks*).

We showed the expressiveness of `rtcc` in [PER 09] by modeling musical dissonances. Since the dissonance phenomena in music can be seen as an ordered sequence of processes, we can express it using concurrent agents that synchronize each other through signals (constraints) that are global to the whole system. Each agent may represent each phase in the dissonance process and also delay its execution until the previous (dependent) phase has been carried out and signals the system to continue the sequence onto the next phase. The sequence consists of the following phases:

Preparation: Prepares the listener to the confusion of tension that the dissonance may generate in the melody. Generally, this preparation carries a harmonic line corresponding to its tonality.

hal-00662257, version 1 - 23 Jan 2012

Dissonance: In this stage, the dissonance or dissonances are produced, often in weak rhythmic beats or in strong ones depending on its relevance and sonority.

Resolution: Here, the dissonance needs to move to a state of resolution or relaxation. It is here that the dissonance is carried to a more pleasing form, often taken to the main tonality on long beats.

The model we propose is the following:

$$
\begin{array}{lcl}
Conductor_{[n,m]} & \stackrel{\text{def}}{=} & Musician \parallel Cycle_{[n,m]} \parallel \mathbf{tell}(\mathtt{go}) \parallel \mathbf{next}\,(\star(!\,\mathbf{tell}(\mathtt{stop}))) \\
 & & \parallel !(\mathbf{unless}\ \mathtt{stop}\ \mathbf{next} \\
 & & \qquad \mathbf{catch}\ \mathtt{stop}\ \mathbf{in}\ \mathbf{when}\ \mathtt{end}\ \mathbf{do} \\
 & & \qquad\quad (Musician \parallel Cycle_{[n,m]} \parallel \mathbf{tell}(\mathtt{go})) \\
Cycle_{[n,m]} & \stackrel{\text{def}}{=} & \star(\mathbf{tell}(\mathtt{prep}) \parallel \star_{[1,n]}(\mathbf{tell}(\mathtt{diss}) \parallel \star_{[1,m]}(\mathbf{tell}(\mathtt{res}))
\end{array}
$$

The main entry point of the model is the agent $Conductor$ (parameters n and m bound the time to change from one stage of the dissonance cycle to the next). This agent will activate the $Musician$ and a process $Cycle$ to motivate a dissonance. It also gives a signal to the musician for starting the melody ($\mathbf{tell}(\mathtt{go})$) and eventually it will give another signal to end music generation ($\star(\mathbf{tell}(\mathtt{stop}))$). Additionally, if the `stop` signal has not already been given and the musician ends a dissonance, $Conductor$ will activate the $Musician$ and the process $Cycle$ again. This could be seen as a loop for the musician to continue playing the melody and eventually to perform a dissonance until the `stop` signal is detected.

The $Cycle$ process posts the signals for each stage of the dissonance.

The agent $Musician$ is defined as follows:

$$
\begin{array}{lcl}
Musician & \stackrel{\text{def}}{=} & \textbf{when}\ \mathtt{go}\ \textbf{do catch}\ \mathtt{prep}\ \textbf{in}\ Melody\ \textbf{finally}\ Stage1 \\
Stage1 & \stackrel{\text{def}}{=} & \textbf{catch}\ \mathtt{diss}\ \textbf{in}\ Preparation\ \textbf{finally}\ Stage2 \\
Stage2 & \stackrel{\text{def}}{=} & \textbf{catch}\ \mathtt{res}\ \textbf{in}\ Dissonance\ \textbf{finally}\ Stage3 \\
Stage3 & \stackrel{\text{def}}{=} & Resolution \parallel \mathbf{tell}(\mathtt{end})
\end{array}
$$

The $Musician$ will start executing the process $Melody$ (supposed to play the main melody of the whole song) waiting to catch the signal `prep` during it. When it catches the signal, it interrupts (`stop`) the $Melody$ and launches the $Stage1$ of the dissonance.

