UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali

Corso di Laurea in Scienze dell'Informazione

UN SISTEMA PER LA GENERAZIONE DI DOCUMENTI MUSICALI XML BASATO SU RETI DI PETRI

Relatore:

Chiar.mo Prof. Goffredo Haus

Correlatore:

Dott. Luca Ludovico

Tesi di Laurea di

Adriano Baratè

Matr. 498415

Anno Accademico 2003-2004

RINGRAZIAMENTI

Un ringraziamento particolare al prof. Goffredo Haus, per avermi consentito di cimentarmi in questo lavoro, ed ai membri del LIM, in particolare a Maurizio Longari, Luca Ludovico e Dante Tanzi, per i preziosi suggerimenti.

Grazie al "collega" Stefano Spagliardi, per la fruttuosa collaborazione.

Grazie infinite alla Musica, ed a coloro che nell'arte e nella tecnologia cercano di guardare sempre *oltre*.

INDICE

CAPITOLO 1

INTRODUZIONE

1.1 DESCRIZIONE DELLA TESI

Alla fine degli anni ’80 all’interno del LIM (Laboratorio di Informatica Musicale) [LIM] viene realizzato un ambiente di progettazione di Reti di Petri, un particolare tipo di grafi, che consente di trattare informazione musicale in formato MIDI, con l'idea di scomporre un documento musicale in frammenti tematici che si ricompongano tramite un processo d’esecuzione della rete.

L’applicazione realizzata, chiamata ScoreSynth, acquisisce subito una certa rilevanza in ambito di ricerca, e da essa si snodano una serie di altri studi complementari.

L’informatica però è una delle discipline che più risente del passare del tempo, con avvicendamenti tecnologici velocissimi. Questo ha portato, nell’informatica musicale, a tralasciare il formato MIDI in ambito di ricerca, semplicemente perché non abbastanza duttile per rappresentare nuovi tipi di informazione musicale.

Negli ultimi anni, sempre all’interno del LIM, viene sviluppato un nuovo tipo di codifica di informazioni musicali, l’MX, più moderna anche nell’adozione del formato XML, una delle tecnologie più potenti e semplici allo stesso tempo, che può essere visto come il classico “uovo di Colombo”.

Il presente lavoro di tesi intende ripresentare la concezione di Rete di Petri Musicale, adattando gli strumenti all’ambiente Windows (il precedente progetto era sviluppato su Apple Macintosh) e al nuovo formato MX, sostituendolo al MIDI. La possibilità offerta all’utente è di costruire una Rete di Petri, utilizzando la palette di strumenti dell’interfaccia, associare frammenti musicali MX a posti e transizioni, ed eseguire la rete, creando in output un file MX in cui vengono combinati i frammenti iniziali.

Il progetto sviluppato consta di una libreria dinamica, riservata al trattamento delle Reti di Petri, costruita modularmente e con tecnologia object oriented, e di un'interfaccia di gestione che permette di interagire con le reti il più intuitivamente possibile. Il nome del progetto è stato mantenuto uguale a quello dell'applicazione da cui deriva: ScoreSynth.

E' stato utilizzato l'ambiente di programmazione Microsoft Visual Studio .NET 2003. Il codice è stato scritto in linguaggio C#, l'ultimo nato in casa Microsoft. La scelta è stata dettata dalla volontà di utilizzare strumenti all'avanguardia, potenti e flessibili, con un basso costo di realizzazione dell'interfaccia, caratteristica che permette di focalizzare l'attenzione sulle caratteristiche sostanziali del progetto, ed allo stesso tempo di creare applicazioni che consentano all'utente di trovarsi subito a proprio agio e di diventare immediatamente produttivo.

1.2 ORGANIZZAZIONE DELLA TESI

La tesi è suddivisa in 10 capitoli ed un'appendice.

Il capitolo 2 presenta il contesto passato e presente relativo agli argomenti trattati nel lavoro, con una panoramica degli altri tools disponibili per il trattamento dei tipi di informazioni trattate.

I capitoli 3, 4 e 5 introducono le tre tecnologie fondamentali dell'applicazione: le Reti di Petri, il formato MX ed il formato PNML, in cui le reti create vengono salvate su file.

Il capitolo 6 fornisce una piccola rassegna di esempi di utilizzo dell'applicazione, ponendo l'attenzione sulle innovazioni apportate rispetto alla precedente versione di ScoreSynth.

I capitoli 7 e 8 sono rivolti agli sviluppatori e descrivono nel dettaglio il progetto, gli algoritmi utilizzati e alcuni esempi di riutilizzo del codice creato.

Il capitolo 9 approfondisce un problema di carattere musicale derivante dalla semantica del formato MX in casi particolari di utilizzo dell'applicazione.

L'ultimo capitolo presenta alcune idee di sviluppo che possono essere implementate in versioni future dell'applicazione.

Segue un'appendice contenente il manuale operativo di ScoreSynth.

CAPITOLO 2

ESPERIENZE PRECEDENTI

2.1 RETI DI PETRI

Le Reti di Petri sono state formalizzate per la prima volta da C. A. Petri negli anni ’60. E’ perciò naturale che in campo informatico vi sia un’ampia rosa di applicazioni che utilizzano questo modello formale.

Tra i tools commerciali general-purpose si segnalano ad esempio ALPHA/Sim® [ALPHA] e IBE PACE [PACE], applicazioni che integrano editor, simulatori, strumenti di analisi e di sviluppo, e operano su estensioni delle Reti di Petri che consentono grandi complessità di modellazione.

Un grande apporto alla diffusione del modello è data da tools freeware, anche molto avanzati, che proliferano sempre più spesso. Alcuni esempi sono Renew [RENEW], basato su Java, e PEP (Programming Environment based on Petri Nets) [PEP], un insieme di componenti per il trattamento di Reti di Petri basato su un’interfaccia grafica in Tcl/Tk.

Design/CPN [DESCPN], ora sostituito da CPN Tools [CPNTOOLS], prende in considerazione un’estensione molto interessante: le Reti di Petri Colorate.

Infine, segnaliamo il Petri Net Kernel [PNK], implementato in Python e Java, un framework per costruire applicazioni che lavorano su Reti di Petri. Dopo un’iniziale analisi, pur essendo questo un approccio interessante, è stato scartato per il presente lavoro; lo sforzo necessario per costruire l’applicazione in questo modo è sembrato rapportabile a quello eseguito utilizzando Microsoft Visual Studio .NET, con il vantaggio per quest’ultimo di un pieno controllo sotto tutti gli aspetti.

2.2 MX

Il formato MX è stato creato all'interno del LIM (Laboratorio di Informatica Musicale dell'Università degli Studi di Milano) [LIM], si basa sulla tecnologia XML, ed è tuttora in fase di aggiornamento ed ampliamento.

Altri formati che utilizzano l'XML per codificare informazione musicale sono ad esempio l'SMDL (Standard Music Description Language) [SMDL], che incorpora una struttura a livelli simile all'MX, e il MusicXml [MXML], creato dalla Recordare, che si è diffuso anche per l'integrazione come plug-in dell'applicazione per l'editing di partiture musicali per eccellenza: Finale [FINALE].

Vari progetti e Tesi di Laurea sono naturalmente stati intrapresi al LIM per affiancare alla stesura del formato MX tools di trattamento dei documenti creati.

2.3 RETI DI PETRI ED INFORMATICA MUSICALE

L'utilizzo delle Reti di Petri per il trattamento di informazione musicale inizia all'interno del LIM alla fine degli anni '80 [Hau84], portando alla creazione di alcune applicazioni nate da Tesi di Laurea, quali ScoreSynth e MediaSynth.

2.3.1 SCORESYNTH

Sviluppato in ambiente Macintosh da A. Sametti [Sam88], ScoreSynth permette di sintetizzare files MIDI partendo da oggetti musicali di base associati ai posti ed algoritmi di modifica associati alle transizioni di una Rete di Petri.

Dopo una serie di evoluzioni si è giunti alla versione 3.0 [Sam92], che si integra nella Stazione di Lavoro Musicale Intelligente [SLMI] sviluppata al LIM nel 1994.

In ScoreSynth sono presenti 3 diversi tipi di utilizzo, l'Editor, il Performer ed il Debugger. Nell'Editor viene disegnata la Rete di Petri e vengono associati a posti e transizioni rispettivamente oggetti musicali (files MIDI o metalinguaggio descrittivo che viene compilato) ed algoritmi (che, seguendo una particolare grammatica, consentono di applicare alle note operazioni di pitch shift, inversione temporale, rotazioni, moltiplicazioni, ecc...). Nel Performer viene eseguita la rete, creato l'output nel formato proprietario .SCORE, ed è possibile ascoltare attraverso una periferica MIDI l'output musicale generato. Nel modulo

Debugger, è possibile controllare l'evoluzione della Rete di Petri, eseguendola step-by-step e visualizzando alcune finestre di riepilogo dei posti, delle transizioni, dei tokens, delle reti aperte, degli oggetti eseguiti.

Attraverso l'uso di ScoreSynth sono stati effettuati due importanti studi sulla struttura del Bolero di Ravel [HR93] e della Sagra della Primavera di Stravinsky [DeMH93].

Lo ScoreSynth presentato in questo lavoro di tesi riprende il discorso interrotto dalla versione 3.0[1], adattandolo alla nuova tecnologia MX sviluppata al LIM ed all'ambiente Microsoft Windows, cercando di mantenere inalterate le idee di base, risultate di sperimentata efficacia, e migliorando alcuni aspetti ove possibile.

2.3.2 MEDIASYNTH

Utilizzando l'esperienza accumulata con ScoreSynth di A. Sametti, all'interno del LIM è stato sviluppato nel 1994 il progetto MediaSynth, che, condividendo l'Editor ed il Debugger con il primo, muta il campo di applicazione.

Con MediaSynth infatti vengono trattati al posto di dati MIDI, files multimediali (si basa sul QuickTime 1.6.1). Alle transizioni vengono associati effetti audio o video.

[1] Allo ScoreSynth "nuova generazione" viene assegnato comunque il numero di versione 1.0, vista la differenza di ambiente e di soluzioni rispetto a quello creato da A. Sametti.

CAPITOLO 3

CENNI SULLE RETI DI PETRI

3.1 INTRODUZIONE

Una Rete di Petri è un modello astratto e formale atto a rappresentare la dinamica di un sistema che esibisce attività asincrone e concorrenti [Sam88].

Poiché la presente trattazione è rivolta anche ai musicisti che intendono sperimentare le potenzialità dell'approccio di ScoreSynth, verrà fornita una trattazione intuitiva e informale delle Reti di Petri.

Il lettore più interessato potrà comunque trovare trattazioni più rigorose nell'ampia bibliografia dedicata all'argomento, ad esempio in [Pet76], [Val78], [Pet81].

Per tutto ciò che concerne le Reti di Petri, si segnala il sito [PNW], in cui è anche presente una bibliografia veramente sconfinata (più di 8500 titoli!).

3.2 DEFINIZIONI DI BASE

Le Reti di Petri (PNs – Petri Nets) consistono di un insieme di oggetti elementari: *posti*, *transizioni* ed *archi*, rappresentati rispettivamente da cerchi, rettangoli e linee orientate. I posti e le transizioni vengono anche detti *nodi*.

Per formare una PN questi elementi si combinano tra loro, con l'accortezza di rispettare alcuni vincoli, per esempio che un arco può connettere solo nodi di tipo diverso.

Il nodo di partenza di un arco si dice *di entrata*, mentre quello d'arrivo *di uscita*.

In riferimento alla Fig. 1: P1, P2 e P3 sono posti, T1, T2 e T3 transizioni. P1 è un posto di ingresso delle transizioni T1 e T2; T3 è una transizione d'uscita di P2 e P3.

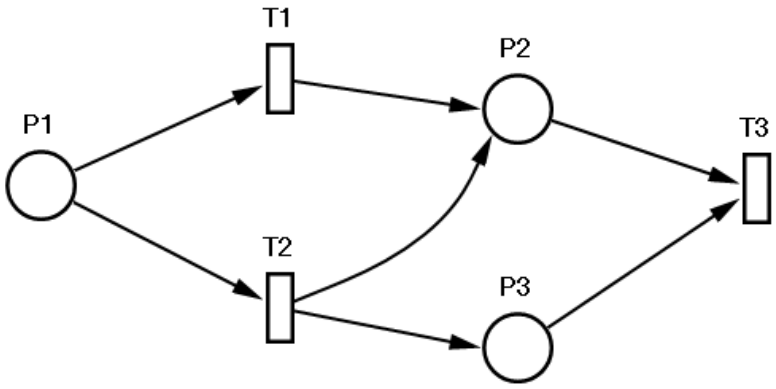


Fig. 1. Esempio di PN

Due nodi possono avere più di un arco che li connette dello stesso orientamento. Una scorciatoia per indicare che esistono *n* archi tra due nodi è di disegnare un unico arco e associargli un numero positivo, detto *peso n*.

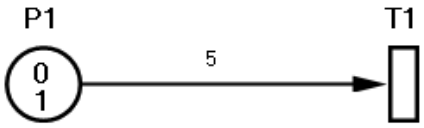


Fig. 2. Arco di peso 5

Un elemento delle PNs che gli consente di automodificarsi è il concetto di *marca*: ad ogni posto viene associato un numero di marche presenti (In Fig. 2 il posto P1 ha 0 marche).

Seguendo alcune regole le marche possono essere spostate di posto in posto e il sistema si evolve.

Le regole che governano questa evoluzione dinamica sono:

- Se tutti i posti di ingresso di una transizione hanno un numero di marche maggiore o uguale al peso dell'arco di ingresso, la transizione si dice *abilitata allo scatto*.
- Se una transizione è abilitata allo scatto, l'esecuzione dello scatto toglierà dai posti in ingresso un numero di marche pari al peso dell'arco in ingresso ed aggiungerà ad ogni posto in uscita tante marche quanto è il peso dell'arco in uscita (v. Fig. 3).

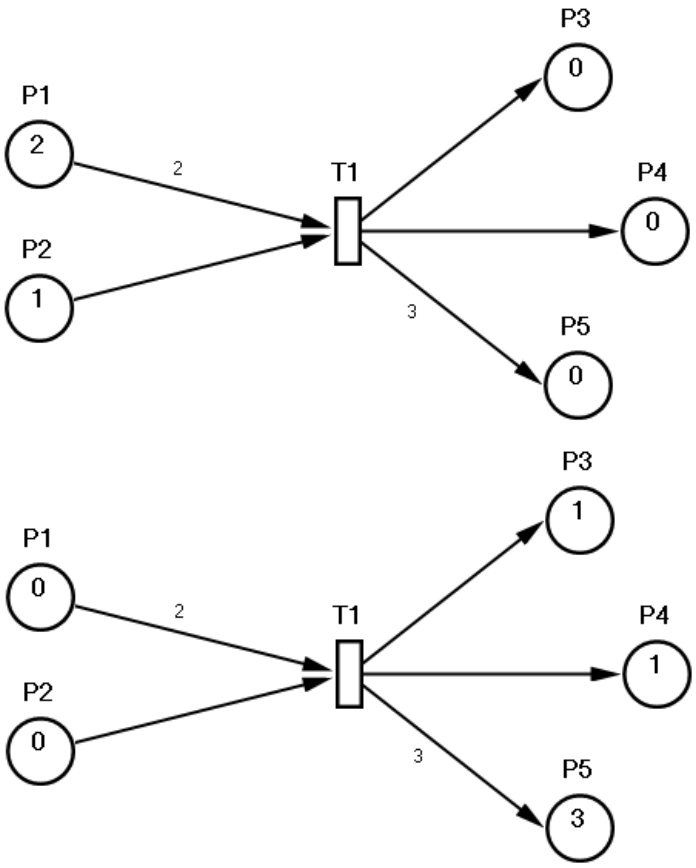


Fig. 3. Scatto di una transizione

Se una transizione è abilitata non significa che debba scattare per forza. A questo proposito si veda la Fig. 4: entrambe le transizioni sono abilitate, ma in ingresso non ci sono abbastanza marche per farle scattare tutte e due. In questo caso le transizioni vengono dette *in alternativa*, e soltanto una di esse scatterà.

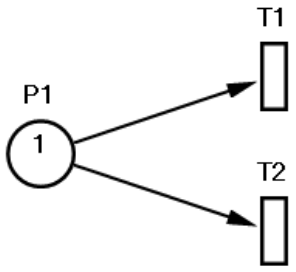


Fig. 4. Transizioni in alternativa

3.3 DEFINIZIONI FORMALI DI PNS

Diamo ora alcune definizione sul concetto di PN. Da notare che esistono diversi tipi di PNs: la definizione seguente è quella di PN generale [Rei00].

*Una **PN** è una tripla:*

PN = (P, T, A)

dove P è detto insieme dei posti, T è detto insieme delle transizioni ed A è detto insieme degli archi. Inoltre devono valere le proprietà:

1. $P \cap T = \varnothing$
2. $P \cup T \neq \varnothing$
3. $A \subseteq (P \times T) \cup (T \times P)$
4. $dom(A) \cup ran(A) = P \cup T$, dove
   $dom(A) = \{x \in P \cup T : (x,y) \in A \text{ per qualche } y \in P \cup T\}$
   $ran(A) = \{y \in P \cup T : (x,y) \in A \text{ per qualche } x \in P \cup T\}$

La 1 afferma che gli insiemi dei posti e transizioni sono disgiunti. La 2 che la rete non deve essere vuota. La 3 che un arco A connette soltanto nodi di tipo diverso. La 4, infine, che non devono esistere nodi isolati, ovvero senza archi entranti o uscenti.

Il tipo di PN trattato da ScoreSynth incorpora alcune estensioni alla definizione generale: la definizione seguente, tratta da [Des00], meglio si adatta al nostro scopo.

*Una **PN P/T** è una tupla:*

PN = (P, T, A, c, p, m_0)

dove (P, T, A) è una PN generale, con
P : insieme dei posti, non vuoto, finito
T : insieme delle transizioni, non vuoto, finito

$A \subseteq (P \times T) \cup (T \times P)$: *insieme degli archi*

$c : P \rightarrow \{1, 2, 3, ...\}$: *capacità dei posti*

$w : A \rightarrow \{1, 2, 3, ...\}$: *peso degli archi*

$m_0 : P \rightarrow \{0, 1, 2, ...\}$: *marcatura iniziale dei posti (deve essere:*

$$\forall\ p \in P : m_0(p) \leq c(p)\ )$$

Un nodo (posto o transizione) può avere più di un nodo del tipo opposto in ingresso o in uscita. Questo fa sì che più archi uniscano un posto ad una transizione. Il numero di tali archi è la *molteplicità* o *peso* di un arco.

3.4 ESTENSIONI

Il modello di rete fin qui descritto è chiamato *Posti/Transizioni*, ed incorpora alcune estensioni e modifiche rispetto alla definizione di PN generale. Noi analizzeremo nei successivi paragrafi le estensioni che ci interessano direttamente.

3.4.1 CAPACITÀ

La *capacità* è una proprietà dei posti che definisce il numero massimo di marche ammesse per ogni nodo. Questa caratteristica aggiunge una nuova condizione per l'abilitazione di una transizione:

- Una transizione non è abilitata se la marcatura di almeno un nodo di uscita supera la sua capacità dopo lo scatto della transizione.

L'introduzione della capacità ha come conseguenza anche l'aggiunta di un nuovo caso di non determinismo, denominato *conflitto*, che si presenta quando due o più transizioni sono abilitate e lo scatto di una inibisce lo scatto delle altre.

Questa inibizione è data dal fatto che lo scatto di tutte le transizioni porterebbe nei posti di uscita un numero di marche superiore rispetto alla capacità dei suddetti posti.

La Fig. 5 mostra un esempio di conflitto.

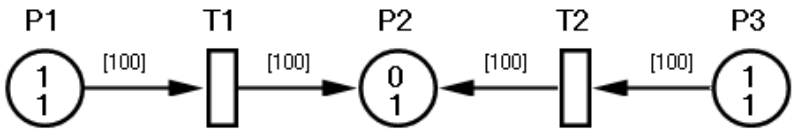


Fig. 5. Esempio di conflitto

3.4.2 MORFISMI

La teoria riguardante i morfismi è estesa e complessa. Noi ci limiteremo ai particolari tipi di morfismo che sono utilizzati in ScoreSynth: i *raffinamenti*.

I raffinamenti sono utili per descrivere modelli complessi di PNs tramite PNs semplici e strutture gerarchiche. Un raffinamento, che qui viene chiamato *sottorete*, è una PN che dà una descrizione più dettagliata di un nodo dal più alto livello di astrazione. Questo nodo viene chiamato *nodo padre* e la sottorete *rete figlia*.

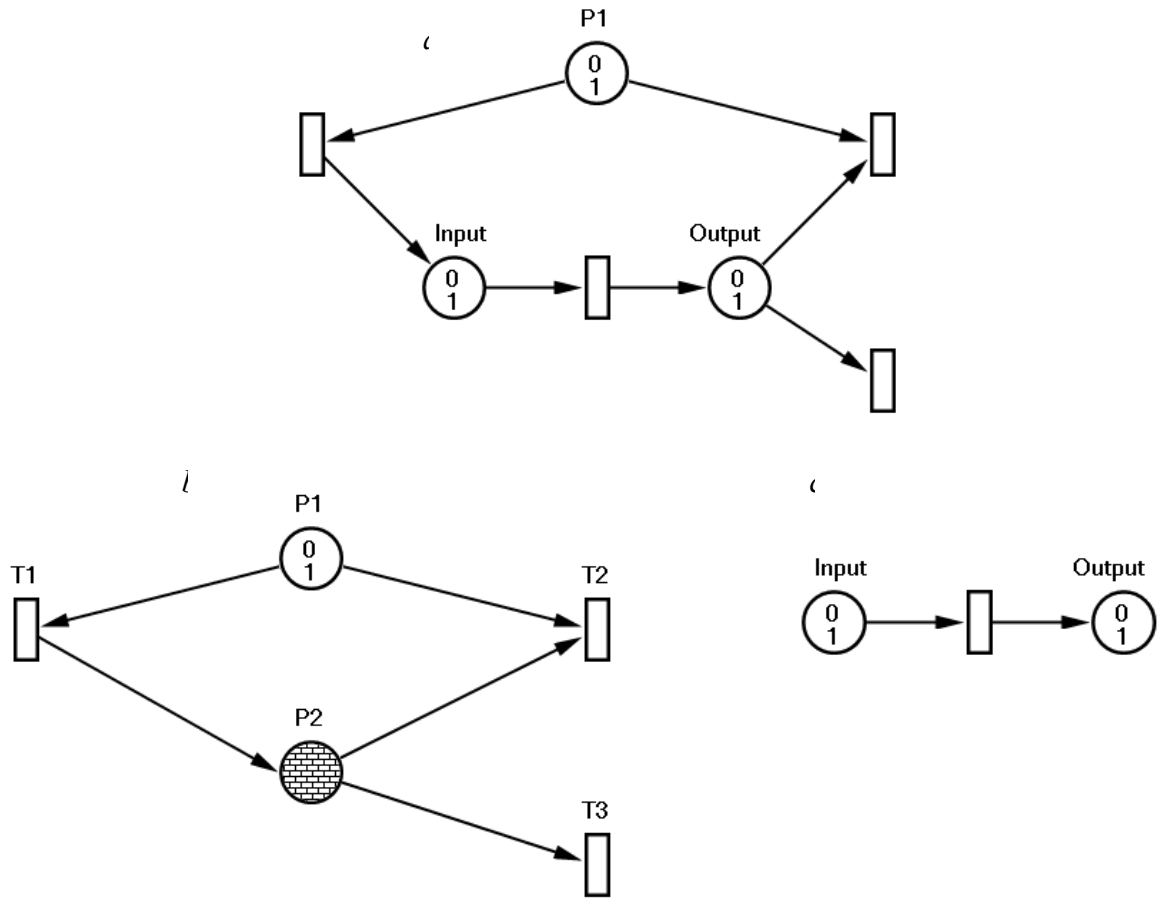


Fig. 6. Concetto di raffinamento

Se si sceglie di definire la sottorete associata ad un posto, allora la rete figlia deve avere due posti speciali: il *posto di input* e il *posto di output*; gli archi di ingresso del posto padre sono gli archi di ingresso della rete figlia; gli archi di uscita del posto padre sono gli archi di uscita della rete figlia. Lo stesso vale per le transizioni.

3.4.3 TEMPORIZZAZIONE

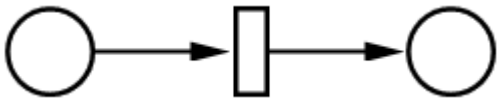
Le PNs sono particolarmente adatte per descrivere processi concorrenti e controllare la loro eventuale sincronizzazione. Quando una transizione può scattare, non si sa quando scatterà e la durata di tale scatto è supposta istantanea. Non ci sono misure di tempo all'interno delle PNs: è il comportamento della rete che implicitamente determina la temporizzazione. È la struttura della rete che determina la sequenza degli scatti.

La sequenza degli scatti può cambiare in diverse esecuzioni della rete: si possono avere più transizioni abilitate nello stesso momento e non si può sapere quale di queste scatterà per prima.

Per descrivere la durata di eventi temporizzati all'interno delle PNs è possibile associare degli intervalli o durate temporali alle transizioni o ai posti. Nel caso particolare di ScoreSynth lo scatto delle transizioni è considerato istantaneo, mentre è possibile inserire nei posti dei frammenti MX di data durata. In questo modo quando una marca è in un posto con associato un frammento MX, la marca non può essere considerata per lo scatto di transizioni connesse al posto per un tempo pari alla durata dell'oggetto.

3.5 STRUTTURE SIGNIFICATIVE DELLE PNS

Esistono alcune strutture elementari utili nella progettazione delle PNs. Esse sono: la *sequenza*, l'*alimentazione congiunta* o parallelismo o split, l'*alimentazione alternativa* o non determinismo, la *congiunzione* o asincronismo o joint e la *fusione* o sincronismo.



