

Chapter 1

Interactive Scores: Past, Present and Future

1.1. Concurrent Constraints Time Model

Process calculi such as the *Non-deterministic Timed Concurrent Constraint* (*ntcc*) [NIE 02] calculus has been used in a variety of musical applications (see Chapter ??). An advantage of *ntcc* is that process synchronization is achieved by adding or deducing constraints from a constraint store, thus is declarative.

In this section we present a model of interactive scores based upon *ntcc*. The model is inspired on a previous model based upon *ntcc* that do not consider hierarchy nor the fact that execution can continue even if interactive events are not launched [ALL 06]. The model is also close in spirit to the *petri nets model*. Regrettably, in *petri nets* is difficult to model global constraints such as the maximum number of simultaneous temporal objects and temporal reductions during execution.

Another related model is *Tempo*, a formalism to define declaratively partial orders of musical and audio processes [RAM 06]. Unfortunately, *Tempo* does not allow us to express choice when multiple conditions hold, simultaneity nor to perform an action if a condition cannot be deduced.

1.1.1. *Ntcc model*

We use *ntcc* to express operational semantics of interactive scores. In order to define operational semantics, we need to transform an interactive score into a graph

Chapter written by Myriam DESAINTE-CATHERINE and Antoine ALLOMBERT and Mauricio TORO.

where the vertices are the start and end points of temporal objects and arcs represent the delays among them. We define the graph as $g = (V, A, lV, lA)$ where V is the set of vertices, A the set of arcs, lV a function that assigns labels to vertices, lA a function that assigns labels to arcs. A vertex is labeled with the type of point (a start point, end point or interactive point) and the temporal object associated to it. An arc is labeled with its duration. The graph is reduced by removing zero-delay arcs and representing several points in the same vertex.

Absence of zero delays simplifies the definition of operational semantics because we do not have to synchronize two processes to occur at the same time. To remove a zero-delay arc between a vertex a and b , we delete a and we connect all its successors and predecessors to b . We also combine the label of a with the label of b , this means that a vertex may represent the start or end of several points.

1.1.1.1. Points

We have two type of points: interactive ($iPoint$) and static points ($sPoint$). Process $Launch_i$ updates the variable $launched_i$ with `true` and persistently assigns to p_i the current value of $clock$. Event e_i is the user event associated to point i : it represent a user interaction. Set Pr represents the predecessors of i . Process $User$ persistently chooses between launching or not an interactive event.

$$iPoint_{i,Pr} \stackrel{def}{=} \text{when } \bigwedge_{j \in Pr} launched_j \text{ do (} \\ \quad \text{when } clock + 1 > p_i \text{ do next } Launch_i \\ \quad \quad \quad \parallel \text{unless } clock + 1 < p_i \text{ next when } e_i \text{ do } Launch_i) \\ \quad \parallel \text{unless } launched_i \text{ next } iPoint_{i,Pr}$$

$$sPoint_{i,Pr} \stackrel{def}{=} \text{when } \bigwedge_{j \in Pr} launched_j \text{ do (} \\ \quad \text{unless } clock + 1 < p_i \text{ next } Launch_i \\ \quad \quad \quad \parallel \text{when } clock + 1 < p_i \text{ do next } sPoint_{i,Pr}) \\ \quad \parallel \text{unless } launched_i \text{ next } sPoint_{i,Pr}$$

$$Points_{lV,A} \stackrel{def}{=} \prod iPoint_{i \in \{e | lV(e) = (interactivePoint, t)\}, Pr = \{a | (a, b) \in A \wedge b = i\}} \\ \parallel \prod sPoint_{i \in \{e | lV(e) \neq (interactivePoint, t)\}, Pr = \{a | (a, b) \in A \wedge b = i\}}$$

$$User_l \stackrel{def}{=} \prod_{i \in \{e | lV(e) = (InteractivePoint, t)\}} !(tell(e_i) + skip)$$

1.1.1.2. Intervals

Intervals are represented by constraints. An interval between points i and j with a duration of Δ is represented by the constraint $p_i + \Delta = p_j$. The value of ∞ is approximated by the a parameter of the model, namely n_∞ .

$$Interval_{i,j,\Delta} \stackrel{def}{=} !tell(p_i + \Delta = p_j) \quad Intervals_{lA} \stackrel{def}{=} \prod_{(i,j,\Delta) \in lA} Interval_{i,j,\Delta}$$

1.1.1.3. Example

Figure 1.2 represents the graph associated to the score in Figure 1.1. It does not contain zero-delays.

