Concurrent Constraints Models for Interactive Scores

Antoine Allombert, Gérard Assayag, M. Desainte-Catherine, Camilo Rueda

CONCURRENT CONSTRAINTS MODELS FOR INTERACTIVE SCORES

A. Allombert, G. Assayag
Ircam

M. Desainte-Catherine
Bordeaux I University

C. Rueda
Ircam and Javeriana Univ.

ABSTRACT

We propose a formalism for construction and performance of musical pieces composed of temporal structures involving discrete interactive events. The occurrence in time of these structures and events is partially defined according to constraints, such as Allen temporal relations. We represent the temporal structures using two constraint models. A constraints propagation model is used for the score composition stage whereas a non deterministic temporal concurrent constraint calculus (NTCC) is used for the performance phase. The models are tested with examples of temporal structures computed with the GECODE constraint system library and run with a NTCC interpreter.

1. INTRODUCTION

Composing an interactive musical piece often necessitates to construct several musical parts before binding them to interactive events or computing programs. But, on the one hand, existing systems for writing music actually propose very limited real-time interaction, and on the other hand, programming languages, such as MAX (or pd) do not provide the composer with very sophisticated tools for composition.

We claim that a new kind of systems is needed for composing interactive musical pieces. Such systems would provide a composition environment for building musical parts as well as programming tools for specifying interaction computation.

In this paper, we propose a formalism for writing musical pieces involving discrete interactive events. As in [2], we shall call *interactive score*, a musical score involving static and interactive events, that are bound by some logical properties. In this paper, we limit our study to temporal relations, such as the Allen's ones. After the presentation of what is exactly an interactive score, we propose an operational model based on concurrent constraints models, and wich provides a sound specification of any interactive score. Our model comprises a compositional phase and a performance phase. For the first one we propose an incremental constraints propagation model based on the GECODE constraints library, and for the second one a non deterministic temporal concurrent constraints calculus. Our preliminary tests show this model to be appropriate both for score editing and for our real-time requirements, but more experiments are needed for this to be conclusive.

2. BACKGROUND

2.1. Temporal structures

An interactive score is a set of musical objects (such as notes and events) that are bound with temporal relations. In this set, some events are choosed to be interactive. That means that they will happen in real-time. In the general case, temporal relations specify in a partial way the musical piece, so that several pieces can be obtained during performance, according to the real-time events coming in input. This indeterminism provides a kind of degree of freedom to the musicians while the resulting piece still satisfies the composer requirements.

The whole process requires the two following consecutive steps.

1. The compositional process: the composer builds his interactive score by creating his musical objects, binding them with temporal relations, and choosing his interactive events;

2. The performance process: the interactive score is no more edited. The system executes and decides at each step what object must start, continue or stop. These decisions are made according to the asynchronous real-time events coming from input, and the indeterminism that remains in the score.

2.2. Concurrent constraints models

Concurrent constraint programming (CCP [8]) is intended as a model of concurrent systems. In CCP a concurrent system is modeled in terms of constraints over the variables of the system. A constraint is a formula representing *partial information* about the values of some of the variables. For example, in a system with variables $pitch_1, pitch_2$ taking MIDI values, the constraint $pitch_1 > pitch_2 + 2$ specifies possible values for $pitch_1$ and $pitch_2$ (those where $pitch_1$ is at least a tone higher than $pitch_2$). The CCP model includes a set of (*basic*) constraints and a so-called *entailment relation* $\models$ between constraints. This relation gives a way of *deducing* a constraint from the information supplied by other constraints. For example, $pitch_1 > pitch_2 + 2,\ pitch_2 > 60 \models pitch_1 > 48$.

Computation in the CCP model proceeds by accumulating information (i.e. constraints) in a *store*. The information specifies all that is known about the values of the variables at a given moment. Information on the $store$ may increase but it cannot decrease. Concurrent processes

interact with the store either *telling* new information or *asking* whether some constraint can be deduced (entailed) from the information contained in it. It may well happens that the constraint cannot be entailed. In this case the interacting process is said to *block* until some other processes tell enough information to the store to deduce its constraint.