Sequenza

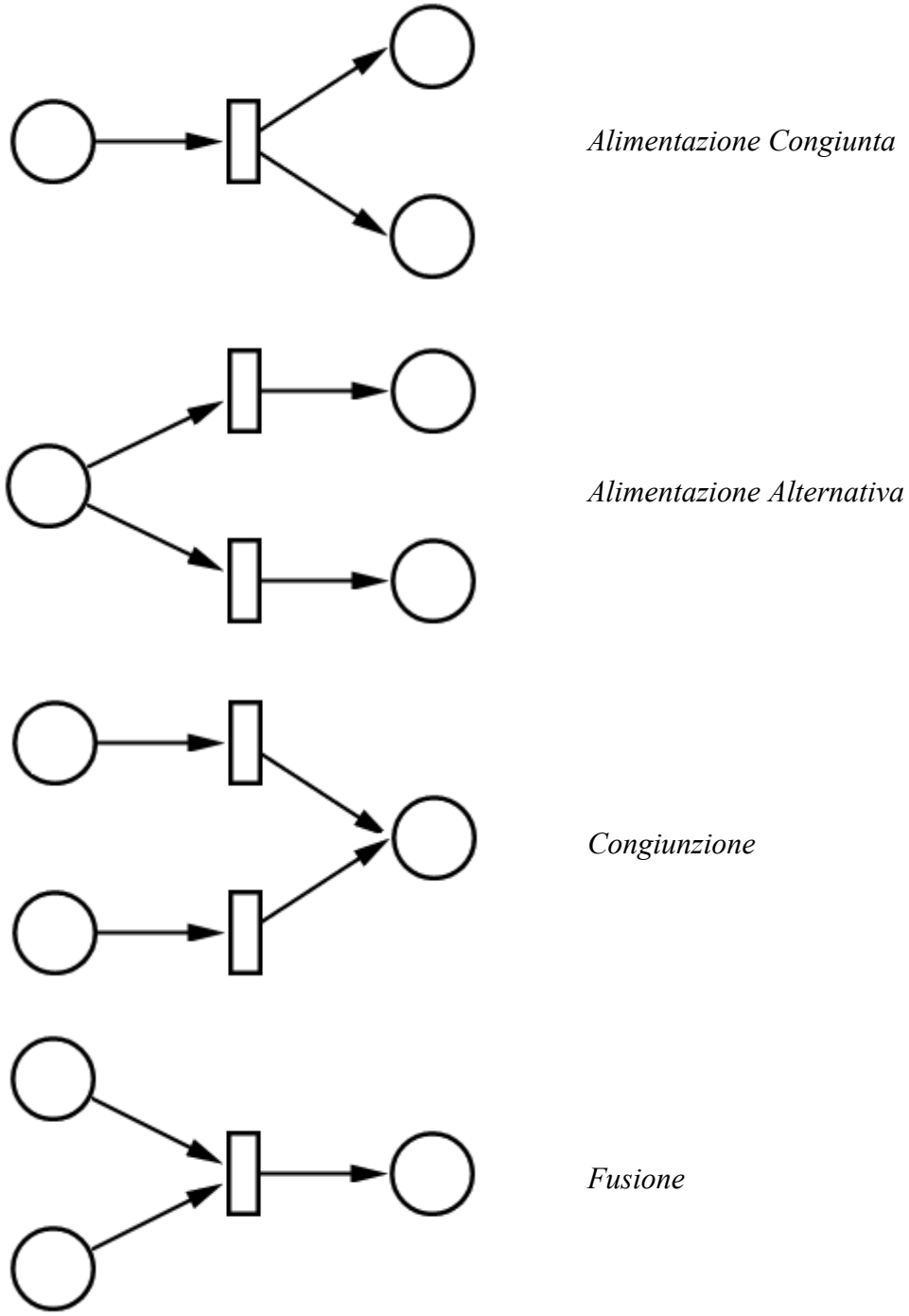


Fig. 7. Strutture elementari delle PNs

Un caso particolarmente interessante è quello dell'*alimentazione alternativa*, in quanto introduce il *non-determinismo*. In presenza di questa struttura, ScoreSynth determina la sequenza degli scatti utilizzando una funzione probabilistica: ad ogni arco è associato un peso probabilistico il cui valore relativo rispetto alla somma totale dei pesi probabilistici degli archi in conflitto indica la probabilità di scelta.

Chiariamo con un esempio questo concetto:

Si consideri una Rete di Petri con i seguenti archi:

- Arco 1: Peso probabilistico 5
- Arco 2: Peso probabilistico 10
- Arco 3: Peso probabilistico 300

Nel caso in cui l'applicazione debba effettuare un scelta non deterministica tra gli archi 1, 2 e 3, l'arco 1 avrà 5 probabilità su 315 di essere scelto (1,6%), l'arco 2: 10 su 315 (3,2%), l'arco 3: 300 su 315 (95,2%).

Se in un passaggio successivo dell'esecuzione gli archi che indicano le transizioni in conflitto sono soltanto l'1 e il 2, le probabilità cambiano dinamicamente e diventano: arco 1: 5 su 15 (33,3%) e arco 2: 10 su 15 (66,7%).

Tutte le strutture elementari considerate possono essere collegate per formare delle PNs più complesse.

Leggermente più complicata è la struttura *loop*, usata per ripetere un determinato numero di volte una sottostruttura. In Fig. 8 ne sono mostrate due possibili implementazioni.

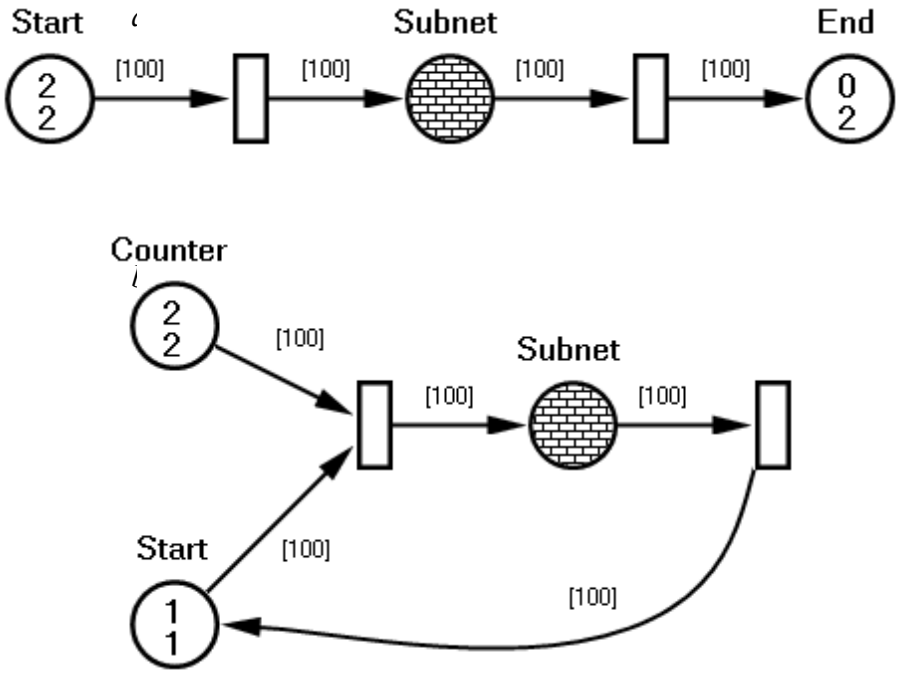


Fig. 8. Due implementazioni della struttura *loop*

CAPITOLO 4

MX

4.1 INTRODUZIONE

I vantaggi offerti dal formato XML sono molti, e la sua diffusione avvenuta negli ultimi anni in tutti i campi dell'informatica lo dimostra.

Una trattazione approfondita di questa tecnologia esula comunque dallo scopo di questo lavoro e si lascia al lettore l'eventuale lettura di materiale introduttivo, sicuramente di facile reperibilità.

Non è la prima volta che un formato di rappresentazione di informazione musicale viene introdotto con lo scopo di diventare uno standard di interscambio, ma non c'è ancora stato un progetto che sia riuscito a mantenere la promessa. Due dei problemi principali di questi tentativi erano la grande specializzazione del singolo formato, limitato ad un contesto specifico (audio, notazionale, …) e la relativa difficoltà di produzione di tools che ne consentivano il trattamento.

Come più volte sottolineato in diversi incontri, non ultimo il MAX2002 [MAX2002], occorreva trovare un modo per rappresentare i fenomeni musicali a vari livelli, e creare uno standard per l'interscambio di questi fenomeni musicali fra applicazioni diverse. L'utilizzo dell'XML consente una facile implementazione di tools, e il formato MX è stato studiato proprio con l'intento di unificare in un unico documento informazioni musicali a livello simbolico, audio, notazionale, ecc...

La facilità di implementazione di applicazioni che trattino files XML deriva dal fatto che esso è un formato non specializzato, e questo ad esempio permette di disporre, all'interno del framework .NET, di classi specializzate nel suo utilizzo generale.

Questo significa che al programmatore viene lasciato l'onere di dare un senso alle informazioni contenute in un file XML di un certo tipo, mentre il framework implementa nativamente le funzionalità generali e rende disponibile una rappresentazione in memoria

della struttura ad albero del file, delle informazioni contenute e una serie di metodi per trattarli.

4.2 I LIVELLI DEL FORMATO MX

Come detto, uno delle caratteristiche principali del formato MX è la sua organizzazione strutturale a livelli (layers), che consente diversi gradi di astrazione dell'informazione musicale. I livelli sono visibili in Fig. 9.

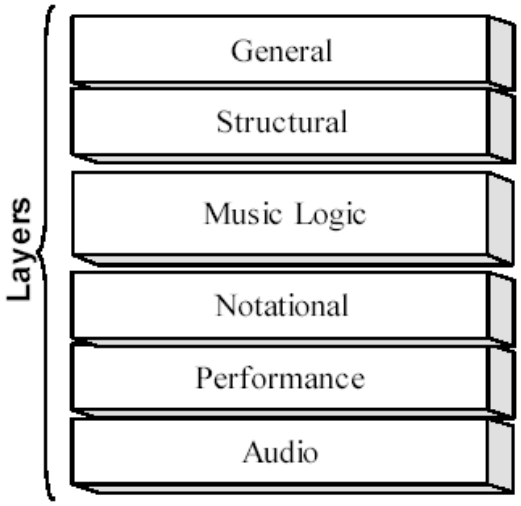


Fig. 9. Livelli del formato MX

Vediamo nel dettaglio quali informazioni sono contenute in ogni livello MX.

General. Nel livello generale vengono codificate le informazioni relative al fenomeno musicale rappresentato, quali: autore e titolo, data e luogo della performance, informazioni sulla registrazione, il mixaggio, note generali, ecc...

Structural. Il livello strutturale ha il compito di ospitare la descrizione degli oggetti musicali che costituiscono il fenomeno descritto e le loro relazioni, sia da un punto di vista compositivo che musicologico. Le PNs costituiscono un possibile approccio al problema, anche se questo livello è il più aperto a future discussioni.

Music Logic. Il più complesso dei livelli MX, e anche il più ricco di informazioni. Nel livello logico è contenuta la struttura *spine* (descritta più avanti), la descrizione simbolica del fenomeno musicale a livello di parti, righi, misure, note, testi, ecc..., ed alcune informazioni di layout, che specificano come rappresentare graficamente le informazioni simboliche di questo livello.

Notational. In questo livello trova posto la rappresentazione notazionale visuale dell'informazione musicale; possono quindi essere presenti riferimenti a immagini della partitura, a files NIFF o ENIGMA, che descrivono la partitura graficamente, oppure altri tipi di rappresentazioni.

Performance. Contiene riferimenti a files che descrivono simbolicamente il modo di produrre output musicale, come ad esempio files MIDI e CSound.

Audio. Nell'ultimo livello vengono ospitati i riferimenti a files contenenti materiale audio, come wave e mp3.

4.3 TAGS PROCESSATI IN SCORESYNTH

In ScoreSynth naturalmente non vengono modificati tutti i tags MX, visto che gran parte di essi devono essere semplicemente ricopiati dai files di input a quelli di output[2].

Tuttavia un certo numero di elementi MX ha un'importanza fondamentale per il funzionamento del software e necessita di un approfondimento opportuno.

4.3.1 LA SPINE

La struttura chiamata *spine* fa parte del livello logico del formato MX e viene utilizzata per sincronizzare tra loro gli eventi occorrenti nel fenomeno musicale descritto dal documento MX (v. Fig. 10).

La spine è quindi una lista di eventi. Ogni evento della spine ha associato tre attributi: il nome, la coordinata temporale relativa e la coordinata spaziale relativa. Si parla di relatività perché gli attributi temporale e spaziale dell'evento vengono espressi rispetto all'evento precedente della spine. L'unità temporale è il *vtu (Virtual Time Unit)*, una coordinata virtuale mutuata dal formato SMDL [New95], mentre l'unità spaziale è il *vpx (Virtual PiXel)*.

[2] In effetti ScoreSynth modifica gran parte dei tag MX, ma solo aggiungendo agli attributi `id` il prefisso che indica il numero del file MX processato. Il resto del contenuto di questi elementi viene copiato senza modifiche.

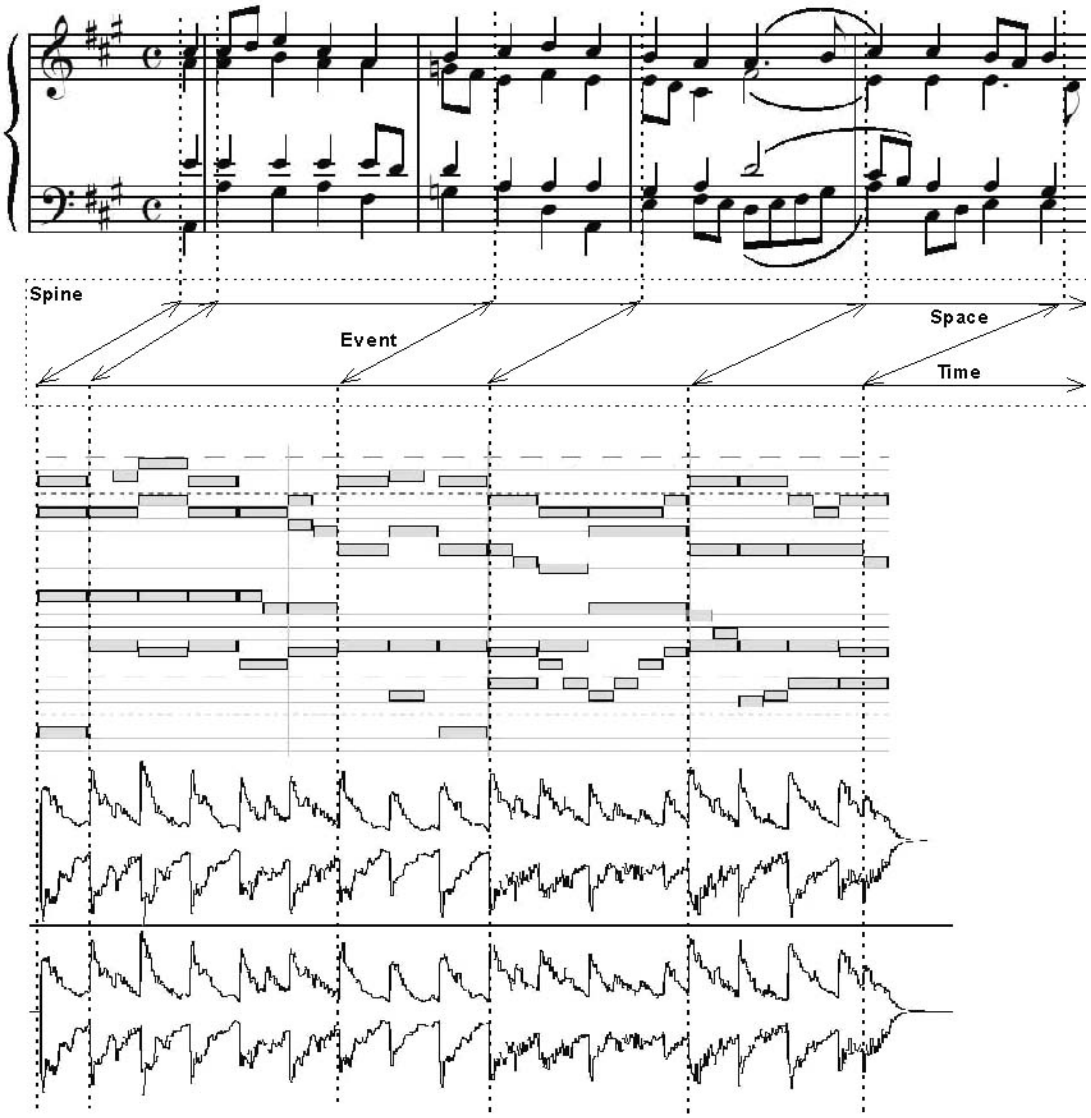


Fig. 10. La spine e i livelli notazionale, audio e di performance

Per consentire un'attualizzazione dei vpx è disponibile (anche se non obbligatorio) l'elemento `layout` del livello logico del formato MX, che presenta gli attributi `h_pos_division` e `standard_h_pos_unit`. I vpx vengono quindi definiti come numero delle divisioni (`h_pos_division`) per unità spaziale assoluta (`standard_h_pos_unit`).

Nelle prime versioni del formato non era previsto invece un elemento specifico per attualizzare i vtu, e occorreva eventualmente ricavare un fattore di attualizzazione analizzando i livelli che puntavano alla spine.

Per generalizzare il procedimento di giustapposizione/sovrapposizione di documenti MX consentito da ScoreSynth, si è introdotto nella versione 1.3.1b l'attributo `vtu_amount` all'elemento `logic/los/staff_list/staff/time_signature`, che specifica ad ogni cambio di segnatura di tempo in quanti vtus è divisa una battuta[3].

La struttura spine è rappresentata nell'elemento `mx/logic/spine`, che contiene una lista di elementi `event`.

Quando vengono processati i files MX associati ai posti della PN la parte più importante dell'algoritmo relativo è proprio il mixaggio degli eventi delle spines. Proprio la presenza di questa struttura infatti consente di mantenere inalterata gran parte del documento MX da inserire in output, che è costituito solo da puntatori che quindi non devono subire modifiche. In questa versione di ScoreSynth i vpx vengono lasciati inalterati, in quanto ulteriori modifiche a questo strumento sono in fase di discussione per future releases del DTD MX.

Vediamo ora nello specifico come vengono miscelate le spines: quando viene prodotto il documento di output dall'esecuzione della PN, i frammenti MX associati ai posti vengono sovrapposti/giustapposti in relazione al loro inizio temporale ed alla loro durata. A questo scopo nella rete esiste una variabile globale che indica il tempo trascorso e che viene aggiornata durante l'esecuzione. Quando una marca arriva in un posto della PN con un oggetto musicale associato (oppure proveniente dalla transizione entrante) il frammento viene inserito nel documento di output, modificandolo preventivamente.

Le modifiche che vengono apportate ai frammenti da inserire sono essenzialmente due:

- viene aggiunto un prefisso **mx#_** (dove # è il numero progressivo dei posti processati) a tutti gli ID degli elementi XML, con la duplice funzione di mantenere l'unicità dell'attributo e di consentire una rapida individuazione del frammento di provenienza di ogni evento.
- gli attributi **timing** degli eventi della spine vengono modificati per sovrapporre/giustapporre il frammento. Se il frammento è da inserire semplicemente alla fine dell'output corrente, il suo primo evento verrà modificato (il primo **timing** vale 0), specificando la distanza temporale dall'ultimo evento della spine di output. Se il frammento è in parziale o totale sovrapposizione temporale con l'output corrente, invece, inizialmente viene trovata la posizione nella spine di output in cui deve essere inserito il primo evento del frammento, poi le due spine vengono fatte scorrere parallelamente e gli eventi vengono interfogliati.

[3] Per un approfondimento sull'attualizzazione dei vtu si veda il capitolo sulla polimetria.

La parte rimanente del file MX da inserire in output viene semplicemente accodato agli elementi esistenti. Questo significa che il numero di elementi **staff** del file di output, ad esempio, sarà uguale alla somma di tutti gli elementi dello stesso tipo di tutti i files MX eseguiti.

La modifica degli attributi **timing** degli eventi delle spines da sovrapporre/giustapporre è un punto fondamentale: per questo forniamo una visualizzazione del procedimento seguito per maggiore chiarezza.

- 1° Caso: Frammenti da giustapporre. In questo caso la spine del frammento da inserire deve essere semplicemente accodata all'output. Supponendo che le due spine da giustapporre siano quelle presentate sotto e che il tempo d'esecuzione corrente sia uguale a 25 risulterebbe in output la spine a destra, con il **timing** dell'evento **mx1_ev1** aggiornato.

Spine di output iniziale	
ID evento	*timing*
mx0_ev1	0
mx0_ev2	10
mx0_ev3	3
mx0_ev4	10

Frammento MX	
ID evento	*timing*
mx1_ev1	0
mx1_ev2	5
mx1_ev3	6
mx1_ev4	2

Spine di output finale	
ID evento	*timing*
mx0_ev1	0
mx0_ev2	10
mx0_ev3	3
mx0_ev4	17
mx1_ev1	2
mx1_ev2	5
mx1_ev3	6
mx1_ev4	2

- 2° Caso: Frammenti da sovrapporre. In questo caso il primo evento della spine del frammento da sovrapporre viene inserito al tempo corrente e tutti gli altri eventi devono essere interfogliati, cambiando i loro parametri **timing**. Supponendo che la spine dell'esempio precedente debba essere inserita al tempo 12, risulterebbe:

Spine di output iniziale	
ID evento	*timing*
mx0_ev1	0
mx0_ev2	10
mx0_ev3	3
mx0_ev4	10

Frammento MX	
ID evento	*timing*
mx1_ev1	0
mx1_ev2	5
mx1_ev3	6
mx1_ev4	2

Spine di output finale	
ID evento	*timing*
mx0_ev1	0
mx0_ev2	10
mx1_ev1	2
mx0_ev3	1
mx1_ev2	4
mx1_ev3	6
mx0_ev4	0
mx1_ev4	2

4.3.2 NOTE E PAUSE

Altri elementi del DTD MX che vengono processati sono quelli che codificano le note e le pause a livello logico, ovvero `logic/los/part/measure/voice/chord/notehead`, e `logic/los/part/measure/voice/rest`.

Questi elementi vengono usati prima di tutto nel metodo `Duration` della classe `Place`, visto che la durata dell'ultimo evento della spine non può essere dedotto dalla stessa ma deve essere ricavato "manualmente" consultando il corrispondente elemento `notehead` o `rest`.

4.4 DTD MX VERS. 1.3.1B (APRILE 2004)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
************************************************************************
 File name:    MX.dtd
 Version:      1.3
 Creation date:     03/09/2003
 Last modification: 17/04/2004

 Description:
 This format definition is under development at
 Laboratorio di Informatica Musicale (LIM).
 For further information please refer to the documentation at ....
-->
<!--
************************************************************************
      Import of SVG Elements
************************************************************************
-->
<!ENTITY % svg PUBLIC
   "//MAX"
   "svg.dtd">
%svg;
<!--
************************************************************************
      Import of MP3_XML Elements
************************************************************************
```

```
-->
<!ENTITY % mp3_xml PUBLIC
   "//MAX"
   "mp3_xml.dtd">
%mp3_xml;
<!--
*************************************************************************
      Import of XRML Elements
*************************************************************************
-->
<!--
ENTITY % XrML PUBLIC
   "//MAX"
   "xrml2core.xsd">
%XrML;
-->
<!--
*************************************************************************
      Common Attributes Parameter entities
*************************************************************************
-->
<!-- Spine reference attributes -->
<!ENTITY % spine_attributes "event_ref IDREF #REQUIRED">
<!ENTITY % placement_def "v_placement (above | center | below) 'above'
               h_placement (left | center | rigth) 'center' ">
<!ENTITY % accattr "(none|sharp|flat|natural|doublesharp|doubleflat)">
<!--
*************************************************************************
      Common Elements
*************************************************************************
-->
<!ELEMENT rights (#PCDATA)>
<!--
work to be done is to study in which way we can use the XRML specification
in this context
-->
<!ELEMENT chord_ref EMPTY>
<!ATTLIST chord_ref
   anchor IDREF #REQUIRED
>
<!--
```

```
*************************************************************************
*************************************************************************
                Root Element
*************************************************************************
*************************************************************************
-->
<!ELEMENT mx (general, structural?, logic, notational?, performance?,
audio?)>
<!ATTLIST mx
   creator CDATA #IMPLIED
>
<!ELEMENT logic (spine, los, layout?)>
<!--
*************************************************************************
*************************************************************************
   General Layer
*************************************************************************
*************************************************************************
-->
<!ELEMENT general (description, casting?, related_files?, analog_media?,
notes?, rights?)>
<!ELEMENT description (work_title?, work_number?, movement_title,
movement_number?, genre?, author+)>
<!ELEMENT genre (#PCDATA)>
<!ATTLIST author
   type CDATA #IMPLIED
>
<!ELEMENT author (#PCDATA)>
<!ELEMENT work_title (#PCDATA)>
<!ELEMENT work_number (#PCDATA)>
<!ELEMENT movement_title (#PCDATA)>
<!ELEMENT movement_number (#PCDATA)>
<!-- description of the music event (genre, date, place); -->
<!ELEMENT casting EMPTY>
<!-- casting information;-->
<!ELEMENT related_files EMPTY>
<!-- the table of related music data files, referring to all layers, with
one or more files for the summarization of each layer;-->
<!-- the table of related multimedia data files, such as images, videos,
and the like;-->
<!ELEMENT analog_media EMPTY>
```

```
<!-- the table of related analog media;-->
<!-- technical  information about related
import/export/restoring/cataloguing/other operations;-->
<!ELEMENT notes EMPTY>
<!-- general notes.-->
<!--
**************************************************************************
**************************************************************************
  Structural Layer
**************************************************************************
**************************************************************************
-->
<!ELEMENT structural (analysis*, CPN*)>
<!--
**************************************************************************
      Melodic themes
**************************************************************************
-->
<!ELEMENT analysis (theme*, segment*, transformation*, relationship*)>
<!ATTLIST theme
   id ID #IMPLIED
   ordinal CDATA #IMPLIED
   desc CDATA #IMPLIED
>
<!--
Desc attribute provides a textual description of the theme. Ordinal
attribute describes the possible numeric characterization of the theme.
 It should be encoded in roman numbers: I, II, III, IV, V, etc (e.g. I and
II themes in a Sonata.)
-->
<!ELEMENT theme (occurrence)>
<!ELEMENT occurrence (thm_desc?, thm_spine_ref+, (transposition | inversion
| retrogradation)*)>
<!ELEMENT thm_spine_ref EMPTY>
<!--
This element is needed for generalization of theme representation, since
there could be themes split in different sequences of notes belonging to
the same part or even to different parts.
-->
<!ATTLIST thm_spine_ref
   spine_start_ref IDREF #REQUIRED
```

```
   spine_end_ref IDREF #REQUIRED
   part_ref IDREF #REQUIRED
   voice_ref IDREF #REQUIRED
>
<!ELEMENT thm_desc (#PCDATA)>
<!ELEMENT transposition EMPTY>
<!--
Interval is an integer number, indicates the interval of transposition and
its iterpretation is related to the type attribute.
When type is real interval indicates the distance in semitones, otherwise
it indicates distance in tonal scale.
-->
<!ATTLIST transposition
   type (real | tonal) #REQUIRED
   interval CDATA #REQUIRED
>
<!ELEMENT inversion EMPTY>
<!--
Staffstep attribute must have the same interpretation as the staff_step
attribute of noteheads.
-->
<!ATTLIST inversion
   type (real | tonal) #REQUIRED
   staff_step CDATA #REQUIRED
>
<!ELEMENT retrogradation EMPTY>
<!--
************************************************************************
      Coloured Petri Nets
************************************************************************
-->
<!ELEMENT CPN EMPTY>
<!--
************************************************************************
      Segments
************************************************************************
-->
<!ATTLIST relationship
   id ID #REQUIRED
   segmentAref IDREF #REQUIRED
   segmentBref IDREF #REQUIRED
```

