A System of Interactive Scores based on Petri Nets

Antoine Allombert, Gérard Assayag, M. Desainte-Catherine

A system of interactive scores based on Petri nets

A. Allombert*, G. Assayag†, M. Desainte-Catherine‡
*Bordeaux 1 University and Ircam
†Ircam
‡Bordeaux 1 University

***Abstract*— We propose a formalism for composition and performance of musical pieces involving temporal structures and discrete interactive events. We use the Allen relations to constrain these structures and to partially define a temporal order on them. During the score composition stage, we use a constraints propagation model to maintain the temporal relations between the structures. For the performance stage, we must allow the composer to trigger the interactive events "whenever" he wants and we have to also maintain the temporal relations in a real-time context. We use a model based on Petri nets for this stage. We also provide a solution to define global constraints in addition of the local temporal constraints inspired by the NTCC formalism.**

I. 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 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 ones. 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 model based on Petri nets. Then we present how we allow the composer to define global constraints and how we maintain them during the performance through a constraints store inspired by the NTCC (Non-deterministic Temporal Concurrent Constraint Calculus) formalism. We have implemented parts of this model in *OpenMusic*, a graphic language for computer assisted composition developed at Ircam.

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.

II. Interactive Scores

We widely presented the model of interactive scores we use in [1]. Thus we just present here the important notions we use in this article.

A. Interactive Score

A score is defined by a tuple $s = \langle t, r \rangle$ 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 [2], 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).

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 control-point.

- 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 texture 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 Allen 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 (Temporal Object) 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 Allen relations. These relations have been introduced by J. Allen

who worked on natural languages in the 80's to formalize the relative positions of temporal intervals [2]. Figure 1 presents these relations.

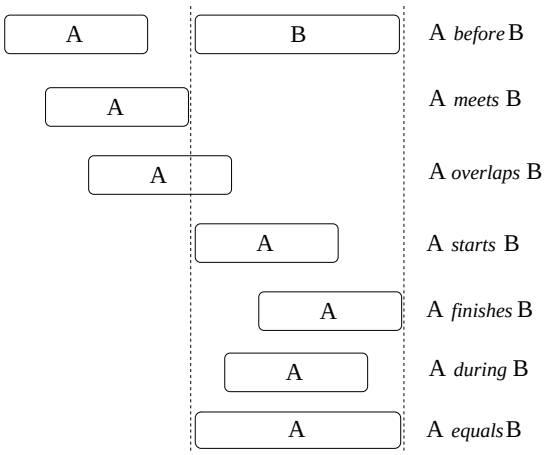


Fig. 1. The Allen relations

The composer 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 automatically added between a TO and its children. Allen relations are only qualitative, while all initial 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 instrumentalist 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 from the composer point of view, 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 exaggerated 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. Addressing 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 2.

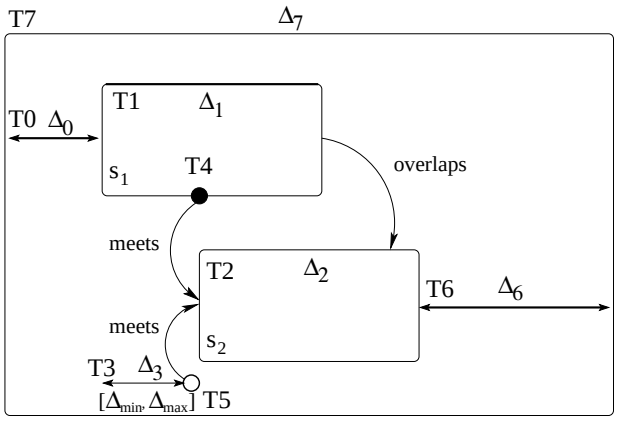


Fig. 2. An example of an interactive score

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

$T0, T3, T6$ 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.

convention we will call s_i and Δ_i the variables defining

the start time and duration of temporal object Ti.

The Allen relations are :

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

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.

At last, 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 [5] 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.

B. The propagation model

During the compositional phase, we face constraints problem when the composer changes the values of the dates of a TO and we have to propagate it through 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. 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 [10], 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.

III. The Real-Time model

To maintain the temporal constraints in a real-time context, we cannot use a propagation model because we cannot control the computing time and then we can't be sure that during the performance, the system will have computed the date of an event before this date occurs. As we explain in [1], we have explored two models, one based on NTCC and an other based on Petri nets. We have developed this last solution.