Basic constraints in a CCP model are chosen so that entailment can be efficiently computed. In programming systems based on the CCP model the user can compute with more complex (non-basic) constraints. These systems provide a *propagator* for each different type of user constraint. The role of the propagator is to translate a given asserted constraint into a collection of basic constraints supplying the same information. For example, assuming basic constraints are all of the form $x \in [a..b]$, and a store containing $\{pitch_1 \in [36..72],\ pitch_2 \in [60..80]\}$, a propagator for the constraint $pitch_1 > pitch_2 + 2$ would tell constraints $pitch_1 \in [63..72]$ and $pitch_2 \in [60..69]$. A good CCP language provides the user with efficient propagators for a rich collection of constraint types. A well-known example is the *distinct* type of constraint, used for asserting that the values of some collection of variables must be all pairwise distinct.

As can be seen in the above example, the action of propagators ends up narrowing down the set of possible values for each variable (so-called its *domain*). This, however, does not guarantee that it will eventually be inferred a single value to each variable. CCP languages thus include in general *search engines*. The purpose of a search engine is to choose additional basic constraints to tell into the store until all variables have reduced their domain into a single value. For example, a search engine might choose to tell constraint $pitch_1 \in [72..72]$ in the example discussed above. This will allow the propagator for $pitch_1 > pitch_2 + 2$ to infer constraint $pitch_2 \in [69..69]$ and thus all variables will get assigned to a single value. It may so happens, of course, that the constraint chosen by the search engine leads to a contradiction. The search engine backs up then and performs a different choice of constraint.

A system providing many efficient propagators and powerful user controllable search engines is GECODE ([9]). We describe below how to use it to model a temporal structure interaction system.

One drawback of the CCP model as presented above is that information is always accumulated. There is no way to eliminate it. This poses difficulties for modeling reactive systems in which information on a given variable changes depending on the interactions of a system with its environment, as is the case, for example, in interactive performance systems. Different extensions on the CCP model have been proposed to handle reactive systems. One such model is the non-deterministic temporal concurrent constraint calculus (NTCC, [6]). This calculus introduces the notion of time, seen as a sequence of *time slots*. At each time slot a CCP computation takes place, starting with an empty store (or one that has been given some information by the environment). Concurrent constraints agents operate on this store as in the usual CCP model to accumulate information into the store. As opposed to the CCP model, however, the agents can schedule processes to be run in future temporal slots. In addition, since at the beginning of each time slot a new store is created information on the value of a variable can change (e.g. it can be forgotten) from one slot to the next. The computational agents of NTCC are describe in table 1. Intuitively, agent $\mathbf{tell}(c)$ adds informatrion c to the store

Agent	meaning
$\mathbf{tell}(c)$	Add c to the current store
$\mathbf{when}\ c\ \mathbf{do}\ A$	if c holds now, run A
$\mathbf{local}\ x\ \mathbf{in}\ P$	run P with local x
$A \parallel B$	Parallel composition
$\mathbf{next}\ A$	run A at the next instant
$\mathbf{unless}\ c\ \mathbf{next}\ A$	unless c can be inferred now, run A
$\sum_{i \in I}\ \mathbf{when}\ c_i\ \mathbf{do}\ P_i$	choose P_i s.t. c_i holds
$*\,P$	delay P undefinitely (not forever)
$!\,P$	Execute P each time unit (from now)

Table 1. NTCC agents

of the current time unit. This information can then be used to deduce other constraints. Agent $\mathbf{when}\ c\ \mathbf{do}\ A$ asks whether c can be deduced to hold from the current store and if so, executes agent A. Computed information that is to remain local to an agent is defined by $\mathbf{local}\ x\ \mathbf{in}\ P$. Here, information on x added by P is only seen by itself or by its subprocesses (if any). Reciprocally, any existing global information on x cannot be seen by P. The parallel composition agent $A \parallel B$ runs A and B in parallel. Agent $\mathbf{next}\ A$ schedules A to be run at the next time unit. Notice that an agent $\mathbf{next}\ \mathbf{tell}(c)$ adds information c to the store of the next time unit. Notice that this store might initially be empty or contain some information provided externally by the environment (e.g. as the result of the system interacting with a musical device), but is completely independent of the store of the current time unit. Agent $\mathbf{unless}\ c\ \mathbf{next}\ A$ offers the possibility of performing activity on the basis of *absence* of information. When constraint c cannot be deduced from the store of the current time unit, action A is performed in the next time unit. It should be noticed that in NTCC this means that entailment checking of c is performed when all other processes have finished, i.e. when it is certain that c cannot be deduced in the current time unit.