```
   transformationref IDREF #REQUIRED
>
<!ELEMENT relationship EMPTY>
<!ATTLIST segment
   id ID #REQUIRED
>
<!ELEMENT segment (segment_event+)>
<!ELEMENT transformation EMPTY>
<!ATTLIST transformation
   id ID #REQUIRED
   description CDATA #REQUIRED
   gis CDATA #IMPLIED
>
<!ATTLIST segment_event
   id_ref IDREF #REQUIRED
>
<!--
************************************************************************
************************************************************************
   Music Logic Layer
************************************************************************
************************************************************************
-->
<!--
************************************************************************
   Spine (space-time structure)
************************************************************************
-->
<!ELEMENT segment_event EMPTY>
<!ELEMENT spine (event)*>
<!ATTLIST spine
   id CDATA #IMPLIED
>
<!ELEMENT event EMPTY>
<!--
The event element can represent both notes and rests.
Timing is expressed by VTU (Virtual Timing Units), a coordinate (like MIDI
ticks) relative to the previous event hpos (horizontal position) is
expressed  in VPX relative to the previous event.
Default values are NULL. An event having timing = NULL is a space event,
```

```
an event having hpos = NULL is a time event. An event with timing and hpos
= NULL is a wrong one.
 -->
<!ATTLIST event
   id ID #REQUIRED
   timing CDATA "NULL"
   hpos CDATA "NULL"
>
<!--
*************************************************************************
   Logical Organized Symbols
*************************************************************************
-->
<!ELEMENT los (agogics*, text_field*, metronomic_indication*, staff_list,
part+, horizontal_symbols, lyric*)>
<!ELEMENT agogics (#PCDATA)>
<!ATTLIST agogics
   font_type CDATA #IMPLIED
   font_size CDATA #IMPLIED
   %spine_attributes;
>
<!--agogics represents the first agogic indication of the piece-->
<!ELEMENT text_field (#PCDATA)>
<!ATTLIST text_field
   font_type CDATA #IMPLIED
   font_size CDATA #IMPLIED
   %spine_attributes;
>
<!--
a text field is provided for generic textual indications. For agogics, the
proper tag should be used; the same for dynamics
-->
<!ELEMENT metronomic_indication EMPTY>
<!ATTLIST metronomic_indication
   num CDATA #REQUIRED
   den CDATA #REQUIRED
   value CDATA #REQUIRED
   %spine_attributes;
>
<!--
```

```
Examples of metronomic_indication: half = 60 -> num = 1, den = 2, value =
60; dotted quarter = 88 -> num = 3, den = 8, value = 88
-->
<!ELEMENT part (voice_list, measure+)>
<!--
dfstaff_ref must be dafaulted in the file loading phase, staff changes wil
be coded in those symbols for which it is necessary
-->
<!ATTLIST part
   id ID #REQUIRED
   dfstaff_ref IDREF #REQUIRED
>
<!ELEMENT voice_list (voice_item+)>
<!ELEMENT voice_item EMPTY>
<!ATTLIST voice_item
   id ID #REQUIRED
>
<!ELEMENT measure (measure_repeat? | voice+)>
<!ATTLIST measure
   id ID #IMPLIED
   number CDATA #REQUIRED
>
<!ELEMENT measure_repeat EMPTY>
<!ATTLIST measure_repeat
   type (single | firsthalf | secondhalf) #REQUIRED
   numeral CDATA #IMPLIED
   %spine_attributes;
>
<!ELEMENT voice (chord | rest)+>
<!--
ref attribute refers to voice_items in the voice_list. They are defined
externally because a voice spans over several measures, ossia voices can
have the id of a  regular voice.
-->
<!ATTLIST voice
   ref IDREF #REQUIRED
   ossia (yes | no) "no"
>
<!ELEMENT rest (duration, aug_dot?)>
<!ATTLIST rest
   id CDATA #IMPLIED
```

```
   staff_ref IDREF #REQUIRED
   %spine_attributes;
   breath_mark (normal | comma | caesura) "normal"
>
<!ELEMENT chord (notehead+, articulation?, arpeggio?)>
<!ATTLIST chord
   id ID #IMPLIED
   %spine_attributes;
   stem_direction (up | down | none) #IMPLIED
   stem_length CDATA #IMPLIED
   stem_share_chord_ref IDREF #IMPLIED
   cue (yes | no) "no"
   grace (yes | no) "no"
>
<!ELEMENT articulation (staccatissimo | staccato | tenuto | marcato |
accento)*>
<!ELEMENT accento EMPTY>
<!ATTLIST accento
   type (strong | medium) #REQUIRED
   %placement_def;
>
<!ELEMENT staccatissimo EMPTY>
<!ELEMENT staccato EMPTY>
<!ELEMENT tenuto EMPTY>
<!ELEMENT marcato EMPTY>
<!ELEMENT arpeggio (note_head_event_ref+)>
<!ATTLIST arpeggio
   shape (wavy | vertical_slur) #REQUIRED
>
<!ELEMENT note_head_event_ref EMPTY>
<!ATTLIST note_head_event_ref
   note_ref IDREF #REQUIRED
   %spine_attributes;
>
<!ELEMENT notehead (pitch_def, duration, accidental?, aug_dot?, tie?,
fingering?)>
<!ATTLIST notehead
   id ID #IMPLIED
   staff_ref IDREF #REQUIRED
>
<!ELEMENT pitch_def EMPTY>
```

```
<!ATTLIST pitch_def
   staff_step CDATA #REQUIRED
>
<!-- NOTE: In this version the duration refers only to the notehead shape -
->
<!ELEMENT duration EMPTY>
<!ATTLIST duration
   num CDATA #REQUIRED
   den CDATA #REQUIRED
>
<!ELEMENT accidental ((sharp)* | (flat)* | (natural))>
<!ATTLIST accidental
   size (normal | small) "normal"
>
<!ELEMENT sharp EMPTY>
<!ATTLIST sharp
   parenthesis (yes | no) "no"
   h_offset CDATA "0"
>
<!ELEMENT flat EMPTY>
<!ATTLIST flat
   parenthesis (yes | no) "no"
   h_offset CDATA "0"
>
<!ELEMENT natural EMPTY>
<!ATTLIST natural
   parenthesis (yes | no) "no"
   h_offset CDATA "0"
>
<!ELEMENT aug_dot EMPTY>
<!ATTLIST aug_dot
   number CDATA "1"
   %placement_def;
>
<!ELEMENT tie EMPTY>
<!--
tie is used to represent simple and single tie; it is a property of
notehead, and doesn't require a reference to an end symbol, as the latter
is implicit (it is the first notehead with the same pitch in the sequence
and in that voice); for more complex types of tie, <generic_tie> in
<horizontal_symbols> can be employed.
```

```
-->
<!ELEMENT fingering EMPTY>
<!ATTLIST fingering
   number (1 | 2 | 3 | 4 | 5) #REQUIRED
   %placement_def;
>
<!--
*********************************************************************
   Lyrics:
   Lyrics are conceived as another logical unit subdivided into syllables.
   Each lyric is related to a voice in a part.
   Each syllable is related to the relative note(s) in the voice or to start
   and end events in the spine structure.
   The visualization of lyrics are handled by the lyric_pieces that are
   elements children of staff_piece in the Layout domain.
-->
<!ELEMENT lyric (syllable+)>
<!ATTLIST lyric
   font_type CDATA #REQUIRED
   font_size CDATA #REQUIRED
   part_ref IDREF #REQUIRED
   voice_ref IDREF #REQUIRED
>
<!ELEMENT syllable (sbtext, (chord_ref*))>
<!--
References to chords (chord_ref) are useful for classic lyric notation
where
each syllable is related to the or more notes.
References to events (start_event_ref, end_event_ref) are useful when the
text is not related to a particular melody (voice) but must be however
synchronized
(e.g. a karaoke piece).
v_offset indicates the offset from the position defined in the relative
lyricpiece.
-->
<!ATTLIST syllable
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
   hyphen (yes | no) "no"
   v_offset CDATA "0"
>
```

```
<!ELEMENT sbtext (#PCDATA)>
<!--
*********************************************************************
   Horizontal Staff Symbols: all the symbols that spans over more events
  and or voices
-->
<!ELEMENT horizontal_symbols (beam* | hairpin* | dynamic* | tuplet* | slur*
| generic_tie* | glissando* | ornament? | chord_symbol* |
projection_symbols?)>
<!ELEMENT beam (chord_ref+)>
<!ATTLIST beam
   id CDATA #IMPLIED
   fanned (none | expanding | shrinking) "none"
>
<!ELEMENT hairpin EMPTY>
<!ATTLIST hairpin
   id CDATA #IMPLIED
   type (crescendo | diminuendo) #REQUIRED
   staff_ref IDREF #REQUIRED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
>
<!ELEMENT dynamic (#PCDATA)>
<!ATTLIST dynamic
   id CDATA #IMPLIED
   type (pppp | ppp | pp | p | mp | mf | f | ff | fff | ffff | sp | sf | sfz
| fz | fp | crescendo | diminuendo) #REQUIRED
   staff_ref IDREF #REQUIRED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #IMPLIED
>
<!ELEMENT tuplet (tuplet*)>
<!ATTLIST tuplet
   id ID #IMPLIED
   part_ref IDREF #IMPLIED
   voice_ref IDREF #IMPLIED
   type (none | round | bracket) "none"
   start_chord_ref IDREF #REQUIRED
   end_chord_ref IDREF #REQUIRED
   num CDATA #REQUIRED
   den CDATA #REQUIRED
```

```
   v_placement (bottom | center | above) "center"
   visualizeden (yes | no) "no"
>
<!ELEMENT slur EMPTY>
<!ATTLIST slur
   start_chord_ref IDREF #REQUIRED
   end_chord_ref IDREF #REQUIRED
   shape_ref IDREF #REQUIRED
>
<!ELEMENT generic_tie (path)>
<!ATTLIST generic_tie
   start_chord_ref IDREF #REQUIRED
   end_chord_ref IDREF #REQUIRED
>
<!ELEMENT ornament (trill | tremolo | mordente | turn | coulee)*>
<!ELEMENT trill EMPTY>
<!ATTLIST trill
   start_chord_ref IDREF #REQUIRED
   end_chord_ref IDREF #REQUIRED
   accidental %accattr; "none"
   hook (none | up | down) "none"
>
<!ELEMENT tremolo EMPTY>
<!--
If start_chord_ref and end_chord_ref have the same value the element refer
to the "single note tremolo"
i.e. the symbol defined only on none note).
-->
<!ATTLIST tremolo
   start_chord_ref IDREF #REQUIRED
   end_chord_ref IDREF #REQUIRED
   line_num CDATA #REQUIRED
>
<!ELEMENT mordente EMPTY>
<!ATTLIST mordente
   chord_ref IDREF #REQUIRED
   type (short | double) #REQUIRED
   inverted (yes | no) "no"
   hook (none | startup | startdown | finalup | finaldown) "none"
>
<!ELEMENT turn EMPTY>
```

```
<!ATTLIST turn
   chord_ref IDREF #REQUIRED
   type (over | after) #REQUIRED
   inverted (yes | no) "no"
   upacc %accattr; "none"
   downacc %accattr; "none"
>
<!ELEMENT coulee EMPTY>
<!ATTLIST coulee
   chord_ref IDREF #REQUIRED
>
<!ELEMENT chord_symbol EMPTY>
<!ATTLIST chord_symbol
   id CDATA #IMPLIED
   event_ref IDREF #REQUIRED
>
<!ELEMENT glissando EMPTY>
<!-- NULL stands for undefined note -->
<!ATTLIST glissando
   start_notehead IDREF #REQUIRED
   end_notehead IDREF #REQUIRED
>
<!--
**********************************************************************
   Projection Score Symbols (i.e. valid for all the staffs)
-->
<!ELEMENT projection_symbols (repeat | coda | segno | fine |
multiple_ending)*>
<!--
Repeat is a general structure for the representation of repetition in the
score
   Examples for the repeat signs:
   D.C. al fine
   <repeat event_ref="(ID of the event relative to the -D.C. al fine-
sign)">
      <text>D.C. al fine</text>
      <jump event_ref="NULL"/> // NULL idicates the begin of the score
      <follow_to event_ref="(ID of the event relative to the Fine sign)"/>
   </repeat>

   D.C. al Segno e poi al Coda
```

```
  <repeat "(ID of the event relative to the -D.C. al Segno e poi al Coda-
sign)">
     <text>D.C. al Segno e poi al Coda</text>
     <jump event_ref="NULL"/>
     <follow_to event_ref="(ID of the event relative to the Segno sign)"/>
     <jump event_ref="(ID of the event relative to the Coda sign)"/>
  </repeat>
-->
<!ELEMENT repeat (symbol_text?, (jump, follow_to?)*)>
<!ATTLIST repeat
  id ID #IMPLIED
  %spine_attributes;
>
<!ELEMENT symbol_text EMPTY>
<!ELEMENT jump EMPTY>
<!ATTLIST jump
  id ID #IMPLIED
  %spine_attributes;
>
<!ELEMENT follow_to EMPTY>
<!ATTLIST follow_to
  id CDATA #IMPLIED
  %spine_attributes;
>
<!ELEMENT segno (#PCDATA)>
<!ATTLIST segno
  id ID #IMPLIED
  %spine_attributes;
>
<!ELEMENT coda (#PCDATA)>
<!ATTLIST coda
  id ID #IMPLIED
  %spine_attributes;
>
<!ELEMENT fine (#PCDATA)>
<!ATTLIST fine
  id ID #IMPLIED
  %spine_attributes;
>
<!ELEMENT multiple_ending (#PCDATA)>
<!ATTLIST multiple_ending
```

```
   id ID #IMPLIED
   start_event_marker_ref IDREF #REQUIRED
   end_event_marker_ref IDREF #IMPLIED
>
<!--
**************************************************************************
   Staff System
-->
<!ELEMENT staff_list (staff*)>
<!ELEMENT staff (clef | key_signature | time_signature | barline |
staff_hiding)*>
<!ATTLIST staff
   id ID #IMPLIED
   ossia (yes | no) "no"
   line_number CDATA "5"
>
<!ELEMENT clef EMPTY>
<!ATTLIST clef
   type (G | F | C | percussion | doubleG | tabguitar) #REQUIRED
   staff_step CDATA #REQUIRED
   octave_num (0 | 8 | -8 | 15 | -15) "0"
   %spine_attributes;
>
<!ELEMENT key_signature ((sharp_num | flat_num), natural_num?)?>
<!ATTLIST key_signature
   %spine_attributes;
>
<!ELEMENT sharp_num EMPTY>
<!ATTLIST sharp_num
   number (1 | 2 | 3 | 4 | 5 | 6 | 7) #REQUIRED
>
<!ELEMENT flat_num EMPTY>
<!ATTLIST flat_num
   number (1 | 2 | 3 | 4 | 5 | 6 | 7) #REQUIRED
>
<!ELEMENT natural_num EMPTY>
<!ATTLIST natural_num
   number (1 | 2 | 3 | 4 | 5 | 6 | 7) #REQUIRED
>
<!ELEMENT time_signature EMPTY>
<!--
```

```
The common non-numeric time signature are indicated by the char_type
attribute.
The cut attribute allows to represent cut time notation (such as C cut for
2/2)
For a single number time-signature, use the single_number.
To place an ordinary time-signature above the staff, use the v_placement
tag, vertical component = above. To place it between the staves, use the
v_placement tag, vertical component = below.
-->
<!ATTLIST time_signature
   num CDATA #REQUIRED
   den CDATA #REQUIRED
   char_type (yes | no) "no"
   cut (yes | no) "no"
   single_number (yes | no) "no"
   v_placement (above | center | below) "center"
   size (normal | large | small) "normal"
   visible (yes | no) "yes"
   vtu_amount CDATA #IMPLIED
   %spine_attributes;
>
<!--
vtu_amount represents the total amount of virtual timing units of the
measure; that is the sum of virtual timing units of the single sequencial
events of the measure. VTU is a concept related to the spine layer.
 -->
<!-- barlines -->
<!ELEMENT barline (repetition | final | double)?>
<!ATTLIST barline
   id ID #IMPLIED
   %spine_attributes;
>
<!ELEMENT repetition EMPTY>
<!ATTLIST repetition
   direction (left | right | both) #REQUIRED
>
<!ELEMENT final EMPTY>
<!ATTLIST final
   broken (yes | no) "no"
>
<!-- used for key signature changing -->
```

```
<!ELEMENT double EMPTY>
<!ELEMENT staff_hiding EMPTY>
<!--
This element is used for ossia and partial hiding of staff
start_eventmarker, end_eventmarker refer to mHiding markers
-->
<!ATTLIST staff_hiding
   id ID #IMPLIED
   start_event CDATA #REQUIRED
   end_event CDATA #REQUIRED
>
<!--
Other elements to add (take examples from NIFF):
- part description override
...
-->
<!--
*************************************************************************
   Layout
*************************************************************************
-->
<!ELEMENT layout (page+, shapes)>
<!--
All positions and lengths are expressed in Virtual PiXels (VPX)
VPX: is the number of division (h_pos_division) per standard unit
(standard_h_pos_unit)
-->
<!ATTLIST layout
   id ID #IMPLIED
   h_pos_division CDATA #REQUIRED
   standard_h_pos_unit (mm | inch | twip | point) #REQUIRED
   font_URI CDATA #REQUIRED
>
<!ELEMENT page (frame*)>
<!ATTLIST page
   id CDATA #IMPLIED
   width CDATA #REQUIRED
   height CDATA #REQUIRED
>
<!ELEMENT frame (system+ | images | texts)>
<!-- Placement coordinates refer to the page origin (top-left corner) -->
```

```
<!ATTLIST frame
   id CDATA #IMPLIED
   top CDATA #REQUIRED
   left CDATA #REQUIRED
   right CDATA #REQUIRED
   bottom CDATA #REQUIRED
>
<!ELEMENT system (staff_piece+)>
<!--
The system is bounded by 2 eventmarkers in the spine domain
(first_event_marker_ref, last_event_marker_ref).
Preamble and postamble attributes define the lenght of the relative layout
sections. A value of 0 indicates the absence of these sections.
Placement coordinates refer to the frame origin (top-left corner)
-->
<!ATTLIST system
   id ID #IMPLIED
   top CDATA #REQUIRED
   left CDATA #REQUIRED
   right CDATA #REQUIRED
   bottom CDATA #REQUIRED
   first_event_marker_ref CDATA #REQUIRED
   last_event_marker_ref CDATA #REQUIRED
   preamble_length CDATA #REQUIRED
   postamble_length CDATA #REQUIRED
>
<!ELEMENT staff_piece (lyric_pieces*)>
<!--
v_pos: vertical position with respect to the system
show_key_signature, show_clef, show_time_signature: are preamble and
postamble properties
ossia: indicates of the staff_piece represents an ossia staff and in this
case it must have al least one staff_hiding child for the correct layout
-->
<!ATTLIST staff_piece
   id ID #IMPLIED
   staff_ref CDATA #REQUIRED
   v_pos CDATA #REQUIRED
   show_key_signature (yes | no) "yes"
   show_clef (yes | no) "yes"
   show_time_signature (yes | no) "yes"
```

```
   ossia (yes | no) "no"
>
<!ELEMENT lyric_pieces EMPTY>
<!-- visualization of lyrics -->
<!ATTLIST lyric_pieces
   lyric_ref IDREF #REQUIRED
   v_pos CDATA #REQUIRED
>
<!ELEMENT images EMPTY>
<!ELEMENT texts EMPTY>
<!--
Shape element is used to define custom graphic elements such as slur or tie
symbols.
The specification is made in SVG syntax. Adaptation of the shape to the
dimenson in the score  is defined by the reference point of the calling
element (i.e. for slur element the dimension is defined by the startnote
ref)
-->
<!ELEMENT shapes (svg)>
<!--
************************************************************************
************************************************************************
   Common elements for net layers
************************************************************************
************************************************************************
-->
<!ELEMENT part_ref EMPTY>
<!ATTLIST part_ref
   part_id IDREF #REQUIRED
   voice IDREF #IMPLIED
   spine_start_ref IDREF #IMPLIED
   spine_end_ref IDREF #IMPLIED
>
<!--
Partref is utilized for identifying a piece of content information about
source material. Pardid and part_ref identify the portion of LOS which the
file is instance of. spine_start_ref and spine_end_ref localize the part on
the spine they are defined as IMPLIED becouse if not specified they
inherhit from performance_instance ELEMENT.
-->
<!--
```

```
**************************************************************************
**************************************************************************
   Notational Layer
**************************************************************************
**************************************************************************
-->
<!ELEMENT notational (notation_instance | graphic_instance)+>
<!ELEMENT notation_instance (desc?, part_ref+, rights)>
<!ATTLIST notation_instance
   file_name CDATA #REQUIRED
   format CDATA #REQUIRED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
>
<!ELEMENT graphic_instance (desc?, part_ref+, rights)>
<!ATTLIST graphic_instance
   file_name CDATA #REQUIRED
   format CDATA #REQUIRED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
>
<!--
**************************************************************************
**************************************************************************
   Performance Layer
**************************************************************************
**************************************************************************
-->
<!ELEMENT performance (performance_instance+)>
<!--
The performance Layer is composed by zero or more  representation in a
subsymbolic form
-->
<!ELEMENT performance_instance (desc?, (MIDI | CSOUND | MPEG4), rights)>
<!--
Each clip refers to a particular rederization of the piece or a part of the
piece
-->
<!ATTLIST performance_instance
   file_name CDATA #REQUIRED
   spine_start_ref IDREF #REQUIRED
```

```
   spine_end_ref IDREF #REQUIRED
>
<!ELEMENT MIDI (MIDI_part_ref+)>
<!ATTLIST MIDI
   format (0 | 1 | 2) #REQUIRED
>
<!ELEMENT MIDI_part_ref EMPTY>
<!--
track and channel attributes are for the identification of parts within the
MIDI file, others attributes are similar to that of part_ref ELEMENT
-->
<!ATTLIST MIDI_part_ref
   part_id IDREF #REQUIRED
   voice IDREF #IMPLIED
   spine_start_ref IDREF #IMPLIED
   spine_end_ref IDREF #IMPLIED
   track CDATA #REQUIRED
   channel CDATA #REQUIRED
>
<!ELEMENT CSOUND (part_ref+)>
<!ELEMENT MPEG4 (part_ref+)>
<!--
**************************************************************************
**************************************************************************
   Audio Layer
**************************************************************************
**************************************************************************
-->
<!ELEMENT audio (clip+)>
<!-- The audio Layer is composed by one or more audio clips -->
<!ELEMENT clip (desc?, part_ref+, performers?, rights, COMMON_HEADER_INFO?,
index+)>
<!-- Each clip refers to a particular redition of the piece or a part of
the piece -->
<!ATTLIST clip
   file_name CDATA #REQUIRED
   format CDATA #REQUIRED
   duration CDATA #REQUIRED
   encoding CDATA #REQUIRED
   freq CDATA #REQUIRED
   n_bit CDATA #REQUIRED
```