The same philosophy applies to the agents $Stage1$ and $Stage2$ each of them waiting for the signal telling to carry on the next stage in the dissonance sequence, also assuming that process $Preparation$ plays the preparation and process $Dissonance$ executes the dissonance.

hal-00662257, version 1 - 23 Jan 2012

To conclude the sequence, the agent $Stage3$ launches the process $Resolution$ (also assumed to play a resolution congruent with the dissonance) and posts a signal `end` telling the conductor that the current dissonance is over.

The process $Melody$ is the main harmonic structure the musician has planned for the song. It is in charge of evolving the melody, so to speak. Processes $Preparation$, $Dissonance$ and $Resolution$ will select nondeterministically a chord to play from a set of chords specifically built to fulfill each one of the process tasks. For example, the set of chords from the process $Preparation$ is able to transcend to a dissonance but when the process $Dissonance$ takes the lead, the set of chords from where it will choose to play will be dissonant ones.

This proposed concurrent model may be adapted easily to fit the management of the dissonance according to the needs of the musician. The reader may see that any of the steps to construct the sequence can be easily left out or changed without affecting the integrity of the whole system.

1.6. Probabilistic extensions and Music Improvisation

Musical improvisation is another natural context for interacting and concurrent agents. Improvisation is effective when agents behavior adapts to what has been learned in previous interactions. A music style-learning/improvisation scheme such as Factor Oracle (FO) [ALL 99, ASS 04] can be seen as a reactive system where several learning and improvising agents react to information provided by the environment or by other agents. In the following we recall the structure of the FO and a model of it using the `utcc` calculus. Then, we show how suitable probabilistic extensions of CCP allows for richer traversals of the structure to improve the quality of the improvisation.

1.6.1. *The Factor Oracle (FO)*

A FO is a finite state automaton constructed in an incremental fashion. A sequence of symbols $s = \sigma_1\sigma_2 \ldots \sigma_n$ is learned in such an automaton, which states are $0, 1, 2 \ldots n$. There is always a transition arrow (called factor link) labeled by the symbol σ_i going from state $i - 1$ to state $i, 1 \leq i < n$. Depending on the structure of s, other arrows will be added. Some are directed from a state i to a state j, where $0 \leq i < j \leq n$. These also belong to the set of factor links and are labeled by symbol σ_j. Some are directed “backwards”, going from a state i to a state j, where $0 \leq j < i \leq n$. They are called suffix links, and bear no label (represented as '$\star$' in Figure 1.5). The factor links model a factor automaton, that is every factor p in s corresponds to a unique factor link path labeled by p, starting in 0 and ending in some other state. Suffix links have an important property: a suffix link goes from i to j iff the longest repeated suffix of $s[1..i]$ is recognized in j. Thus suffix links connect repeated patterns of s.

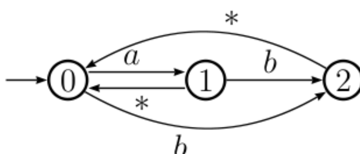


Figure 1.5. *A FO automaton for $s = ab$*

The oracle is learned on-line. For each new input symbol σ_i, a new state i is added and an arrow from $i-1$ to i is created with label σ_i. Starting from $i-1$, the suffix links are iteratively followed backward, until a state is reached where a factor link with label σ_i originates (going to some state j), or until there is no more suffix links to follow. For each state met during this iteration, a new factor link labeled by σ_i is added from this state to i. Finally, a suffix link is added from i to the state j or to state 0 depending on which condition terminated the iteration.

Navigating the oracle in order to generate variants is straightforward : starting in any place, following factor links generates a sequence of labelling symbols that are repetitions of portions of the learned sequence; traversing one suffix link followed by a factor link creates a recombined pattern sharing a common suffix with an existing pattern in the original sequence. This common suffix is, in effect, the musical context at any given time.