The choice agent $\sum_{i \in I}\ \mathbf{when}\ c_i\ \mathbf{do}\ P_i$ *non deterministically* runs some process P_i such that its guard c_i can be deduced from the current store. Several of the c_i's could hold but only one P_i is non deterministically chosen. Agent $*\,P$ schedules P to be run either now or at some unspecified time in the future. In practice, a more controlled version of this agent, denoted $*_{[i,j]}\,P$, is used. This schedules P to be run at some time within the (closed)

interval $[i, j]$. This version can be encoded in the standard NTCC. In NTCC, agents are ephemeral. Their life span is just the time unit in which they run. Agent $!\,P$ adds persistence. It launches process P at the current time unit and at all future time units.

The following example illustrates computation in NTCC.

$$SYST \stackrel{\text{def}}{=} !\,\mathbf{tell}(start > 20) \parallel CHECK \parallel PLAY \parallel *_{[50,200]}\,\mathbf{tell}(play(done)) \parallel BEAT(0)$$

$$PLAY \stackrel{\text{def}}{=} !\,\textstyle\sum_{i\in\{1,2,3\}} \mathbf{when}\, play(on)\, \mathbf{do}\, NOTE_i$$
$$CHECK \stackrel{\text{def}}{=} \mathbf{unless}\, beat < start\, \mathbf{next}\, play(on) \parallel \mathbf{unless}\, play(done)\, \mathbf{next}\, CHECK$$
$$BEAT(i) \stackrel{\text{def}}{=} \mathbf{tell}(beat = i) \parallel \mathbf{next}\, BEAT(i+1)$$

The system asserts (persistently) that the value of $start$ is greater than 20 and runs in parallel three processes $PLAY$, $CHECK$ and $BEAT$. It also launches a process that is to stop performance at some unspecified time unit in the range $50..200$. Process $PLAY$ non deterministically chooses one of three notes when playing is on. Process $CHECK$ asserts that playing is on once it can be deduced that the beat counter is greater than or equal to the starting time. It does so repeatedly until the stop playing signal arrives. The $BEAT$ process is simply a counter (recursive process definition can be encoded in the standard NTCC calculus. See [6]).

The NTCC calculus has an associated linear temporal logic. Desirable properties of an NTCC model can be expressed as a formula in this logic. A proof system allows then to verify wether the NTCC model satisfies or not the property.

The NTCC calculus has been used to model musical improvisation processes ([7]). We use it here to account for the interaction with a composer (or device) during performance as a hierarchical temporal process, constrained in various ways, is run.

3. THE MODEL

In this section, we present our model of *interactive scores* on which is based our study. This model directly stems from the model presented in [5].

3.1. Interactive Score

Intuitively, a score is a representation where a set of temporal objects (TO) are disposed in time. Temporal objects have a start time and a duration (or an end time) . If the score is to be executed in real-time by a computer, an associated process is attached to temporal objects, giving them a musical/sound content. A note, for example, is a particularly simple temporal object to which conventional graphical notations may apply, and for which the associated process could be a simple Midi note-on / note-off triggering mechanism. In the general case, the associated process might be much more complexe and involve, for example, starting a synthesis engine and controlling its parameters in real-time. Or, it could involve the processing of an incoming stream of events or sound. In this case the score will be said to be interactive as its execution depends on asynchronous informations from the outside. Musical processes attached to TOs are beyond the scope of this paper. We see three levels of representations for scores in a computer environment : graphical, structural and temporal. These representations establish a complex network of relationships over the TOs. The graphical level provides a set of surface representations and graphical edition tools that may include conventional music notation (where it may apply) or hierarchical boxing representations such as in OpenMusic Maquettes [4] or Boxes [3]. For a given structural and temporal representation, several graphical representations may interchange, that reveal more or less of the structural / temporal details. Structural representations encompass diverse structural relationships such as hierarchical ones (a son TO may belong to a father TO) or functional ones (the process linked to a TO may provide input informations to another TO/process). Temporal representation expresses all the temporal relationships between TOs, such as before, meets etc. This paper is mostly focused on the the temporal representation, which is enough in order to undestand the propagation and exploration processes that takes place at score composition time as well as performance/execution time. For instance hierarchical relationships, usually represented as boxes inside boxes in graphical scores, although they are necessary for the composer to have a synthetic view of his musical sketch at the graphical/structural level, can be for our purpose easily translated at the temporal level into automatically generated basic relations : a son TO will always be linked by a during relation to its father. Although we will for the sake of clarity represent hierarchical information in the graphical representations, only information at the temporal level will be actually processed by the constraint engines described.