A. Petri Networks

Petri nets is a general-purpose tool for handling concurrency. It has been used in the computer music field by several authors such as Goffredo Haus [6] who used them for formal representation and structural descriptions of music. In these studies music objects are associated to transformation processes that are described by Petri networks. These studies are based on an analysis of musical pieces and of the compositional process that lead to them. Another study on Petri nets for computing music was carried by Travis Pope [9] for his system "DoubleTalk" used for automatic music generation.

Our interest in the Petri nets is different, we use them to manage parallelism between our textures seen as autonomic process that must synchronize which is the case with an interactive score. During the composition of a score, we have seen that the composer defines a partial order between the TOs with the Allen relations. During the performance the events will admit a total order, but because of the interaction points, several total orders can occur. Trying to specify all this eventual orders through a finite automata would lead us to very high number of states. Petri nets allow us to specify a partial order and to handle concurrency. We have presented a first approach of the use of Petri nets to design a system of interactive scores in [4].

B. Formal semantics

A general presentation of Petri nets can be found in [7]. Formally, a Petri Net is a bi-partite directed graph. The two types of vertice are named "places" and "transitions". A last each place contains a number of tokens greater or equal to zero.

Definition 1: A Petri net is tuple $PN = (P, T, V, d)$, where

- P is the set of places
- T is the set of transitions
- $V = (P \times T) \cup (T \times P)$ is the set of arcs
- $d : P \rightarrow \mathbb{N}$ is the distribution of the tokens among the places

Every transition contains a condition which have to be satisfied for tokens to cross it. Moreover, all places admitting an arc towards a transition t have to contain at least a token for the transition t to be passed. When a transition t is passed, one token is removed from all places preceding t and one token is added to all places admitting an arc coming from the transition t. Then, execution of a Petri network is a sequence of tokens moves.

C. Places

In our model we store the events of the score in the places of the net. So each place contains one or more events which are launched when a token is created in this place. The case of a place with more than one events represents the case of several events that must be launched at the same time. Once a token is created in a place, we launch every events it contains.

D. Transitions

In our use of Petri nets, the conditions on the transitions are of two types :

- time-conditions which mean that after every place with an arc toward the transition contains a token, the system must wait at least a certain time before crossing the transition.
- control-conditions which mean that the system must wait the trigger of a discrete control by the musician before crossing the transition.

This formalism allows us to manage concurrency since some parts of a Petri net can execute independently from each other while we can synchronize such parts with the transitions. This is typically what we need to express the partial order of interactive scores. In our case, the source of the net is the beginning of the musical piece. When the performance starts the distribution of tokens is such that there is only one token over the net which is contained in the place of the event S_P with P the musical piece. Then, the transitions provide a way to wait for time intervals that must be respected or input control from the musician. So the performer can express through the interaction points while the temporal relations are maintained.

So we have to convey the information of an interactive score into a Petri net. We present an algorithm which solves part of this problem in the next section.

IV. The transformation Algorithm

A. The elementary transformations

For the moment, our algorithm doesn't hold the supple intervals, so all intervals of the score are supposed to be rigid. The main idea of this algorithm is to reduce the Allen temporal relations between the TOs to elementary temporal constraints between the events of this TOs and then translate this elementary constraints in elementary configurations of Petri nets. The figures 3 and 4 give the list of this elementary transformations.

For example, if the composer defines a relation $TO_2\ during\ TO_2$ such as in the figure 3, this implies some temporal constraints between the dates of the events S_{TO_1}, E_{TO_1}, S_{TO_2} and E_{TO_2} which are

$$S_{TO_2} = S_{TO_1} + \Delta_2$$

$$E_{TO_1} = E_{TO_2} + \Delta_3$$

In these constraints are translated into a configuration of transitions and arcs between the place associated to the events of T_1 and T_2, as shown on the figure.