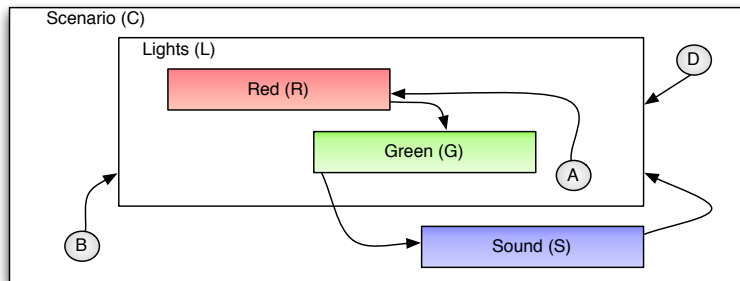


Figure 1.1. Example of a score.

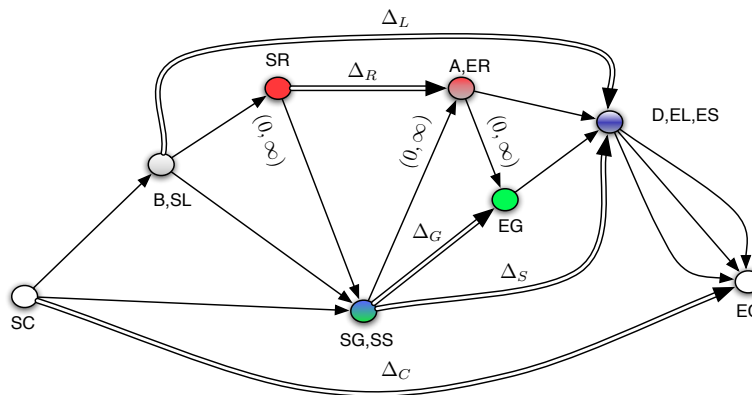


Figure 1.2. Graph representing the score in Fig. 1.1. Label S_t denotes the start point of t and E_t the end point of t . Double arrows represent the duration of temporal objects. Simple arrows with no label represent the constraints imposed by the hierarchy, its duration is $[0, \infty)$.

CLOCK is a process that ticks forever. Process *Score* is parametrized by the graph in Fig. 1.2. We post a constraint $0 \leq p_i < n_\infty$ to allow that $clock + 1 < p_i$ be eventually deduced even in absence of user interactions.

$$CLOCK(k) \stackrel{def}{=} \mathbf{tell}(clock = k) \parallel \mathbf{next} CLOCK(k + 1)$$

$$Score_g \stackrel{def}{=} Points_{IV,A} || Intervals_{IA} || User_{IV} || CLOCK(0) || \prod_{i \in V} 0 \leq p_i < n_\infty$$

1.1.2. Discussion

We presented a model for interactive scores based upon `ntcc` to represent temporal reductions and interactive events. The model is close in spirit to the petri nets model. An advantage of petri nets is that transformation from interactive scores to petri nets is simpler, and synchronization is easier when it depends, for instance, on the transitions that precede a place. An advantage of `ntcc` is that it can easily represent global constraints such number of simultaneous temporal constraints and also temporal reductions.

The `ntcc` model could be implemented. We plan to implement our system using *Ntcrt* [TOR 09], a real-time capable interpreter for `ntcc`. There is an issue with the correctness of the implementation: it heavily relies on propagation. Each time an interactive point is launched, we add a constraint, and we propagate. Does propagation preserves the coherence of the model? Do we need to perform search or propagation is enough? This is an open issue. There is another open issue: We believe that the denotation of a score will help us to understand the behavior of the score without analyzing its operational semantics, to specify properties, and to prove its correctness.

1.2. Timed Conditional Branching Model

There is neither a formal model nor a special-purpose application to support conditional branching in interactive multimedia. Using conditional branching, a designer can create scenarios with choices. The user and the system can take decisions during performance with the degree of freedom described by the designer, and define when a loop ends; for instance, when the user changes the value of a variable.

In the domain of composition of interactive music, there are applications such as *Ableton Live*¹. Using *Live*, a composer can write loops and a musician can control different parameters of the piece during performance. Unfortunately, the means of interaction and the synchronization patterns are limited.

To express complex synchronization patterns and conditions, it was shown in [TOR 10] that interactive scores can describe temporal relations together with conditional branching. In this section, we recall the conditional-branching timed model of interactive scores based upon the *Non-deterministic Timed Concurrent Constraint* (`ntcc`) calculus. We explain how to model temporal relations, conditional branching

1. <http://www.ableton.com/live/>

and discrete interactive events in a single model. We also present performance results of a prototype and we discuss the advantages and limitations of the model.

1.2.1. Specification of the model

A *score* is defined by a tuple composed by a set of points and a set of intervals. A temporal object is a type of interval.