In [OLA 09] a `utcc` model of the FO is proposed. The model assumes a simple constraint system with the predicate symbols $\texttt{edge}(x, y, N)$ representing a link (suffix or factor) between node x and y labeled with N. When a new symbol is provided, the processes defined add the necessary constraints to obtain a representation of the graph in the store. The advantage of using CCP-based calculi for this task is that the inclusion of a new agent in the FO model (e.g. a learner agent for a second performer) entails only a new process and new interactions, both with the new process and among the existing ones. In traditional languages this usually means major changes in the synchronization scheme, which are difficult to localize and control. In `utcc`, all synchronization is done semantically, through the available information in the store.

1.6.2. *Probabilistic Transversal of the FO*

Once we have a model to construct the FO, the next step is to navigate it. It has been shown, e.g., in [ASS 04], that assigning a probability to choose between playing a learned factor or a new sequence leads to a better improvisation scheme.

The authors in [PÉR 08] study the integration of probabilistic information into CCP. The language `pntcc` features both nondeterministic and probabilistic behavior. The operational semantics of `pntcc` ensures the consistent interactions between

hal-00662257, version 1 - 23 Jan 2012

hal-00662257, version 1 - 23 Jan 2012

both the probabilistic and the nondeterministic choices. The semantics is based on a probabilistic automaton [SEG 95] that separates the internal choices made probabilistically by the processes from those external choices made nondeterministically under the influence of a scheduler. As a result, the observable behavior of a system – what the environment perceives from its execution - formalized by the semantics is purely probabilistic; the influence of nondeterminism is regarded as unobservable.

Definition 7 (`pntcc` Processes) *The `pntcc` processes result from adding to the syntax in Definition 2 the process*

$$\bigotimes_{i \in I} \mathbf{when}\ c_i\ \mathbf{do}\ (P_i, a_i)$$

where I is a finite set of indices, and for every $a_i \in \mathbb{R}^{(0,1]}$ we have $\sum_{i\ inI} a_i = 1$

The intuition of this operator is as follows. Each a_i associated with the process P_i represents its probability of being selected for execution. Hence, the collection of all a_i represents a probability distribution (see e.g., [GUP 97] for other probabilistic extension of CCP).

Using `pntcc` we can easily define a process to navigate probabilistically the FO. When the signal `play` is detected, the process below chooses to follow a suffix link with a probability ρ and to follow a factor link with a probability $\rho - 1$ (processes *Suffic* and *Factor* are not important here and omitted for the sake of presentation).

$$Improv \stackrel{\text{def}}{=} \mathbf{when}\ \texttt{play}\ \mathbf{do}\ (Suffix, \rho) \otimes \mathbf{when}\ \texttt{play}\ \mathbf{do}\ (Factor, 1 - \rho)$$

Implementation of `pntcc`. The previous model has been implemented in `Ntccrt` [TOR 09], a framework to execute `ntcc` and `pntcc` models. This system computes each time-unit using the *Gecode* library (`http://www.gecode.org/`). It is worth noticing that it is not necessary to "solve" a constraint satisfaction problem but to use only *constraint propagation* (not enumeration) to compute the output of each time-unit.

`Ntccrt` is written in C++ and specifications can be made in Common Lisp or in *OpenMusic* [AGO 98], and then translated to C++. This framework can be also integrated as a plugin for either *Pure Data (*PD*) or Max/MSP* [PUC 98] to take advantage of the facilities offered by those languages to implement, for example, sound processors. We believe that using the graphical paradigm provided by Max or PD is difficult and time-demanding to synchronize processes depending on complex conditions. On the contrary, using `Ntccrt` we can model such systems where complex conditions can be naturally represented by constraints.

hal-00662257, version 1 - 23 Jan 2012

1.7. Perspectives and Future Work

CCP has recently attracted a renewed attention as witnessed by the works [PAL 06, BUS 07, BEN 09, BAR 10] on formalisms for concurrency exhibiting data-types, logic assertions as well as tell and ask operations. These CCP-based formalisms are at an early stage of development and they should be equipped with proof and verification techniques that take advantage of the constraint nature of CCP. In particular, we envisage two strategic research directions for this purpose: The adaptation of the *bisimilarity*-based proof techniques and *automaton*-based verification for CCP.