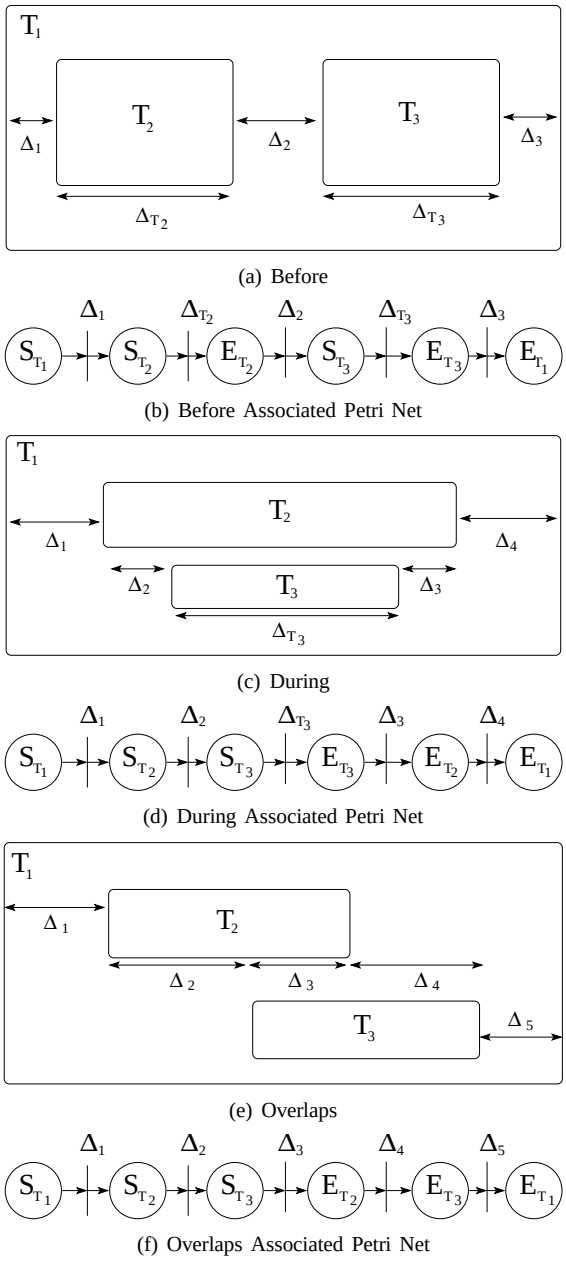
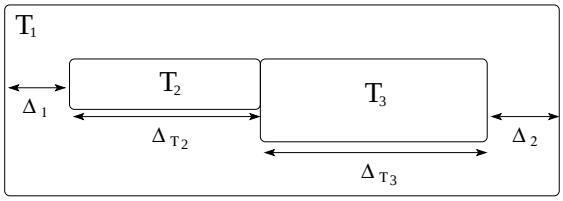


Fig. 3. The set of elementary transformations without merge of place

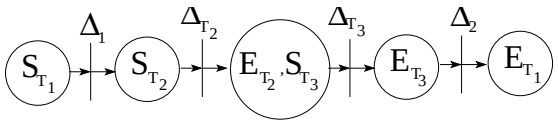
We also have to consider two types of implicit constraints :

- the hierarchy of the piece that allows some TOs to contain other TOs implies that for every TO n contained in a complex texture ct, there is an implicit relation $n\ during\ ct$.
- the integrity of each texture n implies a constraint $E_n = S_n + \Delta_n$ where Δn is the duration of n.

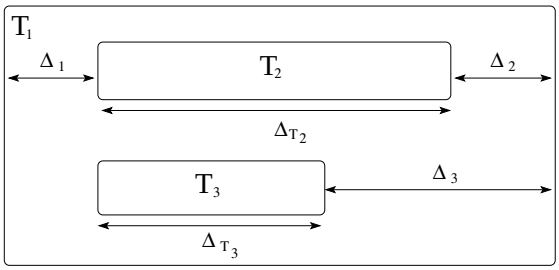
We split the Allen relations in two parts : those which don't need merging places during their transformation and those which need it. Of course those which need this operation are the relations which imply a constraint of equality on the dates of two events. We treat these ones separately from the others because the merge of two places can create duplications of arcs and each time we