1.2.1.1. Points

We say that a *point* p is a *predecessor* of q if there is a relation p before q . Analogically, a point p is a *successor* of r if there is a relation r before p . A Point is defined by $p = \langle b_p, b_s \rangle$, where b_p and b_s represent the behavior of the point. Behavior b_p defines whether the point *Waits until All* its predecessors transfer the control to it or it only *Wait for the First* of them. Behavior b_s defines whether the point *CHooses* one successor which condition holds to transfer the control to it, or it does *Not CHoose*, transferring the control to all its successors.

1.2.1.2. Intervals

An *interval* p before q means that the system waits a certain time to transfer the control (*jumps*) from p to q if the condition in the interval holds, and it executes a process throughout its duration. An interval is composed by a start point, an end point, a condition, a duration, an interpretation for the condition, a local constraint, a process, parameters for the process, children, and local variables.

There are two types of intervals. *Timed conditional relations* have a condition and an interpretation, but they do not have children, their local constraint is `true`, and their process is *silence*². *Temporal objects* may have children, local variables and a local constraint, but their condition is `true`, and their interpretation is “*when* the condition is true, transfer the control from the start point to the end point”.

A *timed conditional relation* includes the start and end points involved in the relation and the condition that determines whether the control *jumps* from start to the end point. There are two possible values for the interpretation of the condition: (i) *when* means that if condition holds, the control jumps to the end point; and (ii) *unless* means that if the condition does not hold or its value cannot be deduced from the environment, the control jumps to end point. A *temporal object* is an interval where the start point launches a new instance of a temporal object and the end point finishes such instance. Its local variables and local constraint can be used by its children and process to synchronize each other.

2. *silence* is a process that does nothing.

1.2.1.3. Example

Figure 1.3 describes a score with a loop. During the execution, the system plays a silence of one second. After the silence, it plays the sound B during three seconds and simultaneously it turns on the lights D for one second. After the sound B, it plays a silence of one second, then it plays video C. If the variable *finish* becomes true, it ends the scenario after playing the video C; otherwise, it jumps back to the beginning of the first silence after playing the video C.

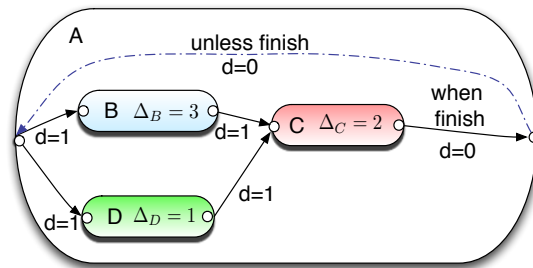


Figure 1.3. A score with a user-controlled loop.

The points have the following behavior. The end point of C is enabled for choice, and the other points transfer the control to all their successors. The start point of C waits for all its predecessors and all the other points only wait for the first predecessor that transfers the control to them. Object A's process is *silence*, its children are B, C and D and its local variable is *finish*. Note that the silence between D and C lasts longer during execution because of the behavior of the points.

1.2.2. Ntcc model

We model points and intervals as processes in *ntcc*. The definition of an interval can be used for both timed conditional relations and temporal objects. We represent the score as a graph; however, there is not yet a procedure to remove zero-delay arcs. Predecessors and successors of point i are $Predec_i$ and $Succ_i$, respectively. For simplicity, we do not include hierarchy, we only model the interpretation *when*, we can only declare a single interval between two points, and we can only execute a single instance of an interval at the same time.

1.2.2.1. Points

We only model points that choose among their successors (*ChoicePoint*), points that transfer the control to all their successors (*JumpToAllPoint*), and points that wait for all their predecessors to transfer the control to them (*WaitForAllPoint*).

To know if at least one point has transferred the control to the point i , we ask to the store if $\bigvee_{j \in Points} Arrived(i, j)$ can be deduced from the store. When all the expected predecessors transfer the control to the point i , we add a constraint $ActivePoints_i$. Analogally, when a point i transfers the control to a point j , we add the constraint $ControlTransferred(j, i)$. In order to represent choice in the example of Fig. 1.1, we use the constraint *finish*.

$$ChoicePoint_{i,a,b} \stackrel{def}{=} ! \text{ when } \bigvee_{j \in P} Arrived(i, j) \text{ do } (\text{tell } (ActivePoints_i) \\ \| \text{ when } finish \text{ do } \text{tell } (ControlTransferred(a, i)) \\ + \text{ when } \neg finish \text{ do } \text{tell } (ControlTransferred(b, i)))$$

The following definition uses the parallel composition agent \parallel to transfer the control to all the successors of the point i .