So, at the temporal level, we will describe the structure of an interactive score as such :

A score is defined by a tuplet $s = \langle t, r\rangle$ where t is a set of temporal objects and r is a set of temporal relations. A temporal relation is defined by $r = \langle a, t_1, t_2\rangle$ where a belongs to A, the set of Allen relations [1], and t_1 and t_2 are temporal objects.

A temporal object is defined by $t = \langle s, d, p, c\rangle$ where s is the start time d is the duration, p is an attached process, c is a constraint attached to t (i.e. its local store).

The local store will be useful later for assigning musical attributes and configuring classes of temporal objects. It can also serve to assert "value fixing" relations (e.g. $s = 20$, or $d > 50$).

When creating new temporal objects, there is the facility to choose it among four classes that differ in the role they play in the score and the constraints in their store. The four classes are : event, texture, interval, and controlpoint.

- An event has the constraint $d = 0$. Events model

discrete interactive actions. Their attached process is specialized in "listening" to the environment and waiting a triggering signal to happen.

- A texture has the constraints $d \in [d_1, d_2], 0 < d_1 \leq d_2$, which gives its duration an authorized range of variation. If we force d_1 and d_2 to be equal to the textures initial duration, then it is considered rigid. Otherwise it is considered supple. A texture has a generative process.

- An interval is exactly like a texture except it has no generative process. Intervals are used as blank *placeholders* in the score. They help to refine Allens relations with respect to authorized time intervals.

- A control-point p is always created in relation with a texture/interval q. A relation $p\ during\ q$ is automatically added to the score. Control points help to express a time relation between any TO and a particular point inside a texture or an interval.

The class information is kept at the structural representation level, just as the hierarchical information: as for the temporal level, objects are handled in a unified fashion.

Temporal relations

The composer can bind the temporal objects with temporal relations based on the Allens relations. He can define the relations before, meets, overlaps, starts, finishes, during between temporal objects ; as said before, to maintain the temporal hierarchy of the score, a during relation is automaticaly added between a TO and its sons. Allens relations are only qualitative, while all inital temporal positions and durations are quantitatively specified in the score. Thus, we keep this information and use it for expressing quantitative temporal properties that may in certain case put restrictions on the Allen relations. For example, a TO defined as rigid will be obliged to keep the duration it is given when created. The temporal relations are used to keep the organization of the score whenever the composer changes the characteristics of a TO (duration, start time) at score edition time. The new values are propagated through the score and the TOs are moved or stretched as necessary in order to respect the constraints.