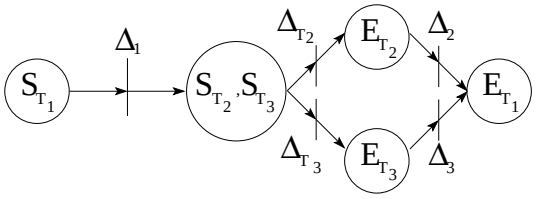
(a) Meets



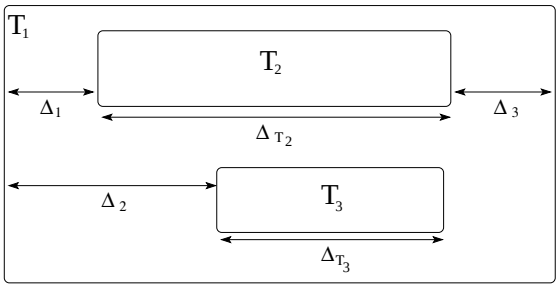
(b) Meets Associated Petri Net



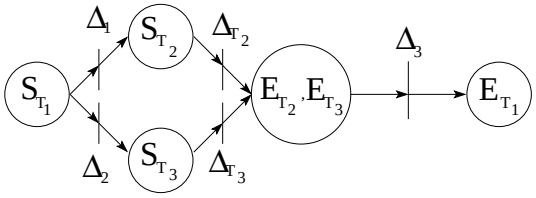
(c) Starts



(d) Starts Associated Petri Net



(e) Finishes



(f) Finishes Associated Petri Net

Fig. 4. The set of elementary transformations with merge of place

merge a place we have to run some verification methods to prevent duplications.

At last for representing interactive events, we must bound the trigger of these events with the incoming of the control message from the musician. Thus after having created the Petri net of the score representing every static information, for each interactive events, we turn the condition of every transitions which admit an arc toward the place representing the interactive event into waiting for the control message associated with the event.

B. The algorithm

Now we can present the algorithm. This version is designed for a piece with no complex texture but it can be generalize for every interactive scores.

$P = (TEXT, IE, R)$ where

- $TEXT$ is the set of textures of P
- IE, the set of interactive events
- R, the set of Allen relations between the TO's, that we split in two sets : $R_{no-merge}$, R_{merge}

In addition, for a texture n of a piece P, $ir(n)$ represents the implicit constraints involving n i.e the relations $n\ during\ P$ and the constraint for maintaining the integrity of n as explicated before. For a relation r and a Petri net PN, we define a method $elementary - transformation$ such as $elementary - transformation(r, PN)\ PN$ in applying the elementary transformation associated with r as described in the figures 3 and 4. We also define a method $add - texture$ such as, with n a texture, $add(n, PN)$ modifies PN in adding two places, one with the event S_n and an other E_n. At last, for an interactive event e, we define a function $add - interactive - event$ such as $add - interactive - event(e, PN)$ turns the condition of the transitions preceding the place of e in PN into waiting the control message associated with e.

For $P = (TEXT, IE, R)$:

$Create\ an\ empty\ Petri\ nets\ PN$

$add - texture(P, PN)$

For each $text\ \in\ TEXT\ add - texture(text, PN)$

For each $text\ \in\ TEXT$

  For each $r\ \in\ ir(text)$
  $elementary - transformation(r, PN)$

For each $r\ \in\ R_{no-merge}$
$elementary - transformation(r, PN)$

For each $r\ \in\ R_{merge}$
$elementary - transformation(r, PN)$

For each $e\ \in\ IE$
$add - interactive - event(e, PN)$

The figure 5 gives an example of the application of this algorithm on an interactive score. In this example, the interactive event T_6 is controlled by the message X.

The proof of this algorithm is not yet designed but we have got several ideas on it. First we should specify the class of the Petri nets which are produced by the algorithm. Indeed, the ensemble of Petri nets representing an interactive score shows common characteristics that we should clearly formalize. Then we assume that the proof consists in exhibiting a bijection from the ensemble of constraints systems involving the events (and not the ensemble of scores) to our particular class of Petri nets. This bijection must be found from the ensemble of constraints systems involving the events and not the ensemble of scores because a Petri net translates constraints between the events and different scores can lead to the same constraints between events and then the same Petri net. The simplest example is a piece with 3 textures A, B, C and the relations $A\ meets\ B$ and $C\ starts\ B$. The constraints system that stems from this piece is :

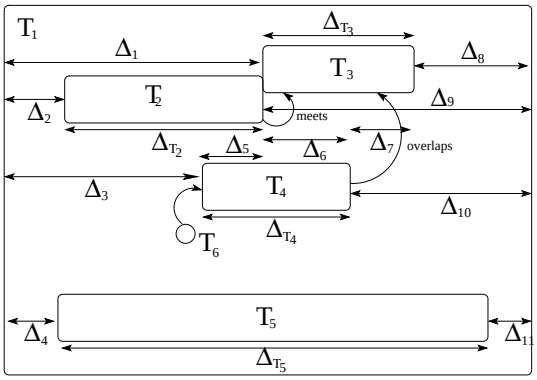
- $date(E_A) = date(S_B)$
- $date(S_B) = date(S_C)$

But the score with the same textures A, B, C and the relations $A\ meets\ C$ and $C\ starts\ B$ gives the same constraints system. So this proof that will be our next work will involve the constraint systems between the events.