```
   n_channel CDATA #REQUIRED
   bitrate CDATA #IMPLIED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
>
<!ELEMENT performers (performer)+>
<!ATTLIST performer
   type CDATA #IMPLIED
>
<!ELEMENT performer EMPTY>
<!ELEMENT index EMPTY>
<!--
This element contains the information about the indexes within the clip
expressed in absolute format (hour, minute, seconds, hundred of seconds)
starting from the beginnning of the clip.
-->
<!ATTLIST index
   time CDATA #REQUIRED
   measure CDATA #REQUIRED
   beat CDATA #REQUIRED
   %spine_attributes;
>
```

CAPITOLO 5

PNML

5.1 INTRODUZIONE

Il formato in cui ScoreSynth salva su disco le reti create è il PNML, del quale ora daremo alcuni cenni introduttivi prima di passare ad un approfondimento più completo.

Nel 1995, all'interno di un progetto volto alla creazione di uno standard internazionale[4] per le PNs di alto livello, appare l'idea di un formato comune di interscambio per la descrizione di PNs; negli anni successivi, durante incontri in cui si succedono varie proposte, si afferma come strumento di base l'XML, che, con la coordinazione di Ekkart Kindler, porta alla creazione del PNML (Petri Net Markup Language [JKW00]).

Lo scopo primario perseguito dalla creazione di uno standard di interscambio di PNs è comune ad una delle ragioni della nascita dell'MX: consentire ad applicazioni diverse che operano su tipi di dati compatibili di poter scambiare fra loro i lavori svolti: nel caso dell'MX il tipo di informazione di base è la musica sotto vari aspetti, nel caso del PNML sono le PNs, di ogni tipo.

Pur non essendo ancora stata approvata la standardizzazione del formato, allo stato attuale il PNML viene supportato da vari tools che operano sulle PNs, ed è stato discusso all'interno di vari incontri, fra i quali nel giugno 2003 nella "Conferenza Internazionale sulla Teoria e le Applicazioni delle Reti di Petri" [ATPN2003].

5.2 CONCETTI BASE

[4] v. http://www.daimi.au.dk/PetriNets/standardisation/: Lo standard è sviluppato dal "Joint Technical Committee on Information Technology" (JTC1) dell'ISO/IEC, e più precisamente dal sottocomitato SC7 (Software and System Engineering, http://www.jtc1-sc7.org/). Il gruppo di lavoro specifico è il WG19 (Open Distributed Processing and Modelling Languages), che all'interno dei suoi progetti conta anche il 15909-1, volto a produrre un standard sulle Reti di Petri diviso in 3 parti, la seconda delle quali interessa proprio la realizzazione di un formato di trasferimento di PNs.

Il PNML, com'è stato accennato, è stato creato con lo scopo di essere indipendente dai tools specifici che lo utilizzano e dalla piattaforma, volendo in più supportare diversi "dialetti" di PNs ed essere estensibile; inoltre, grazie alla tecnologia XML, è leggibile dall'uomo e trattabile con semplici editor testuali.

Partendo da queste premesse, i concetti che governano il PNML sono *flessibilità*, *compatibilità* e *non-ambiguità*.

Flessibilità significa che il PNML deve poter rappresentare ogni tipo di PN, con le sue specifiche estensioni e caratteristiche, non obbligando ad ignorare informazioni contenute in particolari dialetti quando occorre salvare una rete. Per consentire questa flessibilità, il PNML considera una PN come un grafo etichettato, in cui tutte le eventuali informazioni addizionali possono essere poste in etichette associate alla rete stessa, ai suoi nodi o ai suoi archi.

La *compatibilità* è raggiunta specificando con delle convenzioni come definire etichette con un particolare significato, consentendo così l'interscambio di più informazioni possibili tra diversi tipi di PNs. All'interno di un *Conventions Document* viene descritta sia la sintassi che la semantica delle possibili estensioni del PNML. Quando viene creato un nuovo tipo di PN, i tipi di etichette possono essere scelti dal Conventions Document, stabilendo così precisamente il loro significato. In questo modo si consente l'interpretazione della rete anche ad un tool non appositamente sviluppato per il tipo di PN in oggetto.

L'*ambiguità* è rimossa assicurando che la PN originale ed il suo particolare tipo possano essere determinate univocamente dalla loro rappresentazione in PNML. Per consentire ciò il PNML supporta la definizione di diversi tipi di PNs, specificati mediante una "Petri Net Type Definition" (*PNTD*), che indica quali sono le etichette permesse per la particolare PN trattata.

Strutturalmente la definizione del PNML comprende quindi tre parti principali che sviluppano i concetti sopra esposti: il *Meta Model*, il *Conventions Document* e il *PNTD (Petri Net Type Definition)*. Le relazioni fra loro esistenti sono visibili in Fig. 11.

Il *Meta Model* è un modello che costituisce la parte invariante della definizione del PNML, specificando la struttura base di un file PNML, indipendentemente dal tipo di PN in questione. Il Meta Model è quindi invariante, e la sua struttura di base è rappresentata in notazione UML in Fig. 12 (non sono presenti per semplicità i concetti di PNML strutturato e modulare, spiegati più avanti nel paragrafo).

Il *Conventions Document* è un documento che contiene proprietà comuni a tutti i tipi di PNs; all'occorrenza può evolversi incorporando nuove caratteristiche ritenute di interesse pubblico.

Il *PNTD* è la parte in cui vengono introdotti gli elementi distintivi relativi alle particolari PNs, estendendo le PNs base con peculiarità aggiuntive. In questo modo, basandosi sul Meta Model e sul Conventions Document, possono essere creati vari tipi di PNs, con i loro PNTD corrispondenti.

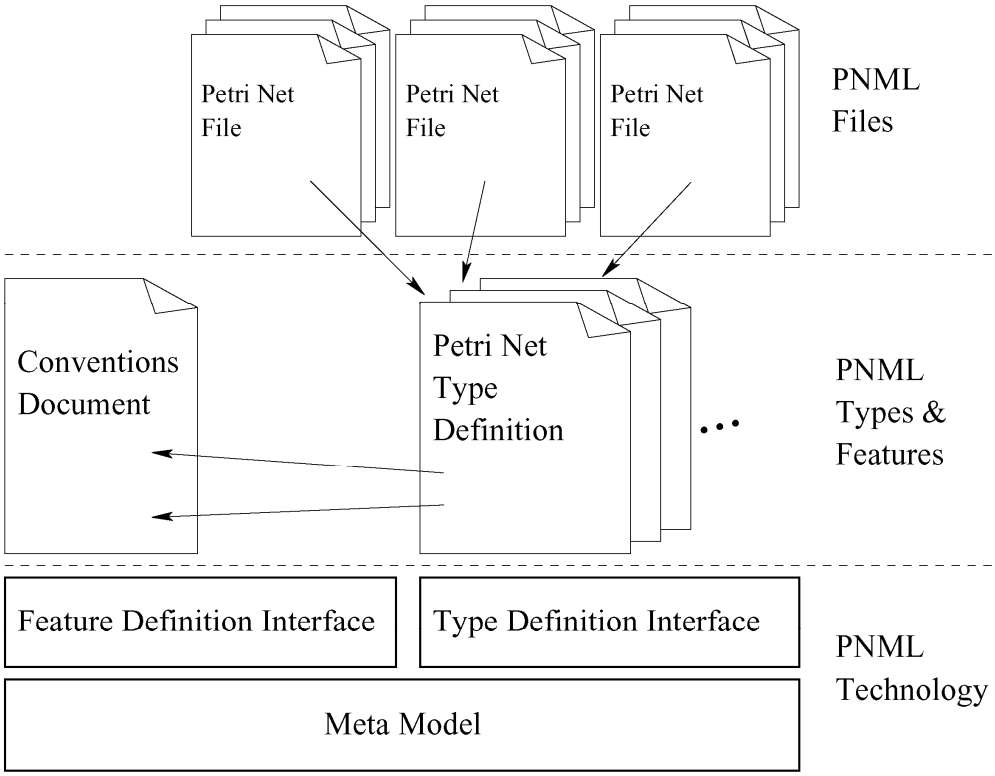


Fig. 11. Panoramica delle parti del PNML

Per consentire una maggiore flessibilità descrittiva, nel PNML sono stati implementati, sempre indipendentemente dal particolare tipo di PNs da trattare, due soluzioni che facilitano la descrizione di PNs di grosse dimensioni e l'astrazione.

Il primo approccio, il più semplice, utilizza *pagine* e *riferimenti*: la PN può essere disegnata su più pagine, che sono oggetti che possono contenere altri oggetti, anche altre pagine. Un riferimento è semplicemente un nodo puntatore ad un altro nodo presente in qualsiasi pagina della rete, con l'unica regola di non creare riferimenti ciclici: in questo modo ogni nodo di riferimento alla fine punta ad esattamente un posto o una transizione della PN. Gli archi della rete possono puntare comunque solamente a nodi presenti nella stessa pagina[5]. I nodi di riferimento possono avere delle etichette per specificare il loro aspetto grafico, ma

[5] Per analogia, non è possibile disegnare una rete connettendo con un arco due nodi presenti su diversi fogli di carta.

non informazioni relative ai nodi a cui si riferiscono, che hanno le proprie etichette per questo scopo.

Il PNML che utilizza *pagine* e *riferimenti* è chiamato *PNML strutturato*, che estende il *PNML di base*. Pur essendo ampiamente riconosciuto da vari tools (ad esempio, il Design/CPN [DESCPN]), il PNML strutturato non consente un'astrazione spinta, ma soltanto una rappresentazione più semplice di reti di grandi dimensioni; con una semplice trasformazione si può "appiattire" la rete strutturata in una rete semplice semanticamente equivalente. Per far ciò occorre soltanto dereferenziare i nodi con riferimenti e ignorare i bordi delle pagine presenti. Tutto ciò è possibile anche semplicemente con l'utilizzo di un foglio di stile XSLT [XSLT].

Un altro approccio che consente di applicare il concetto di astrazione ad un livello maggiore nella compilazione di un documento PNML è il *PNML modulare*, ulteriore estensione del *PNML strutturato*. In questo caso vengono definiti dei *moduli* che possono essere istanziati in fase di run-time e che corrispondono a porzioni di PNML; viene poi divisa la parte di dichiarazione dell'interfaccia del modulo rispetto alla sua implementazione.

Pur essendo più potente e duttile rispetto all'uso di pagine e riferimenti, il PNML modulare non è molto utilizzato, e, soprattutto, la sua semantica cambia rispetto ai tools che lo supportano. Questo avviene nella maggior parte dei casi perché il concetto modulare viene usato specificamente per modellare delle caratteristiche proprie dei singoli tipi di PNs, come nelle reti usate da Renew [RENEW] o nel progetto Moses [MOSES].

Pur non essendo allo stato attuale implementata nessuna estensione al PNML di base nello ScoreSynth, negli ultimi paragrafi del capitolo verranno comunque approfonditi i concetti di pagine, riferimenti e moduli, allo scopo di fornire una corretta visione d'assieme delle potenzialità che potranno eventualmente essere aggiunte in nuove releases del progetto. Si tenga presente che in ogni caso, con l'applicazione di fogli di stile XSLT è possibile convertire un documento PNML strutturato o modulare nell'equivalente di base.

5.3 STRUTTURA DEL PNML

Approfondiamo ora i concetti di base già esposti, addentrandoci nella struttura di un documento PNML.

5.3.1 META MODEL

Meta Model. Un documento PNML che sia aderente al Meta Model (Fig. 12) può contenere una o più PNs, formate da *oggetti* che rappresentano la struttura della rete, costituiti, nel PNML di base, da *posti* e *transizioni* (chiamati anche *nodi*), e da *archi*[6]. Ogni oggetto ha un proprio *identificatore* unico.

Etichette (Labels). Per poter aggiungere informazioni agli elementi di un documento PNML, essi possono avere *etichette*. Tipicamente, un'etichetta rappresenta ad esempio il nome di un nodo, la marcatura iniziale di un posto, la sua capacità, il peso di un arco, o il colore di un oggetto. I tipi di etichette consentite vengono specificate nella definizione del PNTD per un particolare tipo di PN.

Vengono distinti due tipi di etichette: *annotazioni* e *attributi*. Un'annotazione ha tipicamente una sua rappresentazione grafica testuale posta vicino all'oggetto corrispondente a cui si riferisce. Esempi di annotazione sono i nomi dei nodi, le marche, le iscrizioni degli archi. Un attributo, per contro, non ha solitamente una rappresentazione grafica propria, ma indica un parametro riguardante la forma, lo stile o il colore dell'oggetto stesso. Possibili attributi possono essere: colore e grandezza di un nodo, font utilizzato per le iscrizioni, tipi di nodi di ingresso/uscita. A differenza degli attributi, le annotazioni richiedono a volte la specifica della posizione (assoluta o relativa all'oggetto) in cui devono essere visualizzate.

[6] Come si può notare in fig. 2, nel Meta Model è possibile che un arco connetta nodi dello stesso tipo. Questo è consentito in alcuni tipi di PNs, ma non nelle reti trattate nel presente lavoro.

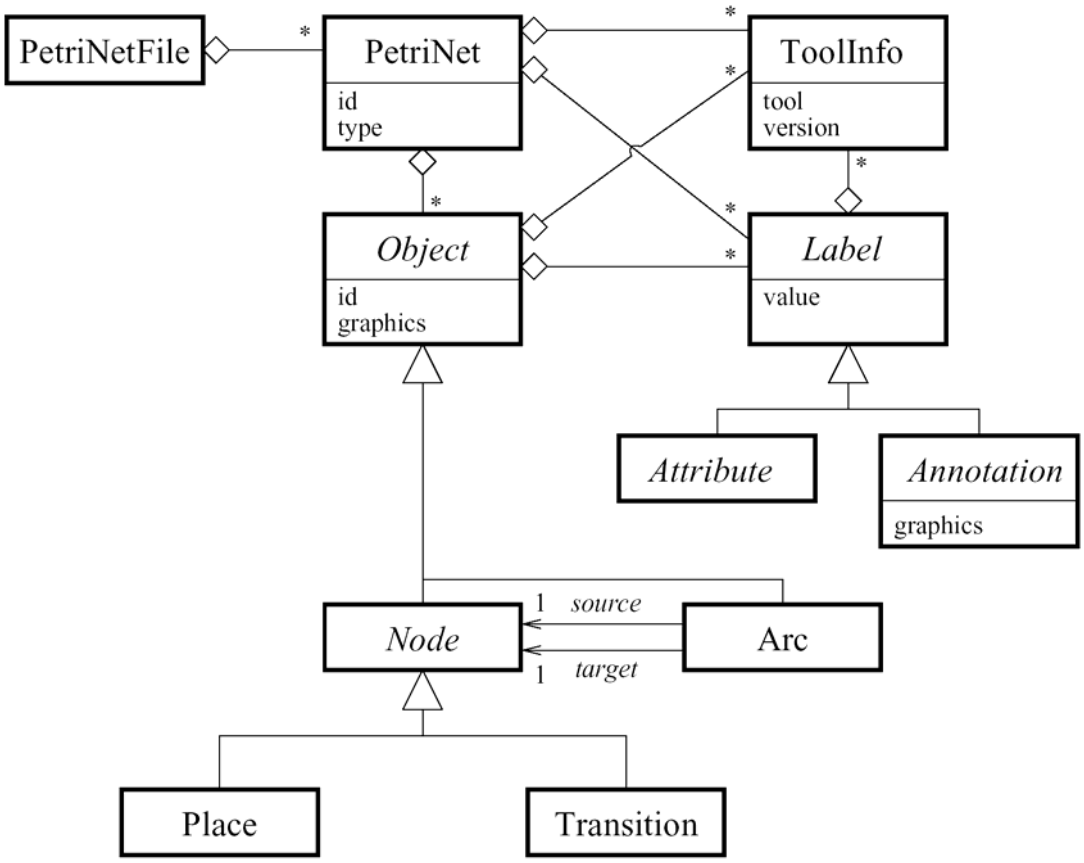


Fig. 12. Meta Model PNML di base

Informazioni grafiche. Ogni oggetto ed ogni annotazione incorporano informazioni grafiche sulla loro rappresentazione. Per i nodi viene specificata la posizione, per gli archi le posizioni dei loro punti intermedi. Per le annotazioni, la distanza relativa al proprio oggetto[7]. Solitamente il punto di riferimento di un nodo è il suo centro, mentre per gli archi, è il punto centrale del loro primo segmento. In estensioni future del PNML è in studio la possibilità di definire esplicitamente quale punto considerare di riferimento per ogni oggetto.

Tool specific information. In alcune applicazione può essere necessario memorizzare nel documento PNML delle informazioni che vengono utilizzate solamente dal tool specifico. In questo caso è possibile incorporare negli oggetti e nelle etichette queste informazioni, unitamente al nome del tool ed alla sua versione. Il Meta Model specifica soltanto come marcare un'informazione come *tool specific*, lasciando libera scelta sul suo contenuto all'applicazione che l'incorpora; in questo modo gli strumenti che lavorano su un documento

[7] Nel caso di annotazioni che si riferiscono alla rete stessa, le coordinate sono assolute.

PNML possono semplicemente ignorare le parti contrassegnate come tool specific e compilate da altri strumenti[8].

5.3.2 PNTDs

Il Meta Model PNML descrive come già detto la struttura di un documento PNML indipendentemente dal tipo di PN utilizzata, prendendo in considerazione soltanto le informazioni standard incluse in tutti i tipi di PNs. Per espandere le specifiche, includendo particolarità proprie di un determinato tipo di rete, vengono utilizzati i PNTDs. Un PNTD specifica quindi la sintassi delle etichette usate in un particolare tipo di PN, precisando le convenzioni generali espresse nel Meta Model.

5.3.3 Conventions Document

In linea di principio, un PNTD può essere definito liberamente. In pratica, esiste un documento che si frappone tra il Meta Model e un PNTD: il *Conventions Document*. Il suo scopo è quello di definire la sintassi di etichette ritenute standard, utilizzate nelle maggior parte di tipi di PNs, in modo da consentire l'interscambio di informazioni fra vari tools che trattano tipi di PNs diverse ma abbastanza simili. Un PNTD quindi sceglierà i nomi delle etichette contenute nel Conventions Document, per quanto possibile.

E' da notare che, mentre il Meta Model è invariante, definisce cioè come devono essere specificate le strutture degli oggetti di un documento PNML qualsiasi, il Conventions Document è un work-in-progress, che viene modificato dagli organi di standardizzazione con l'intento di includere la maggior parte di etichette da considerarsi standard.

5.4 Implementazione

In questo paragrafo verrà descritta l'implementazione dei concetti di base del PNML in formato XML.

Fra le varie tecnologie che consentono di specificare la struttura di un dialetto XML, per lo sviluppo del PNML è stato scelto il *RELAX NG* [RELAXNG], un XML Schema [XSCHEMA] che propone un concetto modulare necessario per separare nettamente il Meta Model dai PNTDs.

[8] In generale in PNML si consiglia di utilizzare il meno possibile il concetto di *tool specific informations*.

5.4.1 META MODEL

Il Meta Model del PNML nella release 1.3 è specificato con una grammatica RELAX NG disponibile sul sito ufficiale del PNML [PNML]. Esso costituisce la parte invariante della codifica PNML, che descrive la struttura di un documento, indipendentemente dal tipo di PN usata.

Elementi PNML. Gli elementi[9] principali specificati nel Meta Model sono esattamente le classi concrete[10] del diagramma in Fig. 12, inclusa la classe Graphics, non presentata in figura per semplicità. Nella Tab. 1 sono visibili le corrispondenze tra Meta Model, elementi ed attributi XML. In Fig. 13 è invece presentato un frammento della grammatica RELAX NG che descrive la struttura dell'elemento `<place>`.

Classe	*Elemento XML*	*Attributi XML*
PetriNetFile	`<pnml>`	
PetriNet	`<net>`	`id`: ID `type`: anyURI
Place	`<place>`	`id`: ID
Transition	`<transition>`	`id`: ID
Arc	`<arc>`	`id`: ID `source`: IDRef (Nodo) `target`: IDRef (Nodo)
ToolInfo	`<toolspecific>`	`tool`: string `version`: string
Graphics	`<graphics>`	

Tab. 1. Elementi XML mutuati dal Meta Model PNML

```
<define name="place.content">
   <a:documentation>
```

[9] Il termine "elemento" nella trattazione seguente indica precisamente un elemento XML.
[10] In un diagramma UML una classe è concreta se il suo nome non è scritto in corsivo.

```
      A place may have several labels (place.labels) and
      the same content as a node.
    </a:documentation>
    <interleave>
      <ref name="place.labels"/>
      <ref name="node.content"/>
    </interleave>
  </define>

  <define name="place.labels">
    <a:documentation>
      A place may have unspecified many labels. This pattern should be
      used within a PNTD to define the net labels.
    </a:documentation>
    <empty/>
  </define>

  <define name="node.content">
    <a:documentation>
      A node has a unique identifier.
    </a:documentation>
    <attribute name="id">
      <data type="ID"/>
    </attribute>
    <interleave>
      <a:documentation>
        The sub-elements of a node occur in any order.
        A node may consist of grahical and tool specific information.
      </a:documentation>
      <optional>
        <element name="graphics">
          <ref name="nodegraphics.content"/>
        </element>
      </optional>
      <zeroOrMore>
        <ref name="toolspecific.element"/>
      </zeroOrMore>
    </interleave>
  </define>
```

Fig. 13. Frammento della grammatica RELAX NG del Meta Model

Etichette. Come è possibile notare dal frammento di grammatica presentato, nel Meta Model non vengono definite le etichette; questo è un compito demandato al Conventions

Document ed ai PNTDs. Un elemento XML non definito nel Meta Model è quindi considerato un'etichetta dell'elemento PNML in cui si presenta.

Graphics. Le informazioni grafiche relative agli oggetti PNML sono indicate negli elementi `<graphics>`, la cui struttura dipende dall'oggetto a cui si riferiscono (PN, nodo, arco, o annotazione). In Tab. 2 vengono indicati i possibili sottoelementi di `<graphics>`, dipendenti dal contesto. Da notare che lo stesso sottoelemento `<position>` è richiesto ed unico per posti e transizioni, mentre per gli archi, rappresentando le coordinate dei punti intermedi fra i nodi connessi, è in numero maggiore o uguale a zero.

L'elemento `<fill>` consente di definire lo sfondo di un nodo o di un'annotazione, con possibilità di specificare una colorazione sfumata o un'immagine di background.

Nell'elemento `<line>` è possibile specificare colore, spessore e stile dei tratti.

Sia nella specifica dei colori che nell'elemento `<font>` viene utilizzato lo standard CSS2 [CSS2].

Si segnala che è in corso un progetto [Ste02] di compilazione di un foglio di stile XSLT che consenta di convertire un documento PNML nella sua rappresentazione grafica descritta in SVG [SVG], un linguaggio di descrizione di grafica vettoriale basato sull'XML.

Elemento XML padre	*Sottoelementi di* `<graphics>`
`<net>`	(nessuno, allo stato attuale)
`<place>`, `<transition>`	`<position>`, `<dimension>`, `<fill>`, `<line>`
`<arc>`	`<position>`, `<line>`
(annotazione)	`<offset>`, `<fill>`, `<line>`, `<font>`

Tab. 2. Possibili sottoelementi di `<graphics>`

5.4.2 PNTDs

Un PNTD definisce (solitamente con un'altra grammatica RELAX NG) la sintassi XML delle etichette di reti e di loro oggetti. Come è infatti possibile notare nella Fig. 13, nel frammento di Meta Model presentato le definizioni delle etichette dei posti vengono inizialmente lasciate vuote, lasciando il compito di specificarle al Conventions Document ed al PNTD.

La creazione di un PNTD specializzato per il trattamento di documenti PNML che descrivano PNs musicali è stato un passaggio importante all'interno del presente progetto; per questo è stato dedicato ad esso un paragrafo più avanti che descrive dettagliatamente il PNTD.

5.4.3 CONVENTIONS DOCUMENT

Come già spiegato, il Conventions Document specifica (attraverso una grammatica RELAX NG) la struttura delle etichette standard.

Allo stato attuale, il Conventions Document 0.1 è diviso in due sezioni: la prima definisce una serie di short cuts che facilitano la definizione di etichette, la seconda le etichette standard vere e proprie. In Tab. 3 e in Tab. 4 vengono presentati short cuts ed etichette standard con una breve descrizione delle caratteristiche. Si noti che il *nome* degli short cuts e delle etichette non corrisponde necessariamente all'elemento XML corrispondente, ma è solo un riferimento da utilizzare con il tag `<ref>` all'interno di una grammatica RELAX NG. Nel Conventions Document viene specificato infatti con un tag `<define>` l'associazione tra nome ed elemento XML.

Nome short cut	*Descrizione*
`attribute.content`	Un attributo, senza informazioni grafiche
`annotationstandard.content`	Un'annotazione che può contenere Informazioni grafiche e tool specific
`simpletextlabel.content`	Una semplice etichetta testuale (l'unione di `attribute.content` e `annotationstandard.content`)
`nonnegativeintegerlabel.content`	Un'etichetta contenente un intero positivo
`complexlabel.content`	Un'etichetta strutturata

Tab. 3. Short cuts contenuti nel Conventions Document

Nome etichetta e Elemento XML	*Descrizione*
`Name` `<name>`	Etichetta testuale che specifica il nome di un oggetto

`PTMarking` `<initialMarking>`	Etichetta numerica che specifica il numero di marche iniziali in una rete P/T
`PTArcInscription` `<inscription>`	Etichetta numerica che specifica il peso di un arco in una rete P/T

Tab. 4. Etichette standard definite nel Conventions Document

Ricordiamo che il Conventions Document è una parte variante della definizione del PNML, che ha una propria politica di compilazione e un team che si occupa del compito. Una regola della politica di gestione è che le modifiche apportate devono essere compatibili sia verso l'alto che verso il basso. Questo comporta che, una volta aggiunta una definizione al Conventions Document, essa deve rimanere invariata nelle stesure successive.

5.5 IL PNTD MX

Per il salvataggio delle PNs create con ScoreSynth è stato creato un PNTD specifico, che incorpora le estensioni delle reti musicali. In Tab. 5 vengono proposte le definizioni usate per ridefinire le etichette del Meta Model `net.labels`, `place.labels`, `transition.labels` e `arc.labels`. In Tab. 6 vengono elencati i campi relativi alla singola ridefinizione. Il PNTD MX completo è visibile in Fig. 14. Tutti le etichette definite sono opzionali: se non vengono specificate ScoreSynth applica i valori interni di default.

Nome definizione e Elemento XML	*Descrizione*
`MXIsInput` `<isInput>`	In un posto o in una transizione, indica se è un nodo di input di una sottorete
`MXIsOutput` `<isOutput>`	In un posto o in una transizione, indica se è un nodo di output di una sottorete
`MXCapacity` `<capacity>`	Capacità associata ad un posto
`MXFile` `<mxFile>`	Nome del file associato ad un posto o ad una transizione (anche una sottorete PNML)
`MXTokensWeight`	Peso in marche di un arco

`<tokensWeight>`	
`MXProbWeight` `<probWeight>`	Peso probabilistico di un arco
`MXArcTension` `<tension>`	Tension di un arco (curvatura grafica)
`MXPlaceTrackInfo` `<trackInfo>`	ID, nome e priorità della traccia associata ad un posto
`MXVtuScale` `<vtuScale>`	Scala da applicare agli eventi della spine del file MX associato a posti e transizioni

Tab. 5. Definizioni degli elementi utilizzati nel PNTD MX

Nome etichetta	*Nomi etichette associate*
`net.labels`	`attribute.content` (Il nome della PN)
`place.labels`	`MXIsInput, MXIsOutput, Name, PTMarking, MXCapacity, MXFile, MXPlaceTrackInfo, MXVtuScale`
`transition.labels`	`MXIsInput, MXIsOutput, Name, MXFile, MXVtuScale`
`arc.labels`	`MXTokensWeight, MXProbWeight, MXArcTension, MXFile`

Tab. 6. Etichette associate agli elementi `<net>`, `<place>`, `<transition>` e `<arc>`

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
jing (RELAX NG validator) does not recognize SGML's !DOCTYPE
Thus, for validating this schema the following line must be hided.
-->
<!--
```

```
<!DOCTYPE grammar PUBLIC "-//thaiopensource//DTD RNG 20010705//EN" "">
-->

<grammar ns="http://www.maxentia.com/pnml/mxNet"
xmlns="http://relaxng.org/ns/structure/1.0"
xmlns:a="http://relaxng.org/ns/compatibility/annotations/1.0"
datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

   <a:documentation>
        Petri Net Type Definition
        for MX nets (musical PNs) (based on basic PNML)
      RELAX NG implementation of mxNetb.pntd
      version: 1.0 (2003)
        Adriano Baratè, adriano@cercatore.com
   </a:documentation>
   <a:documentation>
        We include PNML with the correct URI
        for our Petri Net Type Definition.
   </a:documentation>

   <include href="http://www.informatik.hu-berlin.de/top/pnml/basicPNML.rng">
        <define name="nettype.uri" combine="choice">
             <a:documentation>
                  We define the net type URI declaring the namespace
                  of this Petri net type definition.
             </a:documentation>
             <value>http://www.maxentia.com/pnml/mxNet</value>
        </define>
   </include>
   <include href="http://www.informatik.hu-
berlin.de/top/pnml/conv.rng"/>

   <!-- Label definitions used when defining MX labels afterwards -->
   <define name="MXIsInput">
        <a:documentation>
             Label definition for input node boolean flag in MX nets
             <date>2004-01-27</date>
        </a:documentation>
        <element name="isInput">
             <ref name="attribute.content"/>
```

```
        </element>
    </define>

    <define name="MXIsOutput">
        <a:documentation>
            Label definition for output node boolean flag in MX nets
            <date>2004-01-27</date>
        </a:documentation>
        <element name="isOutput">
            <ref name="attribute.content"/>
        </element>
    </define>

    <define name="MXCapacity">
        <a:documentation>
            Label definition for place capacity in MX nets
            <date>2003-09-11</date>
        </a:documentation>
        <element name="capacity">
            <ref name="nonnegativeintegerlabel.content"/>
        </element>
    </define>

    <define name="MXFile">
        <a:documentation>
            Label definition for the file associated to a
            place/transition
            <date>2004-01-27</date>
        </a:documentation>
        <element name="mxFile">
            <ref name="attribute.content"/>
        </element>
    </define>

    <define name="MXTokensWeight">
        <a:documentation>
            Label definition for arcs tokens weight in MX nets
            <date>2003-09-11</date>
        </a:documentation>
        <element name="tokensWeight">
```

```
            <ref name="nonnegativeintegerlabel.content"/>
        </element>
    </define>

    <define name="MXProbWeight">
        <a:documentation>
            Label definition for arcs probabilistic weight in MX nets
            <date>2003-09-11</date>
        </a:documentation>
        <element name="probWeight">
            <ref name="nonnegativeintegerlabel.content"/>
        </element>
    </define>

    <define name="MXArcTension">
        <a:documentation>
            Label definition for arcs tension in MX nets
            <date>2004-01-27</date>
        </a:documentation>
        <element name="tension">
            <ref name="attribute.content"/>
        </element>
    </define>

    <define name="MXPlaceTrackInfo">
        <a:documentation>
            Label definition for places track infos in MX nets
            <date>2004-03-24</date>
        </a:documentation>
        <element name="trackInfo">
            <attribute name="id">
                <data type="string"/>
            </attribute>
            <attribute name="priority">
                <data type="nonNegativeInteger"/>
            </attribute>
            <attribute name="trackName">
                <data type="string"/>
            </attribute>
        </element>
```