Interactive events

We call an interactive event a particular event that is not to be played by the score player. Rather, it models a discrete, asynchronous event that is supposed to happen at performance time in the external environment and to enter the system through an input channel. Such an event could be related to the triggering of a pedal, or the detection of an instrumentist who begins to play, the recognition of a certain pitch played by a musician etc. The composer can define temporal relations between events and any other TO including events. The meets relation will generally be used to synchronize TOs with the arrival of an interactive event and therefore to explicitly represent the way an external control will be able drive the execution of the score at performance time. The process associated to an event will run from the origin of time in the score until the event happens actually. When it does happen, a special constraint will be added to the store, informing the execution machine that it is time to check all the constraints relating this event to other TOs. This will in turn condition the execution of the TOs (start a TO, stop a TO, etc.) that depend on the event. It must be well understood that interactive events may well happen at a certain distance from the date they are assigned to in the score, because of expressive choices or even mistakes. Thus the event date in the score is only the ideal date, and the Allen relations will be used to maintain the score coherence whatever the anticipation or the delay is. Of course this must stay within reasonable limits : an exagerated anticipation or delay should be interpreted as a mistake or a time out. Such limits can be expressed by setting a before relation between an interactive event and other TOs, in order to forbid the event to happen outside of a certain region of the score. One can also use the intervals we have introduced sooner. By defining an interval supple or rigid, by giving it a duration range, one can control the authorized region for an event (see example further). In case of anticipation error or time out, decisions have to be made, the simple of which is to just ignore the event. This can lead to difficulties : due to the web of dependencies between TOs, it could result in preventing the whole remaining score to be executed. Adressing this problem is beyond the scope of the paper. So, the general philosophy behind this all, at performance time, is "keep as much as possible the coherence of the time structure planned in the score, while taking into account, and accepting up to a certain limit, the expressive freedom of the external agents."

An interactive score is shown in figure 1.

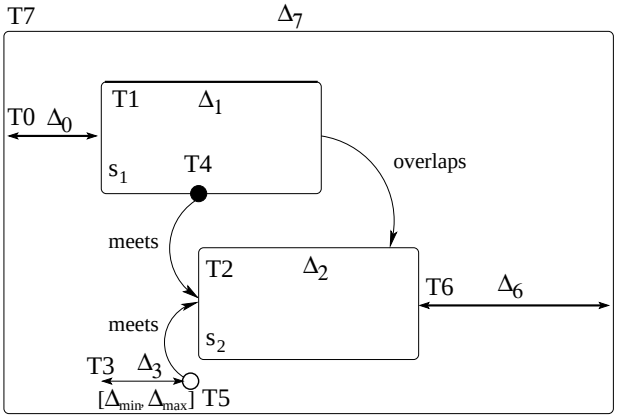

Figure 1. An example of an interactive score

In this example, we have 8 temporal objects $T0$ to $T7$. Objects $T0$ to $T6$ are embedded into $T7$, which means they all have an implicit *during* relation to $T7$. By convention we will call s_i and Δ_i the variables defining the

start time and duration of temporal object Ti.

$T0, T3, T6$ are intervals (drawn as arrows)
$T1, T2$ are textures (drawn as rectangles)
$T5$ is an interactive event (drawn as circle)
$T4$ is a control-point associated to $T1$
(drawn as black circle)
$T0, T1$ and $T6$ are rigid (shown by a bold line)
$T3$ is supple and has a duration range of $[\Delta_{min}, \Delta_{max}]$
$T2$ is supple.

The Allen relations are :

$$T0\ starts\ T7$$
$$T0\ meets\ T1$$
$$T4\ meets\ T2$$
$$T1\ overlaps\ T2$$
$$T5\ meets\ T2$$
$$T3\ starts\ T1$$
$$T3\ meets\ T5$$
$$T2\ meets\ T6$$
$$T6\ meets\ T7$$

The relations involving an interval (e.g. $T0\ meets\ T1$) have not been drawn as the arrow symbol is quite explicit. The interpretation of this score is as follow :

$T7$ is a complex texture that controls the occurrence of a certain number of substructures. From the beginning of execution of $T7$, wait for a duration equal to Δ_0. Then begin playing $T1$. From that point, after duration Δ_{min} has elapsed, we begin to expect an external event ($T5$) that should happen before duration Δ_{max} has elapsed. As soon as $T5$ has been detected, start playing $T2$. When duration $\Delta_0 + \Delta_1$ has elapsed since the beginning of $T7$, stop $T1$. Now the end of $T2$ will depend on the status of $T7$. If $T7$ is rigid, it has a certain duration defined by the composer and the end of $T2$ will occur after duration $\Delta_7 - \Delta_6$ has elapsed since the beginning. If $T7$ is not constrained, then $T2$ will last an undetermined time after $T1$ has finished. Object $T7$ will end Δ_6 units of time after $T2$ has finished.