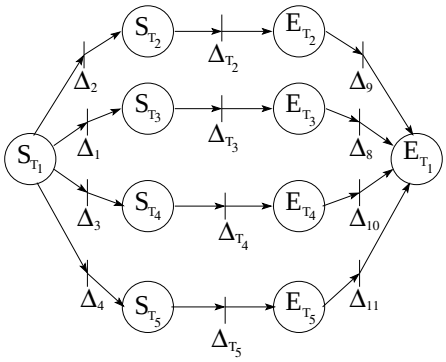
V. The global constraints

The set of Allen relations we use, allows us to design local constraints between the events of scores, but it is impossible to use it in order to design some global constraints over whole parts of the scores. We were given notice that this possibility could relevantly increase our model of interactive scores. We can imagine several types of global constraints. Since we don't interest in the sound parameters of the textures here, the first we imagine is a limitation on the number of simultaneously played textures. But when the parameters of the textures are considered a lot of constraints can be imaged such as harmonic relations between the played textures for example. The use of the global constraints is closely bound to the interactive events because during the composition the system maintains the global constraints in allowing or not the composer to add some textures or change the position of existing textures. Thus, after the compositional phase, the score is such that every global constraints is satisfied, but during the performance the launch of the interactive events when the musician decides can locally modify the organization of the score and then break some global constraints.

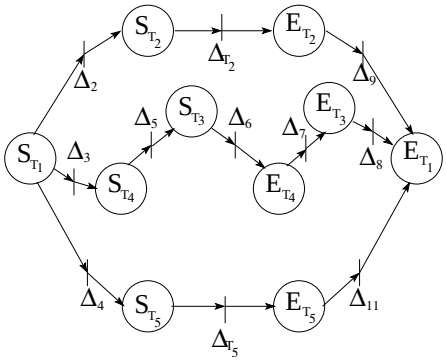
To manage this growth of our model, we use a system derived from the NTCC language. A full description of the NTCC formalism can be found in [8]. Initially, we thought using NTCC as an alternative to Petri nets for the real-time system but this idea has not yet been explored. We use a store of constraints such as in the NTCC formalism but we don't update it at each clock step.



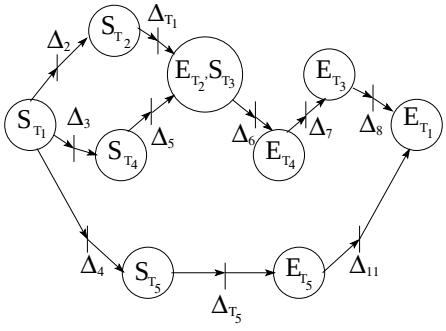
(a) The Interactive Score



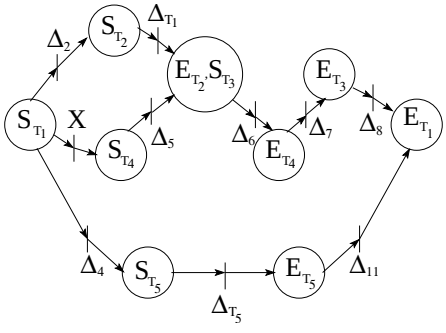
(b) The associated Petri Net after the first step of the algorithm