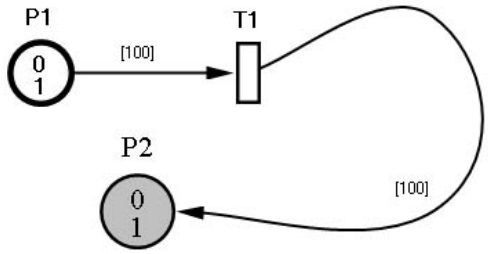
```
    </define>

    <define name="MXVtuScale">
        <a:documentation>
            Label definition for vtus scale in MX nets
            <date>2004-04-17</date>
        </a:documentation>
        <element name="vtuScale">
            <ref name="attribute.content"/>
        </element>
    </define>

    <!--     Labels used in MX nets-->
    <define name="net.labels" combine="interleave">
        <a:documentation>
            An MX net may have a name.
        </a:documentation>
        <optional>
        <ref name="attribute.content"/>
        </optional>
    </define>

    <define name="place.labels" combine="interleave">
        <a:documentation>
            A place of an MX net may have a name,
            an initial marking, a capacity and an associated MX file.
        </a:documentation>
        <interleave>
            <optional>
                <ref name="MXIsInput"/>
            </optional>
            <optional>
                <ref name="MXIsOutput"/>
            </optional>
            <optional>
                <ref name="Name"/>
            </optional>
            <optional>
                <ref name="PTMarking"/>
            </optional>
```

```
            <optional>
                <ref name="MXCapacity"/>
            </optional>
            <optional>
                <ref name="MXFile"/>
            </optional>
            <optional>
                <ref name="MXPlaceTrackInfo"/>
            </optional>
            <optional>
                <ref name="MXVtuScale"/>
            </optional>
        </interleave>
    </define>

    <define name="transition.labels" combine="interleave">
        <a:documentation>
            A transition of an MX net may have a name.
        </a:documentation>
        <interleave>
            <optional>
                <ref name="MXIsInput"/>
            </optional>
            <optional>
                <ref name="MXIsOutput"/>
            </optional>
            <optional>
                <ref name="Name"/>
            </optional>
            <optional>
                <ref name="MXFile"/>
            </optional>
            <optional>
                <ref name="MXVtuScale"/>
            </optional>
        </interleave>
    </define>

    <define name="arc.labels" combine="interleave">
        <a:documentation>
```

```
            An arc of an MX net may have a tokens weight and
            a probabilistic weight.
        </a:documentation>
        <interleave>
            <optional>
                <ref name="MXTokensWeight"/>
            </optional>
            <optional>
                <ref name="MXProbWeight"/>
            </optional>
            <optional>
                <ref name="MXArcTension"/>
            </optional>
        </interleave>
    </define>
</grammar>
```

Fig. 14. PNTD MX

A titolo di esempio proponiamo la seguente PN che genera il file PNML in Fig. 15.




```
<?xml version="1.0" encoding="utf-8"?>
<pnml xmlns="">
        <net id="n1" type="http://www.maxentia.com/pnml/mxNet">
            <text/>
            <place id="p1">
                <isInput>
                    <text>false</text>
                </isInput>
                <isOutput>
                    <text>false</text>
                </isOutput>
                <name>
                    <text>P1</text>
```

```
                    <graphics>
                        <offset x="0" y="-40"/>
                        <font family="Microsoft Sans Serif"/>
                    </graphics>
                </name>
                <initialMarking>
                    <text>0</text>
                </initialMarking>
                <capacity>
                    <text>1</text>
                </capacity>
                <trackInfo id="-1" priority="0" trackName="Default"/>
                <vtuScale>
                    <text>1</text>
                </vtuScale>
                <graphics>
                    <position x="100" y="200"/>
                    <dimension x="20" y="20"/>
                    <fill color="#ffffff"/>
                    <line color="#000000" width="5"/>
                </graphics>
            </place>
            <place id="p2">
                <isInput>
                    <text>false</text>
                </isInput>
                <isOutput>
                    <text>false</text>
                </isOutput>
                <name>
                    <text>P2</text>
                    <graphics>
                        <offset x="0" y="-45"/>
                        <font family="Times New Roman"/>
                    </graphics>
                </name>
                <initialMarking>
                    <text>0</text>
                </initialMarking>
                <capacity>
```

```
                <text>1</text>
            </capacity>
            <trackInfo id="-1" priority="0" trackName="Default"/>
            <vtuScale>
                <text>1</text>
            </vtuScale>
            <graphics>
                <position x="170" y="300"/>
                <dimension x="25" y="25"/>
                <fill color="#c0c0c0"/>
                <line color="#000000" width="3"/>
            </graphics>
        </place>
        <transition id="t1">
            <isInput>
                <text>false</text>
            </isInput>
            <isOutput>
                <text>false</text>
            </isOutput>
            <name>
                <text>T1</text>
                <graphics>
                    <offset x="0" y="-38"/>
                    <font family="Microsoft Sans Serif"/>
                </graphics>
            </name>
            <vtuScale>
                <text>1</text>
            </vtuScale>
            <graphics>
                <position x="250" y="200"/>
                <dimension x="13" y="40"/>
                <fill color="#ffffff"/>
                <line color="#000000" width="3"/>
            </graphics>
        </transition>
        <arc id="a1" source="p1" target="t1">
            <tokensWeight>
                <text>1</text>
```

```
                </tokensWeight>
                <probWeight>
                    <text>100</text>
                    <graphics>
                        <offset x="0" y="-13"/>
                        <font family="Microsoft Sans Serif"
                            size="8.25pt"/>
                    </graphics>
                </probWeight>
                <tension>
                    <text>0</text>
                </tension>
                <graphics>
                    <line color="#000000" width="2"/>
                </graphics>
            </arc>
            <arc id="a2" source="t1" target="p2">
                <tokensWeight>
                    <text>1</text>
                </tokensWeight>
                <probWeight>
                    <text>100</text>
                    <graphics>
                        <offset x="-21" y="-11"/>
                        <font family="Microsoft Sans Serif"
                            size="8.25pt"/>
                    </graphics>
                </probWeight>
                <tension>
                    <text>0.8</text>
                </tension>
                <graphics>
                    <line color="#000000" width="2"/>
                    <position x="384" y="159"/>
                    <position x="390" y="296"/>
                </graphics>
            </arc>
        </net>
```

```
</pnml>
```

Fig. 15. PNML relativo alla PN di esempio

5.6 PNML STRUTTURATO

In questo paragrafo e nel successivo accenniamo alle estensioni del PNML che consentono qualche tipo di astrazione.

Il PNML strutturato aggiunge alle definizioni del Meta Model di base quelle relative a pagine e riferimenti (v. Fig. 16). Un esempio di PNML strutturato è visibile in Fig. 17. Una pagina può contenere tutti gli oggetti che può contenere una rete, incluse altre pagine e riferimenti. Un riferimento (indicato dai tag `<referencePlace>` o `<referenceTransition>`) punta attraverso l'attributo `ref` ad un nodo esistente nella rete. Si ricorda che gli elementi grafici, le etichette e le informazioni tool specific che possono essere inseriti nei riferimenti non hanno un reale significato, in quanto vengono semplicemente ignorati durante la risoluzione dei puntatori alla creazione della rete "appiattita".

La semantica di pagine e riferimenti è semplice: vengono omesse le pagine e risolti tutti i riferimenti rimpiazzando i nodi riferimento coi nodi puntati, ed adattando i nodi di inizio e fine degli archi. La richiesta di non ciclicità nella creazione dei riferimenti assicura che ogni PN strutturata possa essere risolta in una PN di base.

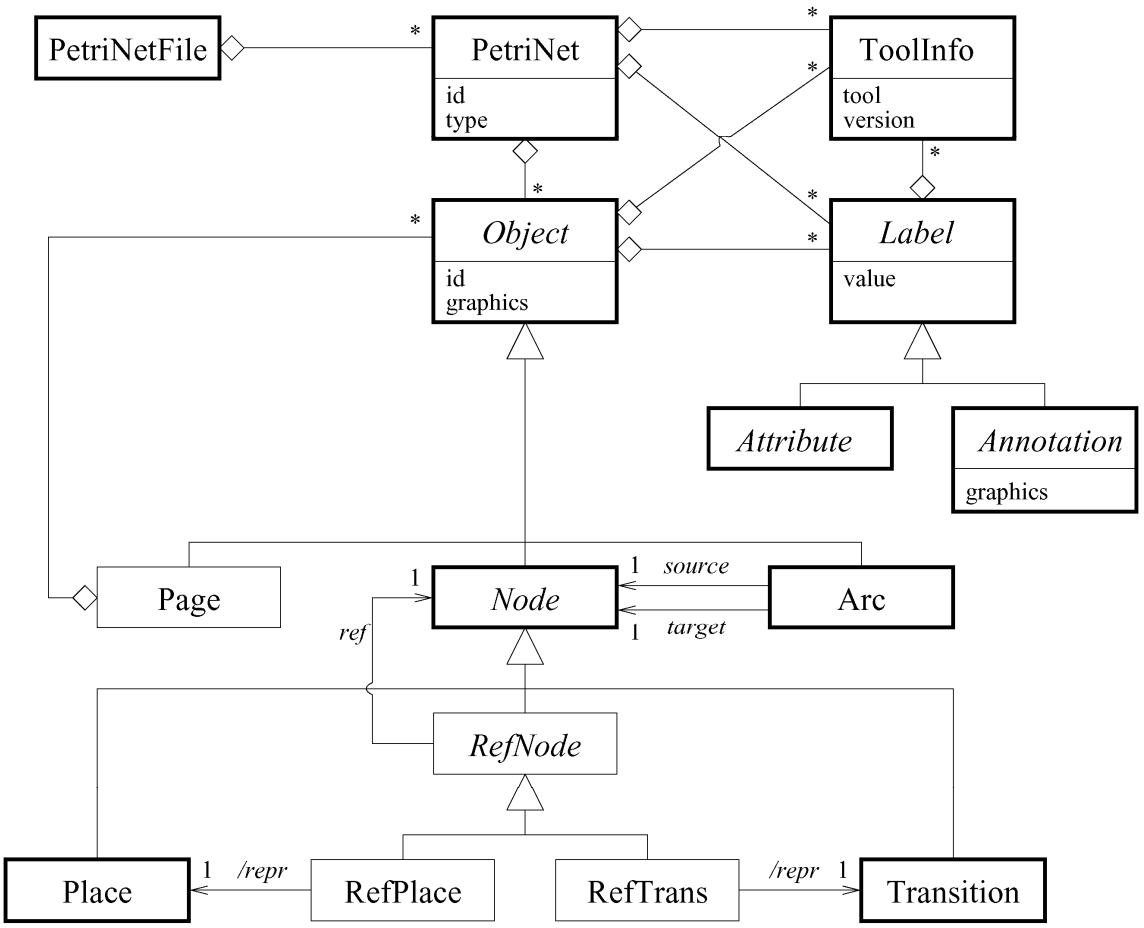


Fig. 16. Meta Model del PNML strutturato

```
<page id="pg1">
        <name>
             <value>Pagina di esempio</value>
        </name>

        <referencePlace id="rp1" ref="p1">
             <name>...</name>
             <graphics>
                  <position x="20" y="20"/>
             </graphics>
        </referencePlace>

        <referenceTransition id="rt1" ref="t1">
        ...
        </referenceTransition>
```

```
        <place id="p2">...</place>
        <transition id="t2">...</transition>
        <arc id="a2" source="rp1" target="t2">
        ...
        </arc>
        ...
</page>
```

Fig. 17. Esempio di PNML strutturato

5.7 PNML MODULARE

Il PNML modulare si basa sul PNML strutturato, aggiungendo altri elementi che consentono la possibilità di una vera astrazione nella descrizione di una PN.

5.7.1 CONCETTI DI BASE

Per illustrare i concetti del PNML modulare ricorreremo ad un esempio.

In Fig. 18 si può osservare la *definizione di un modulo* `M1`, che incorpora due posti `x` e `y`, due transizioni `t1` e `t2`, ed alcuni archi. La particolare caratteristica dei moduli è che la loro implementazione interna non è visibile esternamente. Per consentire l'utilizzo del modulo viene definita al suo interno un'*interfaccia*, in questo caso costituita dal posto `p1`, che all'esterno viene *importato* dall'interno del modulo, e dal posto `p2`, che viene invece *esportato* all'interno.

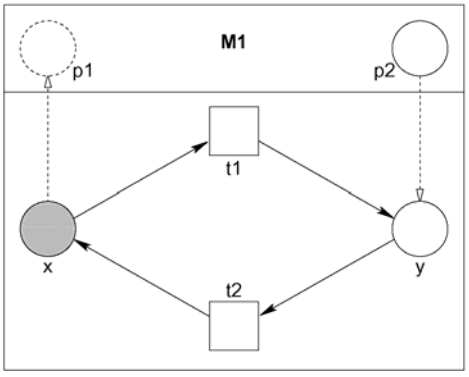


Fig. 18. Modulo `M1`

Il posto `p1` è un parametro formale, che viene specificato quando si istanzia un modulo (v. più avanti) ed è rappresentato da un cerchio tratteggiato. Il posto esportato `p2` invece può essere utilizzato nell'ambiente di istanziazione del modulo ed è rappresentato con un cerchio dal tratto continuo.

Il concetto di nodi importati ed esportati da un modulo consente due possibilità: ad esso di riferirsi ad un nodo definito al suo interno (il nodo passato come argomento del nodo importato quando il modulo viene istanziato); in modo simile, l'ambiente esterno in cui il modulo viene istanziato può riferirsi a nodi dell'istanziazione del modulo (quelli esportati), senza sapere come essi vengono implementati internamente.

Nella figura seguente viene presentato un modello semplificato dei concetti di base del PNML modulare.

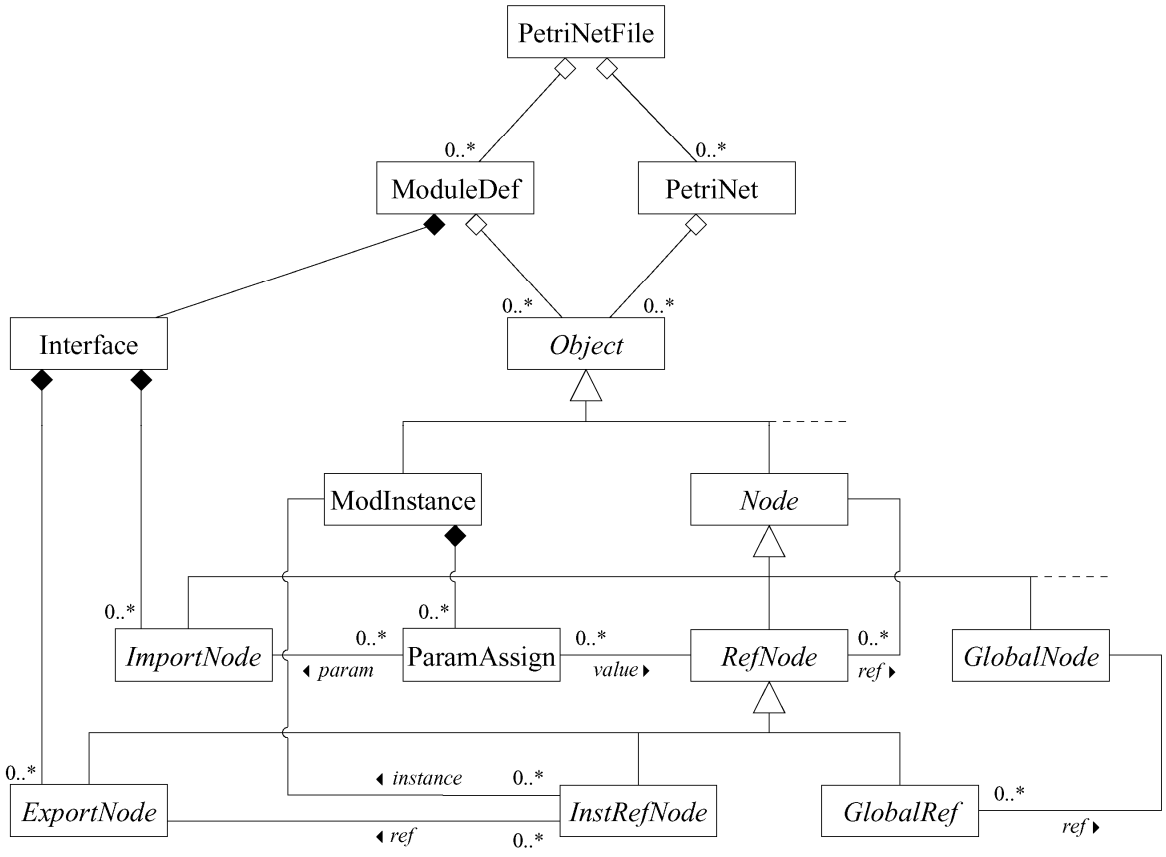


Fig. 19. Modello del PNML modulare

5.7.2 SINTASSI XML

In Fig. 20 vediamo come è possibile definire il modulo `M1` in un documento PNML con le estensioni per il PNML modulare.

```
<module name="M1">
     <interface>
          <importPlace id="p1"/>
          <exportPlace id="p2" ref="y"/>
     </interface>
     <referencePlace id="x" ref="p1"/>
     <transition id="t1"/>
     <transition id="t2"/>
     <place id="y"/>
     <arc source="x" target="t1"/>
     <arc source="t1" target="y"/>
     <arc source="y" target="t2"/>
     <arc source="t2" target="x"/>
</module>
```

Fig. 20. PNML modulare relativo al modulo `M1`

Nella prima parte viene definita l'interfaccia del modulo, indicando i nodi da importare ed esportare all'esterno, segue poi l'implementazione interna. Si noti l'utilizzo dei tag `ref`: nella parte d'interfaccia gli oggetti esportati puntano ad oggetti dell'implementazione, mentre nella parte d'implementazione i nodi riferimento possono puntare agli oggetti importati ma non a quelli esportati.

Se una rete `PN` o un modulo `M1` contiene un'istanza di un modulo `M2` si dice che `PN` (o `M1`) *usa* `M2`. Per usare un modulo esiste l'elemento PNML `<instance>`, che contiene i parametri di istanziazione del modulo. In Fig. 21 vediamo la PN `n1` in cui vengono utilizzati i moduli e in Fig. 22 il corrispondente frammento PNML che la descrive.

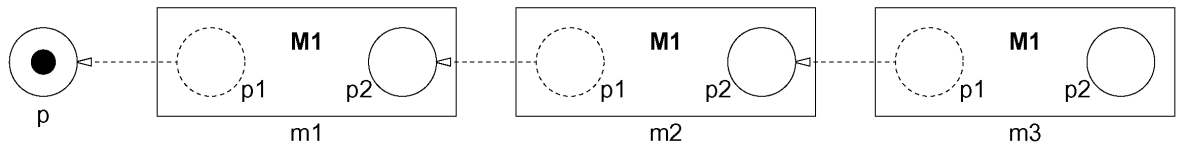


Fig. 21. Rete `n1`

```
<net id="n1">
     <place id="p">
          <initialMarking>
               <value>1</value>
          </initialMarking>
     </place>
     <instance id="m1" ref=URI#M1>
          <importPlace parameter="p1" ref="p"/>
     </instance>
     <instance id="m2" ref=URI#M1>
          <importPlace parameter="p1" instance="m1" ref="p2"/>
     </instance>
     <instance id="m3" ref=URI#M1>
          <importPlace parameter="p1" instance="m2" ref="p2"/>
     </instance>
</net>
```

Fig. 22. PNML modulare della rete `n1`

La rete `n1` contiene un posto `p` con una marca iniziale e tre istanze del modulo `M1`. Il posto `p` viene usato come parametro del posto `p1` nell'istanza `m1`. L'istanza `m2` utilizza invece come parametro di `p1` il posto esportato `p2` dall'istanza `m1`, e similarmente lavora l'istanza `m3`.

La rete `n1` sviluppata in cui sono risolte tutte le istanziazioni dei moduli utilizzati è chiamata *semantica* di n1. Una sua rappresentazione è visibile in Fig. 23.

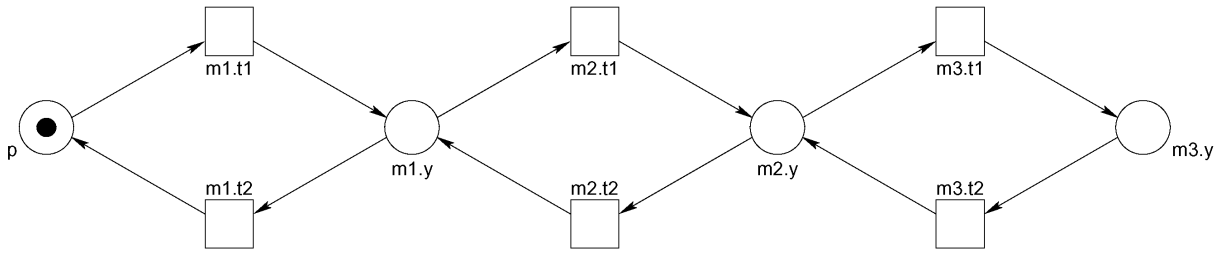


Fig. 23. Semantica della rete `n1`

Un ulteriore esempio che utilizza il modulo `M1` è rappresentato dalla PN in Fig. 24, dal frammento PNML che la descrive (Fig. 25) e dalla sua semantica (Fig. 26). In questo caso ogni istanza di `M1` riceve come parametro del posto importato `p1` il posto esportato `p2` di un'altra istanza.

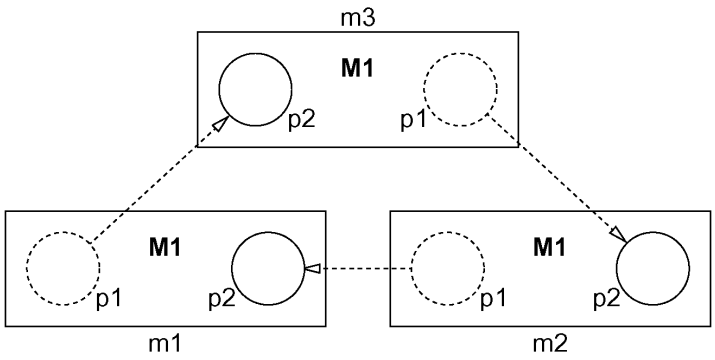


Fig. 24. PN modulare n2

```
<net id="n2">
     <instance id="m1" ref=URI#M1>
          <importPlace parameter="p1" instance="m3" ref="p2"/>
     </instance>
     <instance id="m2" ref=URI#M1>
          <importPlace parameter="p1" instance="m1" ref="p2"/>
     </instance>
     <instance id="m3" ref=URI#M1>
          <importPlace parameter="p1" instance="m2" ref="p2"/>
     </instance>
</net>
```

Fig. 25. PNML modulare della rete n2

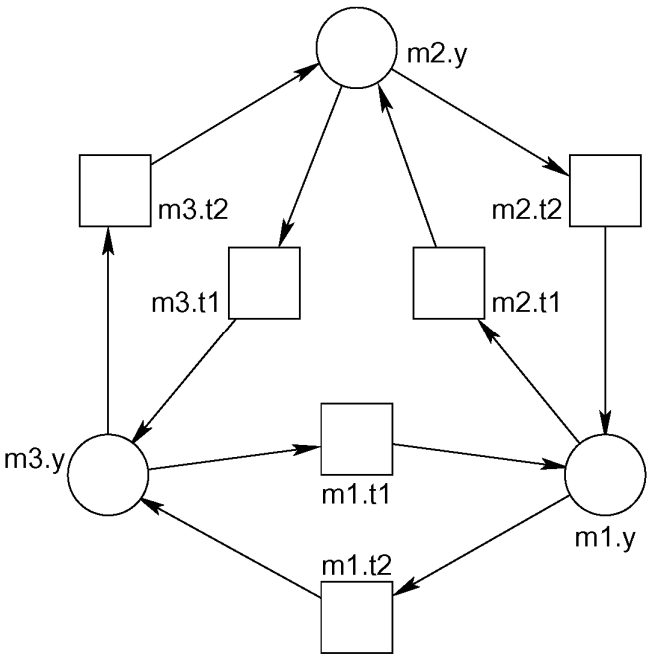


Fig. 26. Semantica della rete n2

5.7.3 SEMANTICA

Il processo che sta alla base della semantica del PNML modulare è di tipo bottom-up, ed è possibile in quanto viene richiesto in fase di progettazione che non ci siano riferimenti ciclici nella costruzione di un documento che descrive una PN.

Partendo dai moduli che non usano istanze di altri moduli (*moduli base*), viene sostituita ad una loro istanza una copia della loro implementazione[11]. Per preservare la struttura modulare del modello ogni implementazione di ogni istanza viene definita in una pagina separata che sarà nominata col nome dell'istanza stessa. Questa sostituzione viene chiamata *inlining*.

Ricorsivamente poi il processo dell'inlining viene applicato ad ogni istanza che contenga altre istanze di moduli, consentendo così in conclusione la traduzione da PNML modulare a PNML strutturato.

[11] Per non creare conflitti sui nomi di moduli istanziati più volte, ad ogni identificatore degli oggetti contenuti nell'implementazione del modulo viene posto come prefisso il nome dell'istanza seguito da un punto. Questo non crea problemi in quanto una regola del PNML modulare è infatti di non utilizzare il punto nel nome delle istanze.

CAPITOLO 6

ESEMPI APPLICATIVI

6.1 INTRODUZIONE

In questo capitolo verranno presentati degli esempi di utilizzo di ScoreSynth. Essendo il formato MX ancora un work-in-progress, non abbondano esempi di questo tipo di documenti musicali. Per questo i frammenti musicali che seguiranno saranno di modeste dimensioni, ma efficaci comunque per esemplificare il funzionamento del software.