3.2. The propagation model

In this section we present how we solve the constraints problem we face during the composition when the composer changes the values of the dates of a TO, and we have to propagate it throught the score to maintain consistency in the relations. A score can be translated into a constraint problem where the variables are the starting dates and durations of the TOs, and the constraints are equations deduced from the temporal relations. For example for two TO n_1 and n_2 linked by a relation $n_1\ meets\ n_2$ we have the constraint $s_1 + \Delta_1 = s_2$ with s_1 the starting date of n_1, Δ_1 the duration of n_1, and s_2 the starting date of n_2 . This leads to a linear constraints problem with a cyclic constraint graph. Since a lot of constraint-propagation algorithms do not admit cyclic constraints graphs, we use GECODE [9], a a very efficient multi-engines constraints-satisfaction library written by Christian Schulte. Conceptually, GECODE divides the constraints graph into several parts with structural particularities before treating each part with a specific domain filtering algorithm. GECODE also propagates intervals of values instead of single values, which makes it admit cyclic constraints graphs.

For the example in figure 1, the constraints set is ($d0, d1$ and $d6$ are locked values fixed by the composer):

$$\Delta_0 = d0, \Delta_1 = d1, \Delta_6 = d6$$
$$s_1 = s_7 + \Delta_0$$
$$\Delta_{min} \leq \Delta_3 \leq \Delta_{max}$$
$$s_1 + \Delta_3 = s_5$$
$$s_2 = s_5$$
$$s_2 + \Delta_2 > s_1 + \Delta_1$$
$$s_2 + \Delta_2 + \Delta_6 = \Delta_7$$

We also add constraints with minor priority imposing that each variable is equal to its current value. These level of "soft" constraints is provided in GECODE by means of constraint "reification". In this scheme, instead of posting some property c, a constraint $b \leftrightarrow c$ is posted. This asserts that b is the boolean value of the result of posting c. If $b = false$ is deduced, then c is inconsistent. The branch and bound search engine of GECODE is used to find a solution maximizing the number of b's with the value $true$. In our case, this scheme gives a way of getting, after the pertubation, the closest solution to the solution before pertubation. Remember that we always have a solution before pertubation since the composer designs the score and therefore gives a value to each variable when he creates and places the TOs (we suppose here that he cannot create inconsistencies).

3.3. The NTCC model

Score and temporal objects are represented by ntcc processes. A *score* is a ntcc process that launches in parallel all its TO's and asserts a conjunction r of temporal relations over the TO variables. We use $\prod_{i \in I} P_i$, where I is finite, to denote the parallel composition of all P_i. We also write $(\bigwedge r)$ for the conjunction of all constraints in the set r. A score $\langle t, r \rangle$ is the process

$$Score \stackrel{\text{def}}{=} (\ \prod_{i \in t}\ TO_{i,[P_i,c_i]}\ )\ \|\ !\ tell(\bigwedge r)$$

Each element of r is a temporal relation. Allen relations are naturally expressed as constraints. Three of them are shown below:

$$Before_{(ob1,ob2)} \stackrel{\text{def}}{=} (dat_{ob1} + dur_{ob1} < dat_{ob2})$$
$$Starts_{(ob1,ob2)} \stackrel{\text{def}}{=} (dat_{ob1} = dat_{ob2}) \wedge (dur_{ob1} < dur_{ob2})$$
$$Overlaps_{(ob1,ob2)} \stackrel{\text{def}}{=} (dat_{ob1} < dat_{ob2}) \wedge (dat_{ob1} + dur_{ob1} < dat_{ob2} + dur_{ob2}) \wedge (dat_{ob2} < dat_{ob1} + dur_{ob1})$$

The score process above defines *permanent* relations, but they could as well have been defined to hold only for particular time intervals.