(c) After the transformation of *overlaps* relation

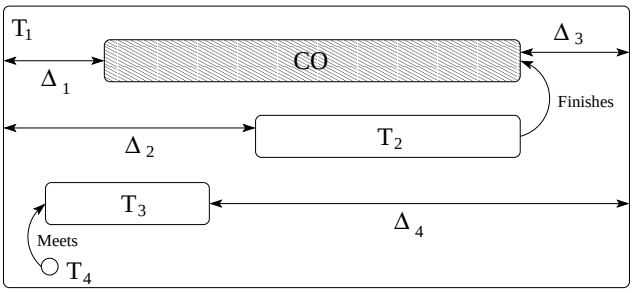


(d) After the transformation of *meets* relation

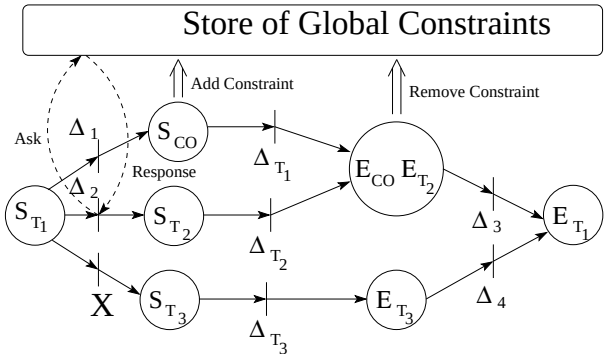


(e) After the transformation of T_6

Fig. 5. An example of the transformation of an interactive score into a Petri net



(a) An interactive score with global constraints



(b) The associated Petri Net

Fig. 6. An example of the use of global constraints

Thus, the composer is allowed to define some global constraints through special temporal objects called *constraints objects* These *constraints object* are used to manage a set of global constraints during the performance. We show the use of these objects in figure 6 ; in this example, the temporal object CO express the will of the composer that during CO the whole piece will be constrained by a global constraint. These *constraints objects* can be related to other TOs with the Allen relations in order to synchronize the changes of the global constraints set with the events of the score. For the performance, we add a constraints store to the Petri net and the effect of the events of the *constraints objects* (*start* and *end*) change the store in adding or removing global constraints.

During the performance, when a transition is crossed, the system ask the constraints store if creating a token in the place which follows the transition and then launching the events it contains will or will not break constraints of the store. If no constraint is broken by the launch of the events then they are directly launched. If a constraint is broken by the launch of an event, several strategies can be adopted :

- we can wait until the launch of the events of the place does not break any constraint of the store
- we can not launch the events and then mute the corresponding texture
- we can modify the parameters of the texture in such a way that as the launch of the events does not break the constraints in the store.

The figure 6 explicits the mechanism of global constraints. Suppose that the object CO holds the global constraint on the number of simultaneously played textures and limit it to only 1. During the compositional phase, after the composer introduces CO, the system automatically prevents the composer from designing more than one texture playing simultaneously, for example the composer cannot organize the score such as T_2 and T_3 play in the same time. But during the performance, since the interactive events T_4 controls the trigger of S_{T_3}, the musician can delay the start of T_3. If this delay is too important, the start of T_2 which is static could occur while T_3 is still playing. In this case, the system cannot launch S_{T_2} because it would break the global constraint. The strategy adopted by the system at this moment has been chosen by the composer, indeed during the composition the composer can define for each texture the behavior of the system in case of global constraint breaking. So the composer can choose if the texture can be mute or modified. For the constraint on the number of simultaneously played textures there is no parameters to modify to prevent from constraint breaking, so in our example if T_2 is unmutable, the system will wait the end of T_3 before triggering T_2, if T_2 is mutable, the system will simply mute it. In the case of a constraint on the sound parameters, if the composer defines T_2 as modifiable, our idea is to run a constraint satisfaction algorithm to compute new parameters for T_2 which don't break the global constraint and then start T_2 with these new parameters but this solution has not yet been implemented. With our example we can show how we hold the modifications of the list of global constraints during the performance, the events of the CO which are stored in places of the Petri net as the events of the texture, change the contain of the store.

The introduction of global constraints in the model leads us to develop several verification algorithm for the compositional phase in order to prevent the composer from defining configurations of textures, Allen relations, interactive events and global constraints that leads to unplayable situations during the performance. Indeed, some configurations could lead to freezing the evolution of the piece during the performance. Such algorithms has not yet been developed.

VI. Architecture and implementation

The model and solutions presented in this paper has been partially implemented in OpenMusic. Since we don't hold the sound parameters, for the moment, the textures simply send OSC messages, one associated to the start of the texture and another associated to the end of the texture. These messages are sent to an external application which holds them and starts or stops some processes associated with the textures. The general architecture of our implementation is shown in the figure 7.

We call musical context the ensemble composed by the Petri net and the store of global constraints. The interpretation of this musical environment consists in running the Petri net as described before, holding the controls input and sending the OSC messages associated to the events.