Bisimilarity is probably the main representative of equivalences in concurrency. It captures our intuitive notion of process equivalence; two processes are equivalent if they can match each other's moves. A promising research direction is to explore sound and complete notions of bisimilarity for CCP that benefit of the feasible proof techniques typically associated with this equivalence in other frameworks.

The issue of automatic or machine-assisted verification has hitherto been far too little considered for concurrent constraint-based formalisms. Several interesting applications of CCP are inherently complex and large, therefore, machine-assisted verification is essential. To the best of our knowledge only Villanueva et al (see e.g, [FAL 06]) have addressed automatic verification but only in the context of a particular timed CCP calculus. Like for other process calculi, automatic verification of CCP systems presents us with serious computational challenge. A potential research direction to take up the this challenge is to identify CCP fragments amenable to automatic verification. We believe that one can use a symbolic approach by taking advantage of the unique constraint nature of constraint-based calculi. Recall that constraints can be used as symbolic compact representation of large, possibly infinite, set of states. We believe that this idea can be adapted to produce a novel symbolic state space representations based on temporal constraints: Namely constraints representing large, possibly infinite, set of evolutions of systems. To the best of our knowledge temporal constraints have not been used for symbolic techniques in verification.

All in all, we believe that concurrent constraint-based techniques can offer important benefits, such as runnable specifications and system properties assurance, to system modeling in real applications, in particular in Music Interaction. Future research should therefore focus on devising techniques that will allow an integrated modeling, execution and verification environment for CCP similar to those available for other formalisms in concurrency theory (e.g., [VIC 94]). Real-time simulators of the kind we described above, coupled with the verifier and extended with means to interface with existing standard music applications, such as sound synthesis/analysis tools, would give musicians a very powerful and novel modeling environment.

hal-00662257, version 1 - 23 Jan 2012

Chapter 2

Bibliography

[AGO 98] AGON C., ASSAYAG G., DELERUE O., RUEDA C., “Objects, Time, and Constraints in OpenMusic”, *ICMC 98*, 1998.

[ALL 83] ALLEN J. F., “Maintaining knowledge about temporal intervals”, *Commun. ACM*, vol. 26, num. 11, ACM, 1983.

[ALL 99] ALLAUZEN C. CROCHEMORE M. R. M., “Factor oracle: a new structure for pattern matching”’, *Proc. of SOFSEM’99*, LNCS, 1999.

[ALL 07] ALLOMBERT A., ASSAYAG G., DESAINTE-CATHERINE M., “A system of interactive scores based on Petri nets”, *proceedings of SMC ’ 07*, 2007.

[ASS 04] ASSAYAG G., DUBNOV S., “Using Factor Oracles for Machine Improvisation”, *Soft Comput.*, vol. 8, num. 9, p. 604-610, 2004.

[BAR 10] BARTOLETTI M., ZUNINO R., “A Calculus of Contracting Processes”, *LICS*, IEEE Computer Society, p. 332-341, 2010.

[BEN 09] BENGTSON J., JOHANSSON M., PARROW J., VICTOR B., “Psi-calculi: Mobile processes, nominal data, and logic”, *Proc. of LICS*, IEEE CS, 2009.

[BER 92] BERRY G., GONTHIER G., “The ESTEREL Synchronous Programming Language: Design, Semantics, Implementation”, *Science of Computer Programming*, vol. 19, num. 2, p. 87–152, 1992.

[BER 93] BERRY G., “Preemption in Concurrent Systems”, *Proceedings of the 13th Conference on Foundations of Software Technology and Theoretical Computer Science*, London, UK, Springer-Verlag, p. 72–93, 1993.

[BOE 97] DE BOER F. S., GABBRIELLI M., MARCHIORI E., PALAMIDESSI C., “Proving Concurrent constraint Programs Correct”, *ACM Transactions on Programming Languages and Systems*, vol. 19, num. 5, 1997.