6.2 I 5 BATTERISTI

Il *problema dei 5 filosofi* è stato ideato da E. W. Dijkstra, ed è un esempio rappresentativo dei problemi di sincronizzazione tra processi, che hanno a disposizione delle risorse condivise limitate.

Il problema, riadattato all'ambito musicale, verrà chiamato *dei 5 batteristi*, e verrà utilizzando per illustrare alcune nuove caratteristiche di ScoreSynth, senza associare alcun documento MX ai nodi della PN.

Il problema è il seguente: vi sono 5 batteristi seduti in cerchio, ed ognuno ha davanti a sé un tamburo. Fra i tamburi sono disposte 5 bacchette (non 10, che permetterebbero a tutti i batteristi di suonare contemporaneamente). Il compositore ha assegnato ad ogni batterista un libro di patterns ritmici, ed egli ha 2 possibilità: scegliere un pattern ritmico da eseguire e suonare il pattern ritmico scelto. Il compositore ha disposto anche che i batteristi spendano un tempo finito a loro scelta per decidere il pattern preferito, e che, una volta scelto, venga eseguito un numero finito di volte deciso dal suonatore. Ogni volta il singolo batterista dovrà scegliere un pattern diverso dal precedente. Il problema che si presenta è che quando un batterista ha deciso di suonare deve impugnare due bacchette per farlo, una alla sua destra ed una alla sua sinistra, che non sempre sono libere, visto che sono condivise dai colleghi al suo

fianco. Finché allora entrambe le bacchette di un batterista non risultano libere, egli aspetterà il suo turno. Naturalmente non potrà mai accadere che due batteristi vicini suonino insieme. Il problema richiede che si trovi una soluzione che consenta di:

- Non far rimanere tutti i batteristi bloccati indefinitamente in attesa reciproca che l'altro rilasci l'ulteriore bacchetta che hanno bisogno per suonare (deadlock)
- Non far rimanere uno o più batteristi senza mai suonare perché gli altri sono più svelti di loro a prendere le bacchette (starvation)

Le Reti di Petri consentono di ottenere una soluzione al problema esposto.

Questo esempio mette alla luce alcune caratteristiche dello ScoreSynth "nuova generazione", rispetto alla vecchia applicazione di A. Sametti.

La PN che modella il problema dei 5 batteristi, disegnata in ScoreSynth 3.0 "vecchia generazione" è presentata in Fig. 27, mentre disegnata in ScoreSynth "nuova generazione" è visibile in Fig. 28.

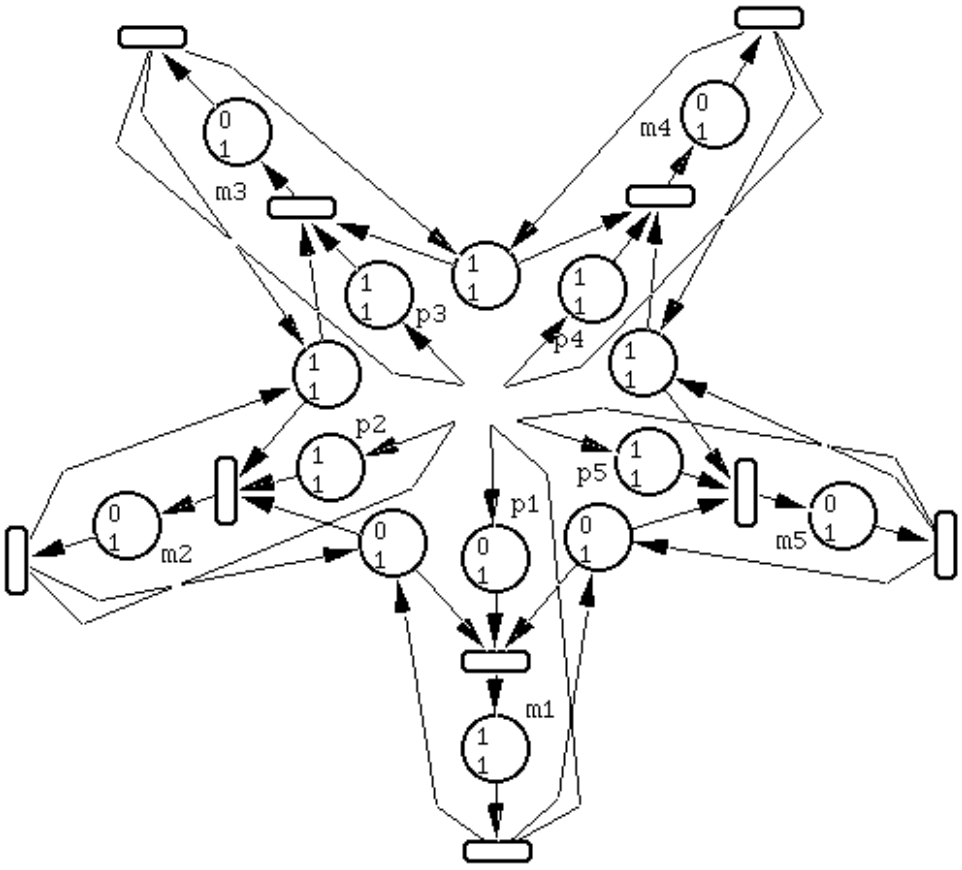


Fig. 27. Il problema dei 5 batteristi (1)

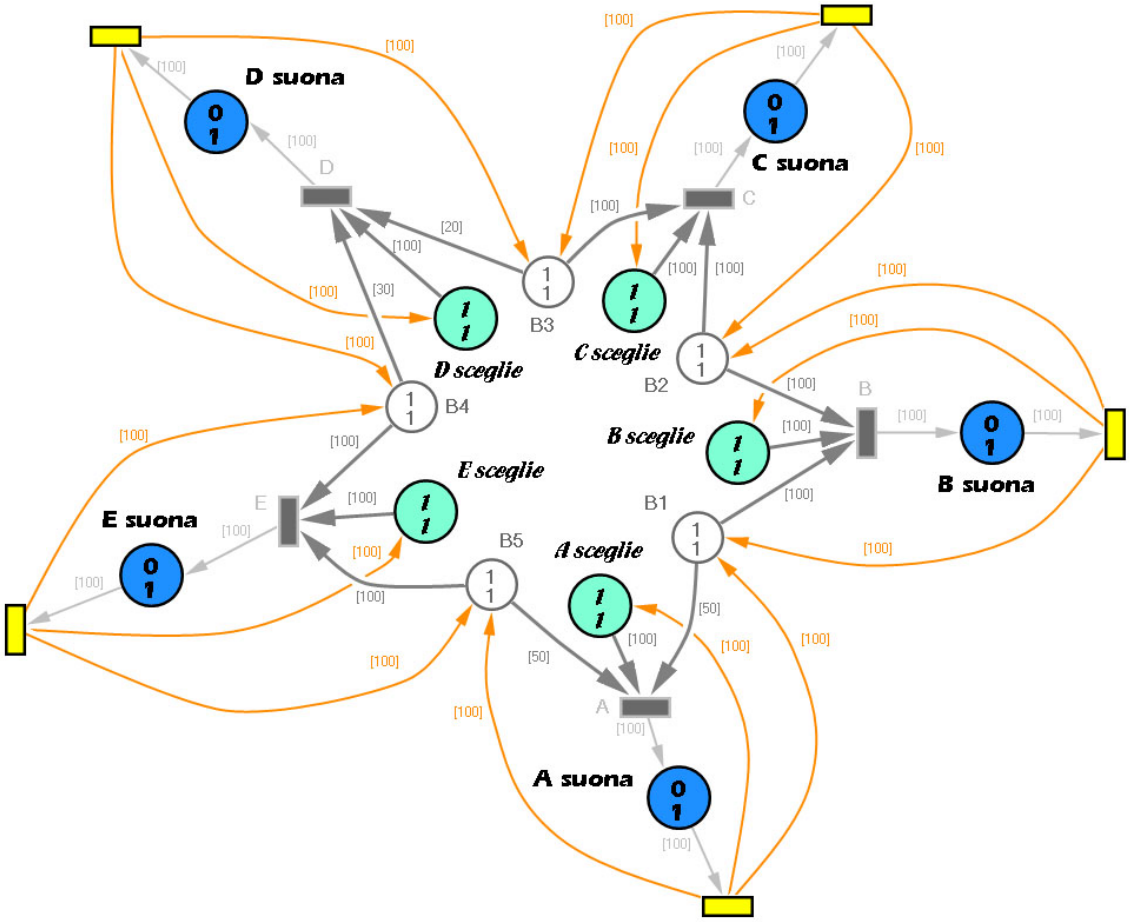


Fig. 28. Il problema dei 5 batteristi (2)

Una delle caratteristiche più evidenti è la possibilità di attribuire dei parametri di visualizzazione diversi agli elementi della rete, per consentire visivamente una maggiore chiarezza del modello.

Una caratteristica più importante e sostanziale del nuovo modello è che è possibile associare agli archi un peso probabilistico (indicato tra parentesi quadre) che entra in gioco in situazioni di alternativa/conflitto. Nel nostro esempio gli archi uscenti dalle bacchette **B1** e **B5** hanno pesi diversi: gli archi che collegano i posti rappresentanti le bacchette alla transizione **A** hanno peso probabilistico corrispondente alla metà del peso degli archi che collegano le bacchette alle altre due transizioni **B** ed **E**. Questo nel nostro modello significa che le transizioni che portano un batterista dallo stato di scelta del pattern al suonare non hanno la stessa probabilità di scattare.

Vediamo quali sono le probabilità di scelta allo stato iniziale della rete, mostrato in figura. Quando si verifica una situazione in alternativa ScoreSynth crea una lista contenente tutti gli

archi che hanno come nodo finale le transizioni in alternativa[12], e dalla lista viene effettuata una scelta pesata dipendente dal peso probabilistico degli archi. Quindi viene fatta scattare la transizione che ha in ingresso l'arco scelto.

Prendiamo ad esempio nel nostro modello la transizione **A**: essa verrà fatta scattare se viene scelto l'arco che la collega con **B1** oppure se viene scelto l'arco che la collega con **B5**. Il peso probabilistico globale della transizione risulta allora uguale alla somma dei suoi due archi, ovvero 50+50=100. Ragionando in modo analogo vediamo che la tabella dei pesi probabilistici associati agli scatti delle transizioni risulta essere:

Transizioni	*Pesi degli archi entranti*	*Peso totale*	*Probabilità*
A	50 + 50	100	13,3 %
B	100 + 100	200	26,7 %
C	100 + 100	200	26,7 %
D	20 + 30	50	6,7 %
E	100 + 100	200	26,7 %

Dalla tabella si deduce che la transizione **A** verrà eseguita la metà delle volte rispetto alle transizioni **B**, **C** ed **E**, mentre **D** verrà eseguita la metà delle volte rispetto ad **A**. Nell'ultima colonna vengono illustrate le probabilità di scatto.

Per come è stato presentato, il modello non rappresenta efficacemente il problema esposto, visto che ai posti associati alla scelta del pattern ed alla sua esecuzione non è associato nessun oggetto MX. Per questo il tempo d'esecuzione dei posti è nullo e ad ogni passaggio di marche ai posti **# suona**, segue subito il ritorno allo stato iniziale.

Per simulare opportunamente il problema basterebbe disporre di *n* frammenti MX di diversa lunghezza ed associare ai posti **#sceglie** e **#suona** una sottorete del tipo rappresentato in Fig. 29, associando i frammenti MX ai posti **MX*n***. In questo modo anche le probabilità di scatto delle transizioni cambierebbero, visto che non sempre tutti i posti **B*n*** conterrebbero una marca; *in generale le probabilità dipendono dallo stato della rete*.

[12] Nel caso di conflitto, la lista contiene tutti gli archi che **partono** dalle transizioni in conflitto.

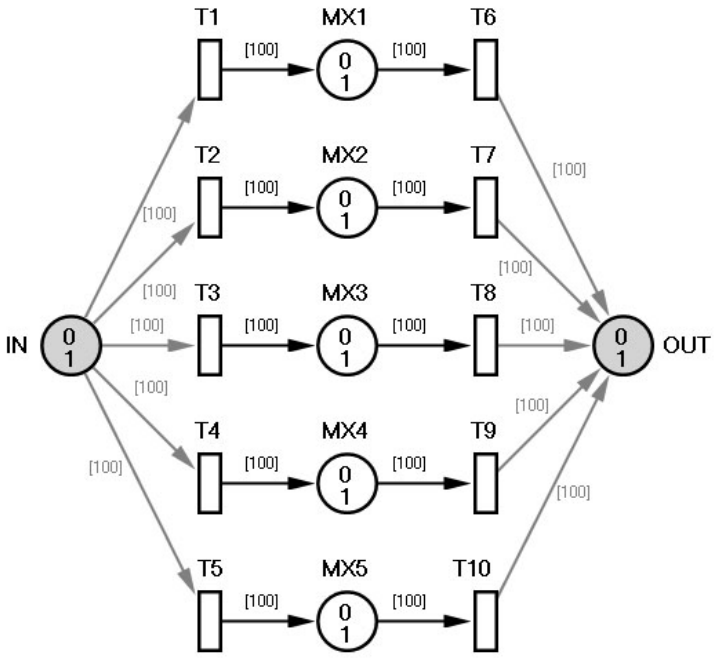


Fig. 29. Scelta casuale tra 5 frammenti MX

6.3 FRA MARTINO

Il secondo esempio è simile a quello usato nella tesi di A. Sametti su ScoreSynth [Sam88]. La PN che modella una semplice struttura musicale a canone è rappresentata in Fig. 30.

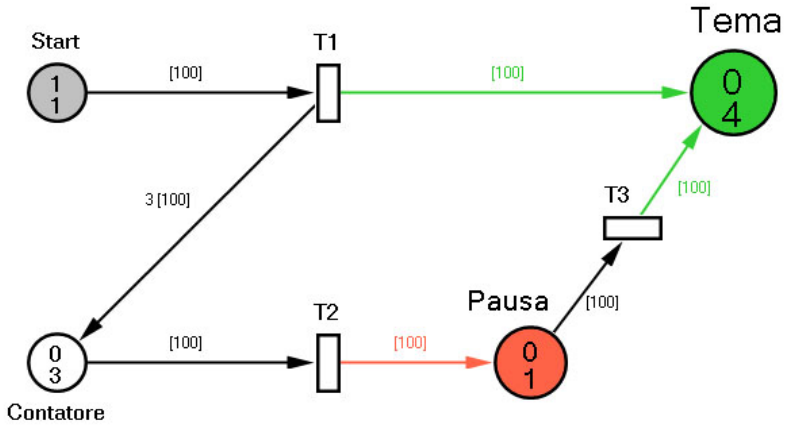


Fig. 30. Esempio di PN di un canone

Al posto **Pausa** viene associata una pausa di 2 battute, mentre il **Tema** è costituito da 8 battute ed è raffigurato in Fig. 31.

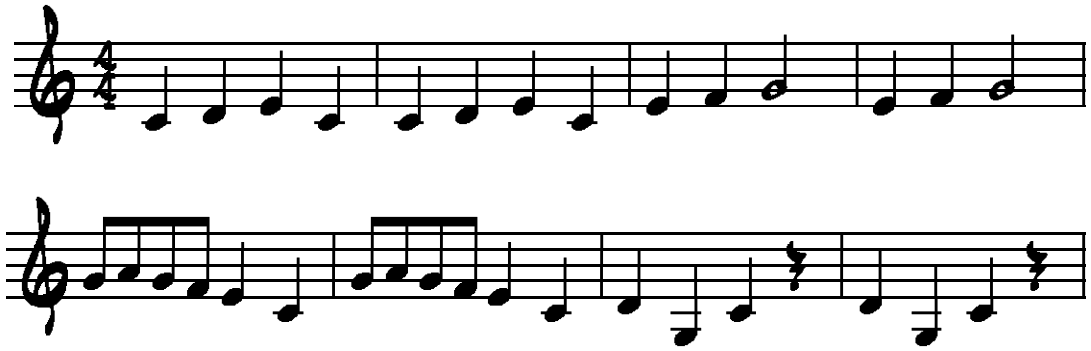


Fig. 31. Il tema di Fra Martino

Il file MX associato al posto **Pausa** è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mx SYSTEM "MX.dtd">
<mx>
   <general>
      <description>
         <movement_title/>
         <author/>
      </description>
   </general>
   <logic>
      <spine>
         <event id="clef_0" hpos="4"/>
         <event id="timesig_0" hpos="9"/>
         <event id="rest_0" timing="0" hpos="9"/>
         <event id="barline_0" hpos="10"/>
         <event id="rest_1" timing="64" hpos="9"/>
         <event id="barline_1" hpos="10"/>
      </spine>
      <los>
         <staff_list>
            <staff id="staff0">
               <clef type="G" event_ref="clef_0" staff_step="2"/>
               <time_signature num="4" den="4" event_ref="timesig_0"
               vtu_amount="64"/>
               <barline id="b_1" event_ref="barline_0"/>
               <barline id="b_2" event_ref="barline_1"/>
            </staff>
```

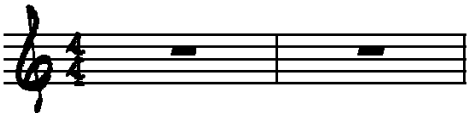


Fig. 32. Pausa di 2 battute

```
        </staff_list>
        <part id="part1" dfstaff_ref="staff0">
          <voice_list>
            <voice_item id="voice1"/>
          </voice_list>
          <measure number="1">
            <voice ref="voice1">
              <rest staff_ref="staff0" event_ref="rest_0">
                <duration num="4" den="4"/>
              </rest>
            </voice>
          </measure>
          <measure number="2">
            <voice ref="voice1">
              <rest staff_ref="staff0" event_ref="rest_1">
                <duration num="4" den="4"/>
              </rest>
            </voice>
          </measure>
        </part>
        <horizontal_symbols/>
      </los>
    </logic>
   </mx>
```

Fig. 33. Frammento MX associato al posto **Pausa**

Il file MX che rappresenta il tema è invece il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mx SYSTEM "MX.dtd">
<mx>
  <general>
    <description>
      <movement_title>
        Fra Martino
      </movement_title>
      <author/>
    </description>
  </general>
```

```
<logic>
  <spine>
    <event id="clef_0" hpos="4"/>
    <event id="timesig_0" hpos="9"/>
    <event id="e0_0" timing="0" hpos="9"/>
    <event id="e0_1" timing="16" hpos="9"/>
    <event id="e0_2" timing="16" hpos="9"/>
    <event id="e0_3" timing="16" hpos="9"/>
    <event id="barline_0" hpos="10"/>
    <event id="e0_4" timing="16" hpos="9"/>
    <event id="e0_5" timing="16" hpos="9"/>
    <event id="e0_6" timing="16" hpos="9"/>
    <event id="e0_7" timing="16" hpos="9"/>
    <event id="barline_1" hpos="10"/>
    <event id="e0_8" timing="16" hpos="9"/>
    <event id="e0_9" timing="16" hpos="9"/>
    <event id="e0_10" timing="16" hpos="9"/>
    <event id="barline_2" hpos="10"/>
    <event id="e0_11" timing="32" hpos="9"/>
    <event id="e0_12" timing="16" hpos="9"/>
    <event id="e0_13" timing="16" hpos="9"/>
    <event id="barline_3" hpos="10"/>
    <event id="e0_14" timing="32" hpos="9"/>
    <event id="e0_15" timing="8" hpos="9"/>
    <event id="e0_16" timing="8" hpos="9"/>
    <event id="e0_17" timing="8" hpos="9"/>
    <event id="e0_18" timing="8" hpos="9"/>
    <event id="e0_19" timing="16" hpos="9"/>
    <event id="barline_4" hpos="10"/>
    <event id="e0_20" timing="16" hpos="9"/>
    <event id="e0_21" timing="8" hpos="9"/>
    <event id="e0_22" timing="8" hpos="9"/>
    <event id="e0_23" timing="8" hpos="9"/>
    <event id="e0_24" timing="8" hpos="9"/>
    <event id="e0_25" timing="16" hpos="9"/>
    <event id="barline_5" hpos="10"/>
    <event id="e0_26" timing="16" hpos="9"/>
    <event id="e0_27" timing="16" hpos="9"/>
    <event id="e0_28" timing="16" hpos="9"/>
    <event id="rest_0" timing="16" hpos="9"/>
```

```
      <event id="barline_6" hpos="10"/>
      <event id="e0_29" timing="16" hpos="9"/>
      <event id="e0_30" timing="16" hpos="9"/>
      <event id="e0_31" timing="16" hpos="9"/>
      <event id="rest_1" timing="16" hpos="9"/>
      <event id="barline_7" hpos="10"/>
   </spine>
   <los>
      <staff_list>
         <staff id="staff0">
            <clef type="G" event_ref="clef_0" staff_step="2"/>
            <time_signature num="4" den="4" event_ref="timesig_0"
            vtu_amount="64"/>
            <barline id="b_1" event_ref="barline_0"/>
            <barline id="b_2" event_ref="barline_1"/>
            <barline id="b_3" event_ref="barline_2"/>
            <barline id="b_4" event_ref="barline_3"/>
            <barline id="b_5" event_ref="barline_4"/>
            <barline id="b_6" event_ref="barline_5"/>
            <barline id="b_7" event_ref="barline_6"/>
            <barline id="b_8" event_ref="barline_7"/>
         </staff>
      </staff_list>
      <part id="part1" dfstaff_ref="staff0">
         <voice_list>
            <voice_item id="voice1"/>
         </voice_list>
         <measure number="1">
            <voice ref="voice1">
               <chord event_ref="e0_0">
                  <notehead staff_ref="staff0">
                     <pitch_def staff_step="-2"/>
                     <duration den="4" num="1"/>
                  </notehead>
               </chord>
               <chord event_ref="e0_1">
                  <notehead staff_ref="staff0">
                     <pitch_def staff_step="-1"/>
                     <duration den="4" num="1"/>
                  </notehead>
```

```
            </chord>
            <chord event_ref="e0_2">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="0"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="e0_3">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
         </voice>
      </measure>
      <measure number="2">
         <voice ref="voice1">
            <chord event_ref="e0_4">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="e0_5">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="e0_6">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="7"/>
                  <duration den="0" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="e0_7">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="5"/>
                  <duration den="-2" num="1"/>
               </notehead>
```

```
      </chord>
    </voice>
  </measure>
  <measure number="3">
    <voice ref="voice1">
      <chord event_ref="e0_8">
        <notehead staff_ref="staff0">
          <pitch_def staff_step="3"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="e0_9">
        <notehead staff_ref="staff0">
          <pitch_def staff_step="4"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="e0_10">
        <notehead staff_ref="staff0">
          <pitch_def staff_step="5"/>
          <duration den="4" num="2"/>
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="4">
    <voice ref="voice1">
      <chord event_ref="e0_11">
        <notehead staff_ref="staff0">
          <pitch_def staff_step="3"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="e0_12">
        <notehead staff_ref="staff0">
          <pitch_def staff_step="4"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="e0_13">
```

```
            <notehead staff_ref="staff0">
              <pitch_def staff_step="5"/>
              <duration den="4" num="2"/>
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="5">
        <voice ref="voice1">
          <chord event_ref="e0_14">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="2"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_15">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="3"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_16">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="2"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_17">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="1"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_18">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="0"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_19">
```

```
            <notehead staff_ref="staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="6">
        <voice ref="voice1">
          <chord event_ref="e0_20">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="2"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_21">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="3"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_22">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="2"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_23">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="1"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_24">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="0"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_25">
```

```
            <notehead staff_ref="staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="7">
        <voice ref="voice1">
          <chord event_ref="e0_26">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="-1"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_27">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="3"/>
              <duration den="-5" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="e0_28">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <rest staff_ref="staff0" event_ref="rest_0">
            <duration den="4" num="1"/>
          </rest>
        </voice>
      </measure>
      <measure number="8">
        <voice ref="voice1">
          <chord event_ref="e0_29">
            <notehead staff_ref="staff0">
              <pitch_def staff_step="-1"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
```

```
            <chord event_ref="e0_30">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="-5" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="e0_31">
               <notehead staff_ref="staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <rest staff_ref="staff0" event_ref="rest_1">
               <duration den="4" num="1"/>
            </rest>
         </voice>
      </measure>
   </part>
   <horizontal_symbols/>
  </los>
 </logic>
</mx>
```

Fig. 34. Frammento MX associato al posto **Tema**

Come è possibile notare, i files MX usati in questo esempio codificano soltanto il livello **logic** del formato MX. Questo non compromette la rappresentatività dell'esempio, visto che gli elementi XML degli altri livelli vengono solamente copiati nel file di output senza modifiche[13].

L'esecuzione della PN provoca nell'istante 0 l'arrivo di una marca nei posti **Tema** e **Pausa**; quando la pausa di 2 battute è stata elaborata, la sua marca viene passata al posto **Tema**, che inizia una nuova esposizione, sfasata di 2 battute rispetto alla precedente.

Il procedimento continua fino alla fine delle 3 marche contenute nel posto **Contatore**, esponendo 4 temi sfasati ognuno di 2 battute.

Durante l'esecuzione della PN ScoreSynth produce la timeline visibile nella figura seguente.

[13] In effetti vengono modificati i puntatori alla spine, ma semplicemente anteponendovi la stringa **mx#_**, dove # è il numero del file MX processato (come succede per gli altri elementi del file di output visibile più avanti).