Each temporal object $\langle s_i, d_i, P_i, c_i \rangle$ is a process launching itself at the right time:

$$
\begin{array}{l}
TO_{i,[P_i,c_i]} \stackrel{\text{def}}{=} \\
\quad !\, tell(c_i) \\
\quad \parallel !\, \textbf{unless}\ clock + 1 < s_i \\
\qquad\qquad \textbf{next}\ (\textbf{tell}\ (clock \geq s_i)\ \parallel\ P_i) \\
\quad \parallel !\, \textbf{when}\ clock \geq s_i\ \textbf{do} \\
\qquad \textbf{next}\ (\ Same_i \\
\qquad\qquad \parallel\ \textbf{unless}\ clock \geq s_i + \Delta_i\ \textbf{next}\ P_i)
\end{array}
$$

Notice that if there is not enough information to conclude that the TO should not start it sets its starting time s_i to the (next) current value of the clock. This would cause the TO to launch its activity at the next instant (this is represented by process P_i). This also include cases where s_i has not been constrained to some specific value and information on it is not enough to infer that its value should be greater than the current value of the clock. It can be seen that TO_i schedules itself to finish process P_i at the right time, unless there is no information on its duration, in which case it just continues acting forever. Objects linked to the occurrence of a particular event would have a somewhat different behavior in that they have to wait for the event to arrive before displaying any activity:

$$
\begin{array}{l}
EV_{i,[c_i]} \stackrel{\text{def}}{=} \\
\quad !\, \textbf{when}\ event_i(on)\ \textbf{do} \\
\quad (\quad !\, tell(c_i) \\
\qquad \parallel !\, \textbf{unless}\ clock + 1 < s_i \\
\qquad\qquad \textbf{next tell}\ (clock \geq s_i) \\
\qquad \parallel !\, \textbf{when}\ clock \geq s_i\ \textbf{do next}\ Same_i\ )
\end{array}
$$

A very powerful feature of the calculus is illustrated in the above example: the ability to compute on the basis of *absence* of information. Notice that not being able to deduce, say, $clock < s_i$ is not the same as being able to infer $clock \geq s_i$. In fact, there could be insufficient information to deduce the former and also the latter. In interactive music environments it is frequent that the time of occurrence (if any) or the type of interaction is not known in advance, and it might be useful in this case that the computation continues on the assumption that such an interaction will not take place once an appropriate amount of time has elapsed.

Process $Same_i$ implements transmitting the current value of s_i to the next time slot (n stands for the duration of the whole piece).

$$
\begin{array}{l}
Same_i \stackrel{\text{def}}{=} \\
\quad \sum_{v \in [0..n]}\ \textbf{when}\ s_i = v\ \textbf{do next tell}\ (s_i = v)
\end{array}
$$

The above process finds out first what the current value of s_i is and then just tells that the same value will hold for the next time unit.

The whole system is defined as follows:

$$System \stackrel{\text{def}}{=}\ Score\ \parallel\ CLOCK(0)$$

The clock simply beats time.

$$CLOCK(v) \stackrel{\text{def}}{=}\ \textbf{tell}\ (clock = v)\ \parallel\ \textbf{next}\ CLOCK(v{+}1)$$

Interaction results in adding (or changing) information on the starting time of certain TO's. Interactions are modeled as processes:

$$Trigger_i \stackrel{\text{def}}{=}\ *_{[0..n]}\ \textbf{tell}(event_i(on))$$

The above represents the result of some device eventually triggering some signal at some unspecified moment within the time span of the piece (from 0 to n). A somewhat more elaborate model would involve a composer performing several interactions, each one fixing some s to some particular value. This could also be easily modeled in NTCC:

$$
\begin{array}{l}
Interaction_i \stackrel{\text{def}}{=} \\
\quad !\, (Same_i\ +\ \sum_{k \in [0..n]}\ \textbf{when}\ k > clock\ \textbf{do}\ Try_i(k)\ ) \\
Try_i(k) \stackrel{\text{def}}{=} \\
\qquad \textbf{unless}\ s_i \leq clock \vee k \geq s_i + \Delta_i \\
\qquad\qquad \textbf{next}\ \textbf{tell}\ (s_i = k) \\
\quad \parallel\ \ \textbf{when}\ s_i \leq clock \vee k \geq s_i + \Delta_i\ \textbf{do}\ Same_i
\end{array}
$$

In the above definition two kind of choices are performed. First, a choice is made of wether doing nothing (i.e. keeping the same current starting time values) or to try changing one s value. In the latter case we use the summation construct of NTCC to non-deterministically choose some time value k. The Try_i process then tries to assign k to s_i provided TO_i has not started playing yet.