[BUS 07] BUSCEMI M. G., MONTANARI U., “CC-Pi: A Constraint-Based Language for Specifying Service Level Agreements”, *In Proc. of ESOP*, 2007.

hal-00662257, version 1 - 23 Jan 2012

[CHE 07] CHEMILLIER M., *Les Mathématiques Naturelles*, Odile Jacob, 2007.

[FAL 06] FALASCHI M., VILLANUEVA A., "Automatic Verification of tccp programs", *Theory and Practice of Logic Programming*, vol. 6, p. 265-300, 2006.

[GUP 97] GUPTA V., JAGADEESAN R., SARASWAT V. A., "Probabilistic Concurrent Constraint Programming", *Proc. of CONCUR 97*, London, UK, Springer-Verlag, 1997.

[HEN 98] HENTENRYCK P. V., SARASWAT V. A., DEVILLE Y., "Design, Implementation, and Evaluation of the Constraint Language cc(FD)", *J. Log. Program.*, vol. 37, num. 1-3, 1998.

[MAN 91] MANNA Z., PNUELI A., *The Temporal Logic of Reactive and Concurrent Systems: Specification*, Springer-Verlag, 1991.

[MIL 99] MILNER R., *Communicating and Mobile Systems: the Pi-Calculus*, Cambridge University Press, 1999.

[NIE 02] NIELSEN M., PALAMIDESSI C., VALENCIA F., "Temporal Concurrent Constraint Programming: Denotation, Logic and Applications", *Nordic Journal of Computing*, vol. 9, num. 1, 2002.

[OLA 08] OLARTE C., VALENCIA F. D., "Universal Concurrent Constraint Programing: Symbolic Semantics and Applications to Security", *Proc. of SAC 2008*, ACM, 2008.

[OLA 09] OLARTE C., RUEDA C., "A declarative language for dynamic multimedia interaction systems", *Proc. of MCM*, Springer-Verlag, 2009.

[PAL 06] PALAMIDESSI C., SARASWAT V., VALENCIA F., VICTOR B., "On the Expressiveness of Linearity vs Persistence in the Asychronous Pi-Calculus", *Proc. of LICS'06*, IEEE CS, 2006.

[PÉR 08] PÉREZ J. A., RUEDA C., "Non-determinism and Probabilities in Timed Concurrent Constraint Programming", *Proc. of ICLP 2008*, vol. 5366 of *LNCS*, Springer, 2008, Extended version available at `http://www.japerez.phipages.com`.

[PER 09] PERCHY S., SARRIA G., "Dissonances: Brief Description and its Computational Representation in the RTCC Calculus", *Proc. of SMC2009*, Porto, Portugal, July 2009.

[PUC 98] PUCKETTE M., APEL T., ZICARELLI D., "Real-time audio analysis tools for Pd and MSP", *Proc. of ICMC 1998*, 1998.

[SAR 93] SARASWAT V. A., *Concurrent Constraint Programming*, MIT Press, 1993.

[SAR 94] SARASWAT V., JAGADEESAN R., GUPTA V., "Foundations of Timed Concurrent Constraint Programming", *Proc. of LICS'94*, IEEE CS, 1994.

[SAR 10] SARRIA G., "Improving the Real-Time Concurrent Constraint Calculus with a Delay Declaration", *IProc. of ICCSA 2010*, San Francisco, California, USA, October 2010.

[SEG 95] SEGALA R., "Modeling and Verification of Randomized Distributed Real-Time Systems. PhD thesis, MIT", 1995.

[TOR 09] TORO M., AGÓN C., ASSAYAG G., RUEDA C., "Ntccrt: A concurrent constraint framework for real-time interaction", *Proc. of ICMC 2009*, 2009.

[VIC 94] VICTOR B., MOLLER F., "The Mobility Workbench - A Tool for the pi-Calculus", *CAV*, Springer, p. 428-440, 1994.