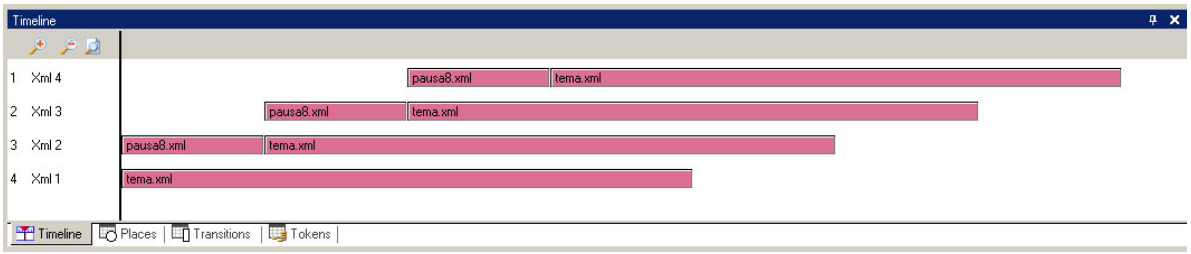


Fig. 35. Timeline dell'esempio Fra Martino

L'output dell'esecuzione della PN è il file MX seguente:

```
<!DOCTYPE mx SYSTEM "mx.dtd">
<mx>
  <general>
    <description>
      <movement_title>Fra Martino</movement_title>
      <author/>
    </description>
  </general>
  <logic>
    <spine>
      <event id="mx0_clef_0" hpos="4" timing="0"/>
      <event id="mx0_timesig_0" hpos="9" timing="0"/>
      <event id="mx0_e0_0" hpos="9" timing="0"/>
      <event id="mx1_clef_0" hpos="4" timing="0"/>
      <event id="mx1_timesig_0" hpos="9" timing="0"/>
      <event id="mx1_rest_0" hpos="9" timing="0"/>
      <event id="mx1_barline_0" hpos="10" timing="0"/>
      <event id="mx0_e0_1" hpos="9" timing="16"/>
      <event id="mx0_e0_2" hpos="9" timing="16"/>
      <event id="mx0_e0_3" hpos="9" timing="16"/>
      <event id="mx0_barline_0" hpos="10" timing="0"/>
      <event id="mx0_e0_4" hpos="9" timing="16"/>
      <event id="mx1_rest_1" hpos="9" timing="0"/>
      <event id="mx1_barline_1" hpos="10" timing="0"/>
      <event id="mx0_e0_5" hpos="9" timing="16"/>
      <event id="mx0_e0_6" hpos="9" timing="16"/>
      <event id="mx0_e0_7" hpos="9" timing="16"/>
```

```
<event id="mx0_barline_1" hpos="10" timing="0"/>
<event id="mx0_e0_8" hpos="9" timing="16"/>
<event id="mx2_clef_0" hpos="4" timing="0"/>
<event id="mx2_timesig_0" hpos="9" timing="0"/>
<event id="mx2_e0_0" hpos="9" timing="0"/>
<event id="mx3_clef_0" hpos="4" timing="0"/>
<event id="mx3_timesig_0" hpos="9" timing="0"/>
<event id="mx3_rest_0" hpos="9" timing="0"/>
<event id="mx3_barline_0" hpos="10" timing="0"/>
<event id="mx0_e0_9" hpos="9" timing="16"/>
<event id="mx2_e0_1" hpos="9" timing="0"/>
<event id="mx0_e0_10" hpos="9" timing="16"/>
<event id="mx0_barline_2" hpos="10" timing="0"/>
<event id="mx2_e0_2" hpos="9" timing="0"/>
<event id="mx2_e0_3" hpos="9" timing="16"/>
<event id="mx2_barline_0" hpos="10" timing="0"/>
<event id="mx0_e0_11" hpos="9" timing="16"/>
<event id="mx2_e0_4" hpos="9" timing="0"/>
<event id="mx3_rest_1" hpos="9" timing="0"/>
<event id="mx3_barline_1" hpos="10" timing="0"/>
<event id="mx0_e0_12" hpos="9" timing="16"/>
<event id="mx2_e0_5" hpos="9" timing="0"/>
<event id="mx0_e0_13" hpos="9" timing="16"/>
<event id="mx0_barline_3" hpos="10" timing="0"/>
<event id="mx2_e0_6" hpos="9" timing="0"/>
<event id="mx2_e0_7" hpos="9" timing="16"/>
<event id="mx2_barline_1" hpos="10" timing="0"/>
<event id="mx0_e0_14" hpos="9" timing="16"/>
<event id="mx2_e0_8" hpos="9" timing="0"/>
<event id="mx4_clef_0" hpos="4" timing="0"/>
<event id="mx4_timesig_0" hpos="9" timing="0"/>
<event id="mx4_e0_0" hpos="9" timing="0"/>
<event id="mx5_clef_0" hpos="4" timing="0"/>
<event id="mx5_timesig_0" hpos="9" timing="0"/>
<event id="mx5_rest_0" hpos="9" timing="0"/>
<event id="mx5_barline_0" hpos="10" timing="0"/>
<event id="mx0_e0_15" hpos="9" timing="8"/>
<event id="mx0_e0_16" hpos="9" timing="8"/>
<event id="mx2_e0_9" hpos="9" timing="0"/>
<event id="mx4_e0_1" hpos="9" timing="0"/>
```

```
<event id="mx0_e0_17" hpos="9" timing="8"/>
<event id="mx0_e0_18" hpos="9" timing="8"/>
<event id="mx2_e0_10" hpos="9" timing="0"/>
<event id="mx2_barline_2" hpos="10" timing="0"/>
<event id="mx4_e0_2" hpos="9" timing="0"/>
<event id="mx0_e0_19" hpos="9" timing="16"/>
<event id="mx0_barline_4" hpos="10" timing="0"/>
<event id="mx4_e0_3" hpos="9" timing="0"/>
<event id="mx4_barline_0" hpos="10" timing="0"/>
<event id="mx0_e0_20" hpos="9" timing="16"/>
<event id="mx2_e0_11" hpos="9" timing="0"/>
<event id="mx4_e0_4" hpos="9" timing="0"/>
<event id="mx5_rest_1" hpos="9" timing="0"/>
<event id="mx5_barline_1" hpos="10" timing="0"/>
<event id="mx0_e0_21" hpos="9" timing="8"/>
<event id="mx0_e0_22" hpos="9" timing="8"/>
<event id="mx2_e0_12" hpos="9" timing="0"/>
<event id="mx4_e0_5" hpos="9" timing="0"/>
<event id="mx0_e0_23" hpos="9" timing="8"/>
<event id="mx0_e0_24" hpos="9" timing="8"/>
<event id="mx2_e0_13" hpos="9" timing="0"/>
<event id="mx2_barline_3" hpos="10" timing="0"/>
<event id="mx4_e0_6" hpos="9" timing="0"/>
<event id="mx0_e0_25" hpos="9" timing="16"/>
<event id="mx0_barline_5" hpos="10" timing="0"/>
<event id="mx4_e0_7" hpos="9" timing="0"/>
<event id="mx4_barline_1" hpos="10" timing="0"/>
<event id="mx0_e0_26" hpos="9" timing="16"/>
<event id="mx2_e0_14" hpos="9" timing="0"/>
<event id="mx4_e0_8" hpos="9" timing="0"/>
<event id="mx6_clef_0" hpos="4" timing="0"/>
<event id="mx6_timesig_0" hpos="9" timing="0"/>
<event id="mx6_e0_0" hpos="9" timing="0"/>
<event id="mx2_e0_15" hpos="9" timing="8"/>
<event id="mx0_e0_27" hpos="9" timing="8"/>
<event id="mx2_e0_16" hpos="9" timing="0"/>
<event id="mx4_e0_9" hpos="9" timing="0"/>
<event id="mx6_e0_1" hpos="9" timing="0"/>
<event id="mx2_e0_17" hpos="9" timing="8"/>
<event id="mx0_e0_28" hpos="9" timing="8"/>
```

```
<event id="mx2_e0_18" hpos="9" timing="0"/>
<event id="mx4_e0_10" hpos="9" timing="0"/>
<event id="mx4_barline_2" hpos="10" timing="0"/>
<event id="mx6_e0_2" hpos="9" timing="0"/>
<event id="mx0_rest_0" hpos="9" timing="16"/>
<event id="mx0_barline_6" hpos="10" timing="0"/>
<event id="mx2_e0_19" hpos="9" timing="0"/>
<event id="mx2_barline_4" hpos="10" timing="0"/>
<event id="mx6_e0_3" hpos="9" timing="0"/>
<event id="mx6_barline_0" hpos="10" timing="0"/>
<event id="mx0_e0_29" hpos="9" timing="16"/>
<event id="mx2_e0_20" hpos="9" timing="0"/>
<event id="mx4_e0_11" hpos="9" timing="0"/>
<event id="mx6_e0_4" hpos="9" timing="0"/>
<event id="mx2_e0_21" hpos="9" timing="8"/>
<event id="mx0_e0_30" hpos="9" timing="8"/>
<event id="mx2_e0_22" hpos="9" timing="0"/>
<event id="mx4_e0_12" hpos="9" timing="0"/>
<event id="mx6_e0_5" hpos="9" timing="0"/>
<event id="mx2_e0_23" hpos="9" timing="8"/>
<event id="mx0_e0_31" hpos="9" timing="8"/>
<event id="mx2_e0_24" hpos="9" timing="0"/>
<event id="mx4_e0_13" hpos="9" timing="0"/>
<event id="mx4_barline_3" hpos="10" timing="0"/>
<event id="mx6_e0_6" hpos="9" timing="0"/>
<event id="mx0_rest_1" hpos="9" timing="16"/>
<event id="mx0_barline_7" hpos="10" timing="0"/>
<event id="mx2_e0_25" hpos="9" timing="0"/>
<event id="mx2_barline_5" hpos="10" timing="0"/>
<event id="mx6_e0_7" hpos="9" timing="0"/>
<event id="mx6_barline_1" hpos="10" timing="0"/>
<event id="mx2_e0_26" hpos="9" timing="16"/>
<event id="mx4_e0_14" hpos="9" timing="0"/>
<event id="mx6_e0_8" hpos="9" timing="0"/>
<event id="mx4_e0_15" hpos="9" timing="8"/>
<event id="mx2_e0_27" hpos="9" timing="8"/>
<event id="mx4_e0_16" hpos="9" timing="0"/>
<event id="mx6_e0_9" hpos="9" timing="0"/>
<event id="mx4_e0_17" hpos="9" timing="8"/>
<event id="mx2_e0_28" hpos="9" timing="8"/>
```

```
<event id="mx4_e0_18" hpos="9" timing="0"/>
<event id="mx6_e0_10" hpos="9" timing="0"/>
<event id="mx6_barline_2" hpos="10" timing="0"/>
<event id="mx2_rest_0" hpos="9" timing="16"/>
<event id="mx2_barline_6" hpos="10" timing="0"/>
<event id="mx4_e0_19" hpos="9" timing="0"/>
<event id="mx4_barline_4" hpos="10" timing="0"/>
<event id="mx2_e0_29" hpos="9" timing="16"/>
<event id="mx4_e0_20" hpos="9" timing="0"/>
<event id="mx6_e0_11" hpos="9" timing="0"/>
<event id="mx4_e0_21" hpos="9" timing="8"/>
<event id="mx2_e0_30" hpos="9" timing="8"/>
<event id="mx4_e0_22" hpos="9" timing="0"/>
<event id="mx6_e0_12" hpos="9" timing="0"/>
<event id="mx4_e0_23" hpos="9" timing="8"/>
<event id="mx2_e0_31" hpos="9" timing="8"/>
<event id="mx4_e0_24" hpos="9" timing="0"/>
<event id="mx6_e0_13" hpos="9" timing="0"/>
<event id="mx6_barline_3" hpos="10" timing="0"/>
<event id="mx2_rest_1" hpos="9" timing="16"/>
<event id="mx2_barline_7" hpos="10" timing="0"/>
<event id="mx4_e0_25" hpos="9" timing="0"/>
<event id="mx4_barline_5" hpos="10" timing="0"/>
<event id="mx4_e0_26" hpos="9" timing="16"/>
<event id="mx6_e0_14" hpos="9" timing="0"/>
<event id="mx6_e0_15" hpos="9" timing="8"/>
<event id="mx4_e0_27" hpos="9" timing="8"/>
<event id="mx6_e0_16" hpos="9" timing="0"/>
<event id="mx6_e0_17" hpos="9" timing="8"/>
<event id="mx4_e0_28" hpos="9" timing="8"/>
<event id="mx6_e0_18" hpos="9" timing="0"/>
<event id="mx4_rest_0" hpos="9" timing="16"/>
<event id="mx4_barline_6" hpos="10" timing="0"/>
<event id="mx6_e0_19" hpos="9" timing="0"/>
<event id="mx6_barline_4" hpos="10" timing="0"/>
<event id="mx4_e0_29" hpos="9" timing="16"/>
<event id="mx6_e0_20" hpos="9" timing="0"/>
<event id="mx6_e0_21" hpos="9" timing="8"/>
<event id="mx4_e0_30" hpos="9" timing="8"/>
<event id="mx6_e0_22" hpos="9" timing="0"/>
```

```
      <event id="mx6_e0_23" hpos="9" timing="8"/>
      <event id="mx4_e0_31" hpos="9" timing="8"/>
      <event id="mx6_e0_24" hpos="9" timing="0"/>
      <event id="mx4_rest_1" hpos="9" timing="16"/>
      <event id="mx4_barline_7" hpos="10" timing="0"/>
      <event id="mx6_e0_25" hpos="9" timing="0"/>
      <event id="mx6_barline_5" hpos="10" timing="0"/>
      <event id="mx6_e0_26" hpos="9" timing="16"/>
      <event id="mx6_e0_27" hpos="9" timing="16"/>
      <event id="mx6_e0_28" hpos="9" timing="16"/>
      <event id="mx6_rest_0" hpos="9" timing="16"/>
      <event id="mx6_barline_6" hpos="10" timing="0"/>
      <event id="mx6_e0_29" hpos="9" timing="16"/>
      <event id="mx6_e0_30" hpos="9" timing="16"/>
      <event id="mx6_e0_31" hpos="9" timing="16"/>
      <event id="mx6_rest_1" hpos="9" timing="16"/>
      <event id="mx6_barline_7" hpos="10" timing="0"/>
   </spine>
   <los>
      <staff_list>
         <staff id="mx0_staff0">
            <clef type="G" event_ref="mx0_clef_0" staff_step="2"/>
            <time_signature num="4" den="4" event_ref="mx0_timesig_0"
            vtu_amount="64"/>
            <barline id="mx0_b_1" event_ref="mx0_barline_0"/>
            <barline id="mx0_b_2" event_ref="mx0_barline_1"/>
            <barline id="mx0_b_3" event_ref="mx0_barline_2"/>
            <barline id="mx0_b_4" event_ref="mx0_barline_3"/>
            <barline id="mx0_b_5" event_ref="mx0_barline_4"/>
            <barline id="mx0_b_6" event_ref="mx0_barline_5"/>
            <barline id="mx0_b_7" event_ref="mx0_barline_6"/>
            <barline id="mx0_b_8" event_ref="mx0_barline_7"/>
         </staff>
         <staff id="mx1_staff0">
            <clef type="G" event_ref="mx1_clef_0" staff_step="2"/>
            <time_signature num="4" den="4" event_ref="mx1_timesig_0"
            vtu_amount="64"/>
            <barline id="mx1_b_1" event_ref="mx1_barline_0"/>
            <barline id="mx1_b_2" event_ref="mx1_barline_1"/>
         </staff>
```

```
<staff id="mx2_staff0">
  <clef type="G" event_ref="mx2_clef_0" staff_step="2"/>
  <time_signature num="4" den="4" event_ref="mx2_timesig_0"
  vtu_amount="64"/>
  <barline id="mx2_b_1" event_ref="mx2_barline_0"/>
  <barline id="mx2_b_2" event_ref="mx2_barline_1"/>
  <barline id="mx2_b_3" event_ref="mx2_barline_2"/>
  <barline id="mx2_b_4" event_ref="mx2_barline_3"/>
  <barline id="mx2_b_5" event_ref="mx2_barline_4"/>
  <barline id="mx2_b_6" event_ref="mx2_barline_5"/>
  <barline id="mx2_b_7" event_ref="mx2_barline_6"/>
  <barline id="mx2_b_8" event_ref="mx2_barline_7"/>
</staff>
<staff id="mx3_staff0">
  <clef type="G" event_ref="mx3_clef_0" staff_step="2"/>
  <time_signature num="4" den="4" event_ref="mx3_timesig_0"
  vtu_amount="64"/>
  <barline id="mx3_b_1" event_ref="mx3_barline_0"/>
  <barline id="mx3_b_2" event_ref="mx3_barline_1"/>
</staff>
<staff id="mx4_staff0">
  <clef type="G" event_ref="mx4_clef_0" staff_step="2"/>
  <time_signature num="4" den="4" event_ref="mx4_timesig_0"
  vtu_amount="64"/>
  <barline id="mx4_b_1" event_ref="mx4_barline_0"/>
  <barline id="mx4_b_2" event_ref="mx4_barline_1"/>
  <barline id="mx4_b_3" event_ref="mx4_barline_2"/>
  <barline id="mx4_b_4" event_ref="mx4_barline_3"/>
  <barline id="mx4_b_5" event_ref="mx4_barline_4"/>
  <barline id="mx4_b_6" event_ref="mx4_barline_5"/>
  <barline id="mx4_b_7" event_ref="mx4_barline_6"/>
  <barline id="mx4_b_8" event_ref="mx4_barline_7"/>
</staff>
<staff id="mx5_staff0">
  <clef type="G" event_ref="mx5_clef_0" staff_step="2"/>
  <time_signature num="4" den="4" event_ref="mx5_timesig_0"
  vtu_amount="64"/>
  <barline id="mx5_b_1" event_ref="mx5_barline_0"/>
  <barline id="mx5_b_2" event_ref="mx5_barline_1"/>
</staff>
```

```
      <staff id="mx6_staff0">
        <clef type="G" event_ref="mx6_clef_0" staff_step="2"/>
        <time_signature num="4" den="4" event_ref="mx6_timesig_0"
        vtu_amount="64"/>
        <barline id="mx6_b_1" event_ref="mx6_barline_0"/>
        <barline id="mx6_b_2" event_ref="mx6_barline_1"/>
        <barline id="mx6_b_3" event_ref="mx6_barline_2"/>
        <barline id="mx6_b_4" event_ref="mx6_barline_3"/>
        <barline id="mx6_b_5" event_ref="mx6_barline_4"/>
        <barline id="mx6_b_6" event_ref="mx6_barline_5"/>
        <barline id="mx6_b_7" event_ref="mx6_barline_6"/>
        <barline id="mx6_b_8" event_ref="mx6_barline_7"/>
      </staff>
    </staff_list>
    <part id="mx0_part1" dfstaff_ref="mx0_staff0">
      <voice_list>
        <voice_item id="mx0_voice1"/>
      </voice_list>
      <measure number="1">
        <voice ref="mx0_voice1">
          <chord event_ref="mx0_e0_0">
            <notehead staff_ref="mx0_staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx0_e0_1">
            <notehead staff_ref="mx0_staff0">
              <pitch_def staff_step="-1"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx0_e0_2">
            <notehead staff_ref="mx0_staff0">
              <pitch_def staff_step="0"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx0_e0_3">
            <notehead staff_ref="mx0_staff0">
```

```
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
      </voice>
    </measure>
    <measure number="2">
      <voice ref="mx0_voice1">
        <chord event_ref="mx0_e0_4">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_5">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="-1"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_6">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="7"/>
            <duration den="0" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_7">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="5"/>
            <duration den="-2" num="1"/>
          </notehead>
        </chord>
      </voice>
    </measure>
    <measure number="3">
      <voice ref="mx0_voice1">
        <chord event_ref="mx0_e0_8">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="3"/>
            <duration den="4" num="1"/>
```

```
            </notehead>
         </chord>
         <chord event_ref="mx0_e0_9">
            <notehead staff_ref="mx0_staff0">
               <pitch_def staff_step="4"/>
               <duration den="4" num="1"/>
            </notehead>
         </chord>
         <chord event_ref="mx0_e0_10">
            <notehead staff_ref="mx0_staff0">
               <pitch_def staff_step="5"/>
               <duration den="4" num="2"/>
            </notehead>
         </chord>
      </voice>
   </measure>
   <measure number="4">
      <voice ref="mx0_voice1">
         <chord event_ref="mx0_e0_11">
            <notehead staff_ref="mx0_staff0">
               <pitch_def staff_step="3"/>
               <duration den="4" num="1"/>
            </notehead>
         </chord>
         <chord event_ref="mx0_e0_12">
            <notehead staff_ref="mx0_staff0">
               <pitch_def staff_step="4"/>
               <duration den="4" num="1"/>
            </notehead>
         </chord>
         <chord event_ref="mx0_e0_13">
            <notehead staff_ref="mx0_staff0">
               <pitch_def staff_step="5"/>
               <duration den="4" num="2"/>
            </notehead>
         </chord>
      </voice>
   </measure>
   <measure number="5">
      <voice ref="mx0_voice1">
```

```
            <chord event_ref="mx0_e0_14">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="2"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_15">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_16">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="2"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_17">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="1"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_18">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="0"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_19">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
         </voice>
      </measure>
      <measure number="6">
         <voice ref="mx0_voice1">
```

```
        <chord event_ref="mx0_e0_20">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="2"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_21">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="3"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_22">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="2"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_23">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="1"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_24">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="0"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx0_e0_25">
          <notehead staff_ref="mx0_staff0">
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
      </voice>
    </measure>
    <measure number="7">
      <voice ref="mx0_voice1">
```

```
            <chord event_ref="mx0_e0_26">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_27">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="-5" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_28">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <rest staff_ref="mx0_staff0" event_ref="mx0_rest_0">
               <duration den="4" num="1"/>
            </rest>
         </voice>
      </measure>
      <measure number="8">
         <voice ref="mx0_voice1">
            <chord event_ref="mx0_e0_29">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_30">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="-5" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx0_e0_31">
               <notehead staff_ref="mx0_staff0">
                  <pitch_def staff_step="-2"/>
```

```
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <rest staff_ref="mx0_staff0" event_ref="mx0_rest_1">
          <duration den="4" num="1"/>
        </rest>
      </voice>
    </measure>
  </part>
  <part id="mx1_part1" dfstaff_ref="mx1_staff0">
    <voice_list>
      <voice_item id="mx1_voice1"/>
    </voice_list>
    <measure number="1">
      <voice ref="mx1_voice1">
        <rest staff_ref="mx1_staff0" event_ref="mx1_rest_0">
          <duration num="4" den="4"/>
        </rest>
      </voice>
    </measure>
    <measure number="2">
      <voice ref="mx1_voice1">
        <rest staff_ref="mx1_staff0" event_ref="mx1_rest_1">
          <duration num="4" den="4"/>
        </rest>
      </voice>
    </measure>
  </part>
  <part id="mx2_part1" dfstaff_ref="mx2_staff0">
    <voice_list>
      <voice_item id="mx2_voice1"/>
    </voice_list>
    <measure number="1">
      <voice ref="mx2_voice1">
        <chord event_ref="mx2_e0_0">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
```

```
            <chord event_ref="mx2_e0_1">
               <notehead staff_ref="mx2_staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx2_e0_2">
               <notehead staff_ref="mx2_staff0">
                  <pitch_def staff_step="0"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx2_e0_3">
               <notehead staff_ref="mx2_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
         </voice>
      </measure>
      <measure number="2">
         <voice ref="mx2_voice1">
            <chord event_ref="mx2_e0_4">
               <notehead staff_ref="mx2_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx2_e0_5">
               <notehead staff_ref="mx2_staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx2_e0_6">
               <notehead staff_ref="mx2_staff0">
                  <pitch_def staff_step="7"/>
                  <duration den="0" num="1"/>
               </notehead>
            </chord>
```

```
      <chord event_ref="mx2_e0_7">
        <notehead staff_ref="mx2_staff0">
          <pitch_def staff_step="5"/>
          <duration den="-2" num="1"/>
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="3">
    <voice ref="mx2_voice1">
      <chord event_ref="mx2_e0_8">
        <notehead staff_ref="mx2_staff0">
          <pitch_def staff_step="3"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="mx2_e0_9">
        <notehead staff_ref="mx2_staff0">
          <pitch_def staff_step="4"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="mx2_e0_10">
        <notehead staff_ref="mx2_staff0">
          <pitch_def staff_step="5"/>
          <duration den="4" num="2"/>
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="4">
    <voice ref="mx2_voice1">
      <chord event_ref="mx2_e0_11">
        <notehead staff_ref="mx2_staff0">
          <pitch_def staff_step="3"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="mx2_e0_12">
        <notehead staff_ref="mx2_staff0">
```

```
                <pitch_def staff_step="4"/>
                <duration den="4" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx2_e0_13">
              <notehead staff_ref="mx2_staff0">
                <pitch_def staff_step="5"/>
                <duration den="4" num="2"/>
              </notehead>
            </chord>
          </voice>
        </measure>
        <measure number="5">
          <voice ref="mx2_voice1">
            <chord event_ref="mx2_e0_14">
              <notehead staff_ref="mx2_staff0">
                <pitch_def staff_step="2"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx2_e0_15">
              <notehead staff_ref="mx2_staff0">
                <pitch_def staff_step="3"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx2_e0_16">
              <notehead staff_ref="mx2_staff0">
                <pitch_def staff_step="2"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx2_e0_17">
              <notehead staff_ref="mx2_staff0">
                <pitch_def staff_step="1"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx2_e0_18">
              <notehead staff_ref="mx2_staff0">
```

```
            <pitch_def staff_step="0"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_19">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
      </voice>
    </measure>
    <measure number="6">
      <voice ref="mx2_voice1">
        <chord event_ref="mx2_e0_20">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="2"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_21">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="3"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_22">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="2"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_23">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="1"/>
            <duration den="8" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_24">
          <notehead staff_ref="mx2_staff0">
```

```
            <pitch_def staff_step="0"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_25">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
      </voice>
    </measure>
    <measure number="7">
      <voice ref="mx2_voice1">
        <chord event_ref="mx2_e0_26">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="-1"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_27">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="3"/>
            <duration den="-5" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx2_e0_28">
          <notehead staff_ref="mx2_staff0">
            <pitch_def staff_step="-2"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <rest staff_ref="mx2_staff0" event_ref="mx2_rest_0">
          <duration den="4" num="1"/>
        </rest>
      </voice>
    </measure>
    <measure number="8">
      <voice ref="mx2_voice1">
        <chord event_ref="mx2_e0_29">
```

```
            <notehead staff_ref="mx2_staff0">
              <pitch_def staff_step="-1"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx2_e0_30">
            <notehead staff_ref="mx2_staff0">
              <pitch_def staff_step="3"/>
              <duration den="-5" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx2_e0_31">
            <notehead staff_ref="mx2_staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <rest staff_ref="mx2_staff0" event_ref="mx2_rest_1">
            <duration den="4" num="1"/>
          </rest>
        </voice>
      </measure>
    </part>
    <part id="mx3_part1" dfstaff_ref="mx3_staff0">
      <voice_list>
        <voice_item id="mx3_voice1"/>
      </voice_list>
      <measure number="1">
        <voice ref="mx3_voice1">
          <rest staff_ref="mx3_staff0" event_ref="mx3_rest_0">
            <duration num="4" den="4"/>
          </rest>
        </voice>
      </measure>
      <measure number="2">
        <voice ref="mx3_voice1">
          <rest staff_ref="mx3_staff0" event_ref="mx3_rest_1">
            <duration num="4" den="4"/>
          </rest>
        </voice>
```

```
      </measure>
    </part>
    <part id="mx4_part1" dfstaff_ref="mx4_staff0">
      <voice_list>
        <voice_item id="mx4_voice1"/>
      </voice_list>
      <measure number="1">
        <voice ref="mx4_voice1">
          <chord event_ref="mx4_e0_0">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_1">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="-1"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_2">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="0"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_3">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="2">
        <voice ref="mx4_voice1">
          <chord event_ref="mx4_e0_4">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="-2"/>
              <duration den="4" num="1"/>
```

```
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_5">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="-1"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_6">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="7"/>
              <duration den="0" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_7">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="5"/>
              <duration den="-2" num="1"/>
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="3">
        <voice ref="mx4_voice1">
          <chord event_ref="mx4_e0_8">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="3"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_9">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="4"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_10">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="5"/>
              <duration den="4" num="2"/>
```

```
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="4">
        <voice ref="mx4_voice1">
          <chord event_ref="mx4_e0_11">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="3"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_12">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="4"/>
              <duration den="4" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_13">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="5"/>
              <duration den="4" num="2"/>
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="5">
        <voice ref="mx4_voice1">
          <chord event_ref="mx4_e0_14">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="2"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
          <chord event_ref="mx4_e0_15">
            <notehead staff_ref="mx4_staff0">
              <pitch_def staff_step="3"/>
              <duration den="8" num="1"/>
            </notehead>
          </chord>
```

```
            <chord event_ref="mx4_e0_16">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="2"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_17">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="1"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_18">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="0"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_19">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
         </voice>
      </measure>
      <measure number="6">
         <voice ref="mx4_voice1">
            <chord event_ref="mx4_e0_20">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="2"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_21">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
```

```
            <chord event_ref="mx4_e0_22">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="2"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_23">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="1"/>
                  <duration den="8" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_24">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="0"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_25">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
         </voice>
      </measure>
      <measure number="7">
         <voice ref="mx4_voice1">
            <chord event_ref="mx4_e0_26">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_27">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="-5" num="1"/>
               </notehead>
            </chord>
```

```
            <chord event_ref="mx4_e0_28">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <rest staff_ref="mx4_staff0" event_ref="mx4_rest_0">
               <duration den="4" num="1"/>
            </rest>
         </voice>
      </measure>
      <measure number="8">
         <voice ref="mx4_voice1">
            <chord event_ref="mx4_e0_29">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="-1"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_30">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="3"/>
                  <duration den="-5" num="1"/>
               </notehead>
            </chord>
            <chord event_ref="mx4_e0_31">
               <notehead staff_ref="mx4_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
               </notehead>
            </chord>
            <rest staff_ref="mx4_staff0" event_ref="mx4_rest_1">
               <duration den="4" num="1"/>
            </rest>
         </voice>
      </measure>
   </part>
   <part id="mx5_part1" dfstaff_ref="mx5_staff0">
      <voice_list>
         <voice_item id="mx5_voice1"/>
```

```
        </voice_list>
        <measure number="1">
          <voice ref="mx5_voice1">
            <rest staff_ref="mx5_staff0" event_ref="mx5_rest_0">
              <duration num="4" den="4"/>
            </rest>
          </voice>
        </measure>
        <measure number="2">
          <voice ref="mx5_voice1">
            <rest staff_ref="mx5_staff0" event_ref="mx5_rest_1">
              <duration num="4" den="4"/>
            </rest>
          </voice>
        </measure>
      </part>
      <part id="mx6_part1" dfstaff_ref="mx6_staff0">
        <voice_list>
          <voice_item id="mx6_voice1"/>
        </voice_list>
        <measure number="1">
          <voice ref="mx6_voice1">
            <chord event_ref="mx6_e0_0">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="-2"/>
                <duration den="4" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_1">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="-1"/>
                <duration den="4" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_2">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="0"/>
                <duration den="4" num="1"/>
              </notehead>
            </chord>
```

```
      <chord event_ref="mx6_e0_3">
        <notehead staff_ref="mx6_staff0">
          <pitch_def staff_step="-2"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="2">
    <voice ref="mx6_voice1">
      <chord event_ref="mx6_e0_4">
        <notehead staff_ref="mx6_staff0">
          <pitch_def staff_step="-2"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="mx6_e0_5">
        <notehead staff_ref="mx6_staff0">
          <pitch_def staff_step="-1"/>
          <duration den="4" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="mx6_e0_6">
        <notehead staff_ref="mx6_staff0">
          <pitch_def staff_step="7"/>
          <duration den="0" num="1"/>
        </notehead>
      </chord>
      <chord event_ref="mx6_e0_7">
        <notehead staff_ref="mx6_staff0">
          <pitch_def staff_step="5"/>
          <duration den="-2" num="1"/>
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="3">
    <voice ref="mx6_voice1">
      <chord event_ref="mx6_e0_8">
        <notehead staff_ref="mx6_staff0">
```

```
            <pitch_def staff_step="3"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx6_e0_9">
          <notehead staff_ref="mx6_staff0">
            <pitch_def staff_step="4"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx6_e0_10">
          <notehead staff_ref="mx6_staff0">
            <pitch_def staff_step="5"/>
            <duration den="4" num="2"/>
          </notehead>
        </chord>
      </voice>
    </measure>
    <measure number="4">
      <voice ref="mx6_voice1">
        <chord event_ref="mx6_e0_11">
          <notehead staff_ref="mx6_staff0">
            <pitch_def staff_step="3"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx6_e0_12">
          <notehead staff_ref="mx6_staff0">
            <pitch_def staff_step="4"/>
            <duration den="4" num="1"/>
          </notehead>
        </chord>
        <chord event_ref="mx6_e0_13">
          <notehead staff_ref="mx6_staff0">
            <pitch_def staff_step="5"/>
            <duration den="4" num="2"/>
          </notehead>
        </chord>
      </voice>
    </measure>
```

```
<measure number="5">
  <voice ref="mx6_voice1">
    <chord event_ref="mx6_e0_14">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="2"/>
        <duration den="8" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_15">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="3"/>
        <duration den="8" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_16">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="2"/>
        <duration den="8" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_17">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="1"/>
        <duration den="8" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_18">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="0"/>
        <duration den="4" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_19">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="-2"/>
        <duration den="4" num="1"/>
      </notehead>
    </chord>
  </voice>
</measure>
```

```
        <measure number="6">
          <voice ref="mx6_voice1">
            <chord event_ref="mx6_e0_20">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="2"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_21">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="3"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_22">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="2"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_23">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="1"/>
                <duration den="8" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_24">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="0"/>
                <duration den="4" num="1"/>
              </notehead>
            </chord>
            <chord event_ref="mx6_e0_25">
              <notehead staff_ref="mx6_staff0">
                <pitch_def staff_step="-2"/>
                <duration den="4" num="1"/>
              </notehead>
            </chord>
          </voice>
        </measure>
```

```
<measure number="7">
  <voice ref="mx6_voice1">
    <chord event_ref="mx6_e0_26">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="-1"/>
        <duration den="4" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_27">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="3"/>
        <duration den="-5" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_28">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="-2"/>
        <duration den="4" num="1"/>
      </notehead>
    </chord>
    <rest staff_ref="mx6_staff0" event_ref="mx6_rest_0">
      <duration den="4" num="1"/>
    </rest>
  </voice>
</measure>
<measure number="8">
  <voice ref="mx6_voice1">
    <chord event_ref="mx6_e0_29">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="-1"/>
        <duration den="4" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_30">
      <notehead staff_ref="mx6_staff0">
        <pitch_def staff_step="3"/>
        <duration den="-5" num="1"/>
      </notehead>
    </chord>
    <chord event_ref="mx6_e0_31">
```

```
                <notehead staff_ref="mx6_staff0">
                  <pitch_def staff_step="-2"/>
                  <duration den="4" num="1"/>
                </notehead>
              </chord>
              <rest staff_ref="mx6_staff0" event_ref="mx6_rest_1">
                <duration den="4" num="1"/>
              </rest>
            </voice>
          </measure>
        </part>
        <horizontal_symbols/>
      </los>
    </logic>
</mx>
```