We proceed now to model the example of figure 1. Textures are represented by the above TO_i process. Intervals and control points are TO_i processes such that $P_i = \textbf{skip}$, the null process.

Let

$$
\begin{array}{l}
t = \{0, 1, ..., 7\} \\
r = \{Starts_{0,7}, Meets_{0,1}, ..., Meets_{6,7}, During_{4,1}, ...\}
\end{array}
$$

The score is

$$
\begin{array}{l}
\quad TO_{0,[\textbf{skip},\Delta_0 = d0]}\ \parallel TO_{1,[P_1,\Delta_1 = d1]}\ \parallel\ TO_{2,[P_2,true]} \\
\parallel\ TO_{3,[\textbf{skip},\Delta_{min} \leq \Delta_3 \leq \Delta_{max}]}\ \parallel\ TO_{4,[\textbf{skip},true]} \\
\parallel\ EV_{5,[\textbf{skip},true]}\ \parallel\ TO_{6,[\textbf{skip},\Delta_6 = d6]}\ \parallel\ TO_{7,[\textbf{skip},true]} \\
\\
\parallel !\, tell(\bigwedge r)\ \parallel\ Trigger_5
\end{array}
$$

4. CONCLUSIONS AND FUTURE WORK

We described in this paper how interactive scores could be conveniently represented in a concurrent constraints model. We used a constraints propagation scheme for the interactive editing composition phase and a temporal concurrent constraints calculus for the interactive performance phase. Preliminary results of implementations of some test cases is encouraging.

In the near future we plan to pursue the work presented here in several directions. Both the editing and performance phases are to be integrated as a music modeling tool within the Open Music environment. This will require devising an efficient two way interface between the GECODE library Common Lisp. Even though NTCC seems to be a good choice for the performance phase, we plan to assess the behavior of NTCC in real-time contexts where complex interactions may occur. The examples presented in this paper were run in an experimental NTCC interpreter implemented in the Mozart programming language[10]. We plan to build from an existing Linux version running in C [11] to develop an efficient implementation for the Mac OS X platform.

5. REFERENCES

[1] Allen, J.F. "Maintaining Knowledge about Temporal Intervals" *Communications of the ACM* 1983

[2] M. Dessainte-Catherine and A. Allombert "Specification of temporal relations betwenn interactive events", *Proc. of the SMC 2004 (Sound and Music Computing), Paris, France* 2004

[3] A. Beurivé "Un logiciel de composition musicale combinant un modle spectral, des structures hirarchiques et des contraintes" *Journes d'Informatique Musicale, JIM 2000*, 2000

[4] Grard Assayag, Camilo Rueda, Mikael Laurson, Carlos Agon, and Olivier Delerue "Computer Assisted Composition at IRCAM: From PatchWork to OpenMusic". *COMPUTER MUSIC JOURNAL Volume 23 No. 3*, 1999

[5] M. Dessainte-Catherine and A. Allombert. "Interactive Scores : A Model for Specifying Temporal Relations between Interactive and Static Events" *JNMR Vol. 34(4)*, 2005.

[6] , C. Palamidessi and F. Valencia. "A Temporal Concurrent Constraint Programming Calculus" *Proc. of the Seventh International Conference on Principles and Practice of Constraint Programming CP2001*, 2001.

[7] C. Rueda and F. Valencia. "Proving musical properties Using a temporal Concurrent Constraints calculus" *Procedings of the ICMC2002*, Goteborg, Sweden, 2002.

[8] V. Saraswat. *Concurrent Constraint Programming* The MIT Press, Cambridge, MA, 1993.

[9] C. Schulte and G. Tack. "Views and Iterators for Generic Constraint Implementations", *Proceedings of the Fifth International Colloqium on Implementation of Constraint and Logic Programming Systems, CICLOPS05.* 2005. Software homepage: http://www.gecode.org

[10] G. Smolka "The Oz Programming Model". *Computer Science Today. Lecture Notes in Computer Science, vol. 1000*,1995

[11] R. Hurtado and P. Munoz and C. Rueda and F. valencia. "Programming Robotic Devices with a Timed Concurrent Constraint Language", *Proc. of the Tenth International Conference on Principles and Practice of Constraint Programming CP2004*, Toronto 2004.