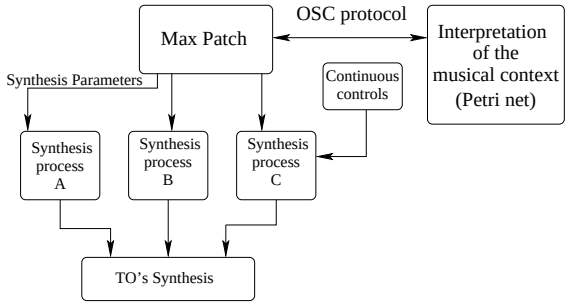


Fig. 7. The architecture of the system

VII. Applications

Relatively to the previous section, we can see that one of the direct applications of this system is to propose an original way to deal with the time-line in Max/MSP. Several composers and performers told us that the management of the time-line is one of the deficiencies of Max and that they have to develop clever devices to deal with time-line. Our system will allow composers to schedule the sending of OSC messages to Max. Then, the composers will design patches directly in Max and schedule the trigger and release of them through an interactive score.

The initial application is of course to provide composers a tool that brings a formalism of interpretation taken from music in electro-acoustic music. In extension, this tool will allow composers to define static scores and design different ways to interpret them. Indeed, a composer could create several configurations of interactive events for the same static score and then propose different ways to perform his piece of music. For example, composers could define a version for multiple performers and another for only one performer in decreasing the number of interactive events. We can also imagine that performers could change the number of interactive events by themselves and then adapt an interactive score to their skills or ability. Then a beginner, a virtuoso or a handicapped performer could interpret the same score in adapting the difficulty of execution.

One can also imagine pedagogical applications, for example a performer could progressively increase the number of interactive events during practicing to play a piece. Then, we can see that a performer can begin to practice the interpretation of a piece before technically be able to play it.

A last, we can see that this system can provide an accurate system of play-back since we can synchronize temporal objects with the controls triggered by performers.

VIII. Conclusion and future work

We presented in this paper how we developed a system for composing interactive scores and playing them. We used constraints propagation scheme for editing the scores and a model based on Petri nets for managing the performance phase in holding the control inputs and sending the musical parameters. We also present how we increased our model with global constraints. In addition, we have implemented parts of this system in OpenMusic with encouraging preliminary results.

The very next step of this work will consist in enlarging our model of Petri nets for holding the supple intervals. Then we will change the transformation algorithm for creating the new type of Petri nets and design a complete proof of this algorithm. Once this theoretical step will be finished, we will extend our implementation and make it fully usable for composers and performers in order to receive feed-backs on the use of the model.

References

[1] M. D.-C. A. Allombert, G. Assayag and C. Rueda, *Concurrent constraints models for interactive scores*, in Pr. of Sound and Music Computing 2006, GMEM, Marseille, France, May 2006.

[2] J. Allen, *Maintaining knowledge about temporal intervals*, Communications of the ACM, 26 (1983), pp. 832–843.

[3] A. Beurivé, *Un logiciel de composition musicale combinant un modèle spectral, des structures hiérarchiques et des contraintes*, in Journées d'Informatique Musicale, JIM 2000, 2000.

[4] M. Desainte-Catherine and A. Allombert, *Specification of temporal relations between interactive events*, in Pr. of Sound and Music Computing 2004, IRCAM, Paris, France, October 2004.

[5] M. L.-C. A. Gérard Assayag, Camilo Rueda and O. Delerue, *Computer assisted composition at ircam : From patchwork to openmusic*, Computer Music Journal, 23 (1999).

[6] G. Haus and A. Sametti, *Scoresynth : a system for the synthesis of music scores based on petri nets and a music algebra*, IEEE Journal, 24 (1991), pp. 56–60.

[7] T. Murata, *Petri nets: Properties, analysis and applications*, in Proceedings of the IEEE, vol. 77(4), April 1989, pp. 541–580.

[8] C. Palamidessi and F. Valentia, *A temporal concurrent constraint programming calculus*, in Proc. of the Seventh International Conference on Principles and Practice of Constraint Programming, CP2001, 2001.

[9] S. Pope, *The development of an intelligent composer's assistant: Interactive graphics tools and knowledge representation for composers.*, in Pr. of the 1986 International Computer Music Conference, The Hague, October 1986.

[10] C. Schulte and G. Tack, *Views and iterators for generic constraint implementations*, in Pr. of the Fifth International Colloqium on Implementation of Constraint and Logic Programming Systems, CICLOPS05, 2005.