$$ToAll_i \stackrel{def}{=} \prod_{j \in P} \text{ when } Succs(i, j) \text{ do } \text{tell } (ControlTransferred(j, i)) \\ \| \text{tell } (ActivePoints_i)$$

Using the definition $ToAll_i$, we define the two types of point that transfer the control to all its successors. To wait for all the predecessors, we ask the store if the constraint $Arrived = Predec$ holds.

$$JumpToAllPoint_i \stackrel{def}{=} ! \text{ when } \bigvee_{j \in Points} Arrived(i, j) \text{ do } ToAll_i$$

1.2.2.2. Intervals

Process I waits until at least one point transfers the control to its start point i , and at least one point has chosen to transfer the control to its destination j . Afterwards, it waits until the duration of the interval is over³. Finally, it transfers the control from i to j .

$$I_{i,j,d} \stackrel{def}{=} !(\text{tell } (Predec(j, i)) \| \text{tell } (Succ(i, j))) \\ \| ! \text{ when } \bigvee_{k \in P} ControlTransferred(j, k) \wedge \bigvee_{k \in P} Arrived(i, k) \text{ do} \\ \text{next}^d(\text{tell}(Arrived(j, i)) \| PredecessorsWait(i, j))$$

PredecessorsWait adds the constraint $Arrived(j, i)$ until the point j is active.

1.2.2.3. Example

We can represent the score in Fig. 1.3 in $ntcc$. The start point of *textscc* waits for all the point, the end of *textscc* chooses a point and the other points jump to all points.

3. $next^d$ is a process *next* nested d times ($next(next(next\dots))$).

The intervals have the duration described in Fig. 1.3. We also need to model a user making choices. *User* tells to the store that *finish* is not true during the first n time units, then it tells that *finish* is true. It is initialized with $i = 0$. An advantage of `ntcc` is that the constraint $i \geq n$ can be easily replaced by more complex ones; for instance, “there are only three active points at this moment in the score”.

$$User_n(i) \stackrel{def}{=} \mathbf{when} \ i \geq n \ \mathbf{do} \ \mathbf{tell} \ (finish) \ || \ \mathbf{unless} \ i \geq n \ \mathbf{next} \ \mathbf{tell} \ (\neg finish) \\ || \ \mathbf{next} \ User_n(i+1)$$

1.2.3. Results and Discussion

Performance results are described in [TOR 10]. They ran a prototype of the model over *Ntccrt*. The tests were performed on an iMac 2.6 GHz with 2 GB of RAM under Mac OS 10.5.7. They compiled it with GCC 4.2 and Gecode 3.2.2. The authors of the Continuator [PAC 02] argue that a multimedia interaction system with a response time less than 30 ms is able to interact in real-time with even a very fast guitar player. Response time was less than 30ms for the conditional branching model for up to 500 temporal objects.

An advantage of `ntcc` with respect to previous models of interactive scores, Pure Data (PD), Max and Petri Nets is representing declaratively conditions by the means of constraints. Complex conditions, in particular those with an unknown number of parameters, are difficult to model in Max or PD [PUC 98]. To model generic conditions in Max or PD, we would have to define each condition either in a new patch or in a predefined library. In Petri nets, we would have to define a net for each condition. A disadvantage of this model is that we cannot always synchronize two objects to happen at the same time: one reason is choice and the other is that using *jumps* is not always possible to respect the constraints on their durations.

Chapter 2

Bibliography

- [ALL 06] ALLOMBERT A., ASSAYAG G., DESAINTE-CATHERINE M., RUEDA C., “Concurrent Constraint Models for Interactive Scores”, *Proc. of SMC '06*, May 2006.
- [NIE 02] NIELSEN M., PALAMIDESSI C., VALENCIA F., “Temporal Concurrent Constraint Programming: Denotation, Logic and Applications”, *Nordic Journal of Comp.*, vol. 1, 2002.
- [PAC 02] PACHET F., “Playing with Virtual Musicians: the Continuator in Practice”, *IEEE Multimedia*, vol. 9, p. 77–82, 2002.
- [PUC 98] PUCKETTE M., APEL T., ZICARELLI D., “Real-time audio analysis tools for Pd and MSP”, *Proc. of ICMC '98*, 1998.
- [RAM 06] RAMIREZ R., “A Logic-based Language for Modeling and Verifying Musical Processes”, *Proc. of ICMC '06*, 2006.
- [TOR 09] TORO M., AGÓN C., ASSAYAG G., RUEDA C., “Ntcert: A concurrent constraint framework for real-time interaction”, *Proc. of ICMC '09*, 2009.
- [TOR 10] TORO M., DESAINTE-CATHERINE M., “Concurrent Constraint Conditional Branching Interactive Scores”, *Proc. of SMC '10*, 2010.