Fig. 36. Output MX dell'esecuzione della rete di Fra Martino

Come è visibile dal listato MX, ad ogni posto eseguito corrisponde un prefisso che viene aggiunto agli ID degli elementi che hanno tale attributo.

I prefissi aggiunti, visibile per esempio negli eventi della spine (e naturalmente ai puntatori ad essi, nel resto del file) sono **mx0_, mx2_, mx4_, mx6_,** che corrispondono agli eventi del **Tema**, e **mx1_, mx3_, mx5_,** per i posti **Pausa**. Gli attributi **timing** degli eventi della spine vengono processati, consentendo la sovrapposizione dei temi alle pause ed agli altri temi.

Per ogni posto eseguito è stato creato un elemento **staff** in **staff_list**, e un elemento **part** in **los**.

Nella pagina seguente è visibile l'output MX in forma notazionale.

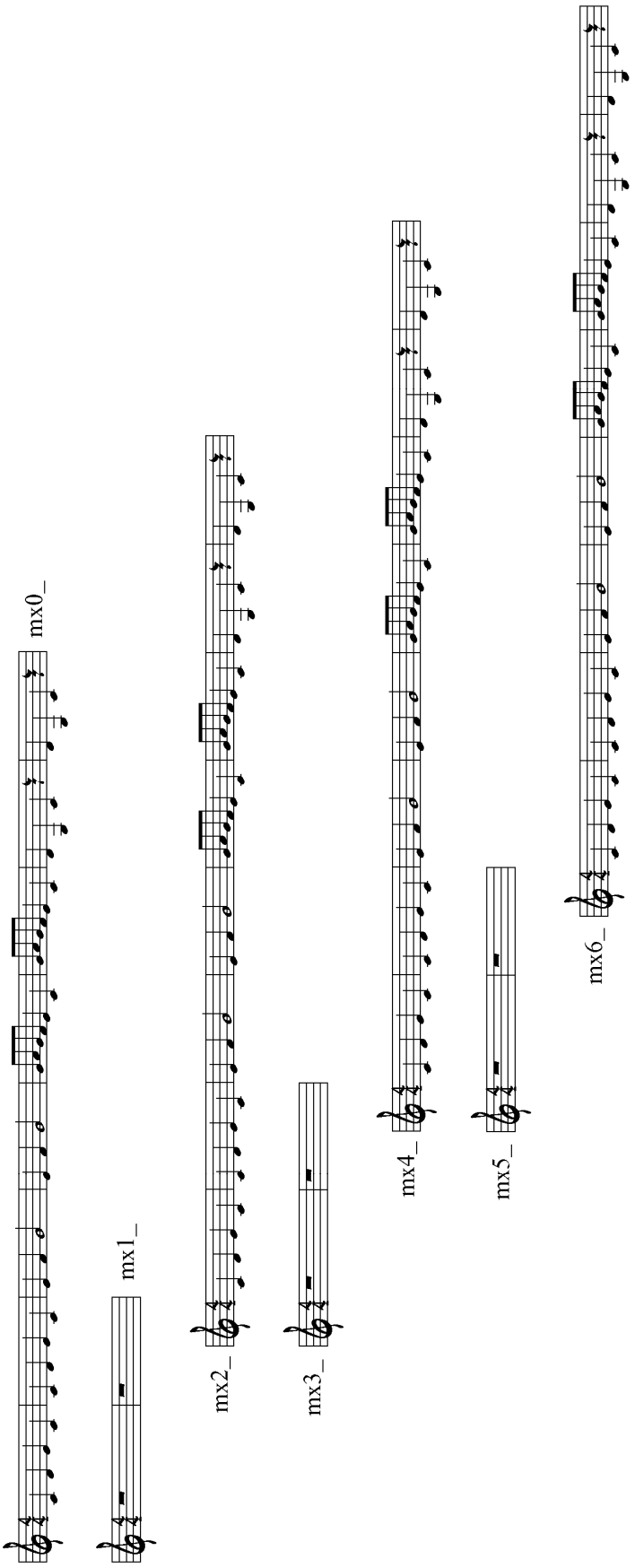


Fig. 37. Output della PN Fra Martino con i prefissi aggiunti agli IDs

Nella figura precedente gli eventi concorrenti sono allineati verticalmente, mentre l'attributo **hpos** non è considerato nella rappresentazione[14].

I files MX trattati sono stati compilati ad hoc per gli esempi esposti; questo ha comportato che un grande problema trattato nel processing non sia intervenuto: quello delle diverse granularità di definizione dei vtus. Per una trattazione specifica di questo problema rimandiamo al capitolo sulla polimetria.

6.4 SELETTORE DI TEMI

Questo esempio illustra l'utilizzo del peso probabilistico nelle situazioni di alternativa tra transizioni, realizzando una PN che, ponendo nel posto **Selettore** un numero di marche *n* fra 1 e 4, seleziona tra 4 frammenti musicali (nei posti **MX1, MX2, MX2** e **MX4**) quello numero *n*.

La PN è visibile in Fig. 38.

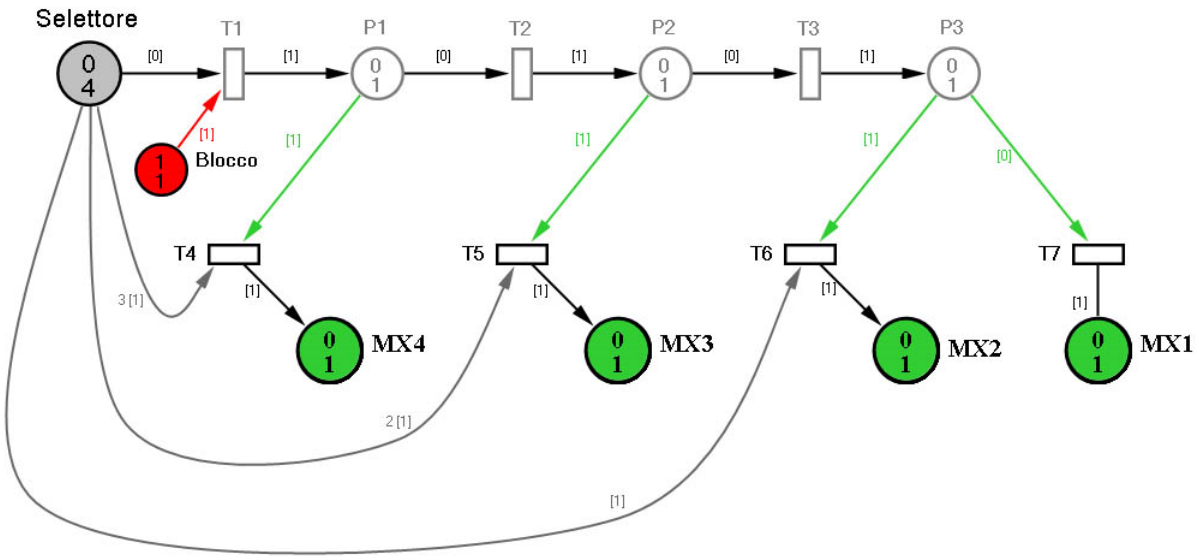


Fig. 38. PN **Selettore di temi**

Prima di illustrare il funzionamento della rete, facciamo qualche osservazione sui pesi probabilistici associati agli archi: un arco con peso probabilistico 0 farà scattare la transizione associata soltanto quando non è in alternativa/conflitto con altri archi con peso maggiore di 0.

[14] Quando ScoreSynth sovrappone o giustappone frammenti MX gli attributi **hpos** dei posti vengono semplicemente copiati del file di output elaborato. Questa scelta è dettata dal fatto che le specifiche dell'MX su questo parametro non sono ancora definitive, ed è sembrato inutile trattarlo nel processing dei documenti basandosi su regole in corso di modifica.

L'arco quindi potrà far scattare la transizione soltanto: a) quando è l'unico arco della rete che possa far scattare una transizione, oppure b) quando sono in alternativa/conflitto soltanto transizioni con archi di peso probabilistico 0, nel qual caso la transizione che scatta sarà scelta casualmente[15].

Vediamo il funzionamento della rete con il numero variabile di marche poste inizialmente nel posto **Selettore**:

- **4 marche**: all'inizio dell'esecuzione l'unica transizione che può scattare è **T1**, visto che le altre transizioni hanno almeno un posto in ingresso senza marche. La transizione scatta anche se l'arco da **Selettore** a **T1** ha peso probabilistico 0, visto che non ci sono altri archi in alternativa/conflitto. Ora **P1** ha una marca e 3 marche sono rimaste in **Selettore**. Questo fa sì che esista un alternativa tra lo scatto di **T2** e **T4**. Considerando però che l'arco che collega **P1** a **T2** ha peso 0, verrà scelta **T4** che, scattando, esegue il frammento MX nel posto **MX4**.
- **3 marche**: dopo lo scatto iniziale di **T1 (**come visto nel caso precedente), **T4** non può scattare perché l'arco che proviene da **Selettore** ha peso in marche 3 mentre il posto in ingresso ha soltanto 2 marche. L'unica transizione abilitata è **T2** che, scattando, pone una marca in **P2**. Ora si verifica una situazione di alternativa analoga al punto precedente, con **T5** che, avendo peso probabilistico maggiore di 0, scatterà eseguendo **MX3**.
- **2 marche**: il funzionamento iniziale della rete è analogo ai punti precedenti: riprendiamo quindi dallo stato illustrato in Fig. 39. A questo punto entrano in gioco il posto **Blocco**: se esso non fosse presente in questo stato scatterebbero le transizioni **T3** e **T1**, che hanno entrambe in ingresso archi con peso probabilistico 0, ma che non sono in alternativa/conflitto con altre. Questo comporterebbe la mancanza di un'altra marca in **Selettore** e l'inibizione quindi di **T6**. Inserendo invece il posto **Blocco**, **T1** può scattare solo all'inizio dell'esecuzione, e, partendo dalla situazione in Fig. 39, scatta **T3**, poi **T6**, eseguendo il posto **MX2**.
- **1 marca**: nella situazione finale del punto precedente **P3** conteneva una marca. In questo caso **T6** non può scattare perché **Selettore** è vuoto: viene eseguito il posto **MX1**.

[15] In verità nell'algoritmo di scelta viene selezionato in questo caso il primo arco della lista degli archi in alternativa/conflitto e viene eseguita la transizione corrispondente.

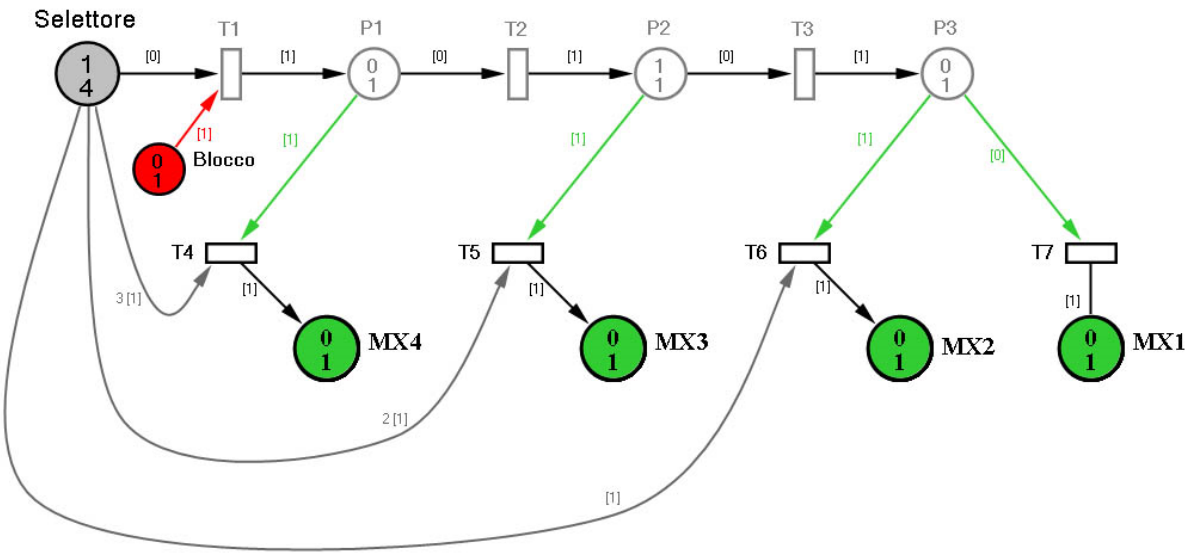


Fig. 39. PN **Selettore di temi** durante l'esecuzione

Naturalmente la rete descritta può essere estesa ad un selettore fra un numero di temi qualsiasi, soltanto aggiungendo alla struttura altri elementi in modo simile a quelli esistenti.

CAPITOLO 7

COMPONENTI E ALGORITMI

7.1 TECNOLOGIE UTILIZZATE

L'IDE di Microsoft Visual Studio .NET 2003 utilizzato per il presente lavoro incorpora il concetto di *soluzione*, ovvero un contenitore di tutti i progetti che contribuiscono a risolvere un problema e di eventuali dati ad esso correlati. In questo modo è possibile organizzare gerarchicamente un progetto complesso, dividendolo in sottoprogetti, anche sviluppati da team diversi, consentendo allo stesso tempo di avere sotto controllo la visione globale e di definire modi comuni per lo sviluppo, la generazione e il deployment.

Nel presente caso la soluzione comprende il progetto `AB.PetriNets`, che genera una dll per il trattamento di PNs, e il progetto `ScoreSynth`, l'applicazione vera e propria che, basandosi sulla dll di cui sopra, fornisce l'interfaccia grafica per lavorare con PNs musicali contenenti dati MX.

Entrambi i progetti si avvantaggiano di alcune soluzioni innovative, quali i *namespaces* e il *commento XML* [PG02].

Namespaces. Il concetto di namespace assicura che non esistano conflitti di nomi di classi, garantendo che tutti i nomi usati in un progetto siano unici.

Può succedere infatti che più sviluppatori utilizzino nelle loro classi gli stessi nomi: in questo caso inserendo le dichiarazioni delle classi in un namespace diverso vengono risolte le ambiguità che potrebbero sorgere. Chiariamo il concetto con un esempio, che illustra anche la sintassi di dichiarazione in C#: supponiamo che due aziende, la Macrosoftware e la Pear implementino la stessa classe `OS`, e che si voglia, in un progetto, utilizzarle entrambe; in questo caso se le due case software hanno utilizzano i namespaces in C# avranno dichiarato le loro classi in un modo simile al seguente:

```
namespace Macrosoftware.TheBestSoftware
{
    class OS
```

```
        …
    }
```

e

```
    namespace Pear.UserFriendlySoftware
    {
        class OS
        …
    }
```

Nel caso occorra utilizzare la prima classe la si indicherà con `Macrosoftware.TheBestSoftware.OS`, mentre per la seconda classe si scriverà `Pear.UserFriendlySoftware.OS`.

Naturalmente esiste un modo per ridurre la quantità di codice da scrivere: usando la parola chiave `using`, che dichiara un prefisso da utilizzare automaticamente nei riferimenti dei nomi e che consente inoltre di creare degli alias come scorciatoie.

Lo stesso .NET Framework definisce quasi 90 namespaces che iniziano col prefisso `System` e 5 che iniziano con `Microsoft`; inoltre al suo interno riutilizza gli stessi nomi di classi in namespaces diversi (come nell'esempio appena esposto), come la classe `Timer`, che si trova nei namespaces `System.Timers`, `System.Threading` e `System.Windows.Forms`.

Commenti XML. Il C# incorpora un sistema di generazione automatica di file di supporto basata sull'utilizzo, all'interno del codice, di commenti XML. Un commento XML, relativo ad una classe, un delegato, un'interfaccia, campi, eventi, proprietà e metodi, viene specificato utilizzando tre barre /// a cui far seguire un vero e proprio frammento di XML che commenta l'elemento che segue.

Se nel codice sono presenti commenti XML l'ambiente Microsoft genera quindi durante la compilazione un file XML come report, contenente gli elementi presenti nel codice e i commenti relativi. Attraverso l'IDE è anche possibile creare automaticamente una serie di pagine HTML che visualizzano gerarchicamente le classi dei progetti e consentono di navigare attraverso di esse, incorporando i commenti annessi e informazioni sui tipi di dati e i modificatori di accesso.

Questa tecnologia, presente soltanto nel linguaggio C#, facilita enormemente il riutilizzo di codice esistente, con un metodo user-friendly per la sua consultazione.

7.2 STRUTTURA DELLE CLASSI

7.2.1 IL PROGETTO AB.PETRINETS

Il progetto è costituito da 15 classi, tutte facenti parte del namespace `AB.PetriNets`. I files del progetto sono organizzati anteponendo ad ogni loro nome (tranne il file `PNException.cs`, contenente la classe usata per creare eccezioni custom) un prefisso che indica una funzionalità comune all'interno della filosofia di progetto. I prefissi utilizzati sono: `Base`, che raggruppa le classi base per la gestione a livello primitivo di PNs, `Collections`, che gestisce gli insiemi di archi, posti e transizioni costituenti una PN, `Graphic`, con tutte le procedure di visualizzazione grafica di PNs, e `Execution`, per l'esecuzione delle PNs. Le classi del gruppo `Graphic` sono dei controlli creati da zero [Pet02].

In Tab. 7 vengono mostrate le classi facenti parte di ogni file del progetto, insieme alle funzionalità peculiari ivi contenute, mentre in Fig. 40 è presentato uno schema della struttura delle classi del progetto.

Si sono volute mantenere ben divise le classi che operano sulle PNs a livello non grafico da quelle di visualizzazione ed interazione, con la possibilità di riutilizzo di parte del codice in ambiti diversi[16].

File	*Classi*	*Funzionalità*
`Base.PetriNet.cs`	`PetriNet`	Creazione della PN, campi e metodi generali, load/save in PNML, gestione dell'esecuzione della PN
`Base.Place.cs`	`Place`	Gestione generale di un posto, definizione dell'evento generato all'esecuzione di un posto
`Base.Transition.cs`	`Transition`	Gestione generale di una transizione, definizione dell'evento generato allo scatto di una transizione
`Base.Arc.cs`	`Arc`	Gestione generale di un arco
`Collections.Places.cs`	`Places`	Gestione dei posti di una PN

[16] Un esempio è presentato nel capitolo successivo, con un programma console che da riga di comando accetta il nome del file PNML di input e genera il file MX dall'esecuzione della PN corrispondente.

`Collections.Transitions.cs`	`Transitions`	Gestione delle transizioni di una PN
`Collections.Arcs.cs`	`Arcs`	Gestione degli archi di una PN
`Graphic.PetriNet.cs`	`GraphicPetriNet`	Gestione della grafica di una PN, interazione con mouse e tastiera, dialoghi di cambiamento delle proprietà, stampa della PN
`Graphic.Place.cs`	`GraphicPlace`	Gestione della grafica di un posto, interazione con mouse e tastiera
`Graphic.Transition.cs`	`GraphicTransition`	Gestione della grafica di una transizione, interazione con mouse e tastiera
`Graphic.Arc.cs`	`GraphicArc`	Gestione della grafica di un arco, interazione con mouse e tastiera
`Execution.TimeLine.cs`	`TimeLine`	Gestione della timeline globale dell'esecuzione di una PN, sviluppo di sottonodi, handler degli eventi di esecuzione per la gestione dei documenti MX
`Execution.TimeLineStep.cs`	`TimeLineStep`	Singolo passo dell'esecuzione di una PN (algoritmo di esecuzione)
`Execution.TokensMap.cs`	`TokensMap`, `TokensMapRecord`	Gestione della mappa delle marche attive durante l'esecuzione di una PN
`PNException.cs`	`PNException`	Eccezione custom invocata nel progetto

Tab. 7. File del progetto `AB.PetriNets`

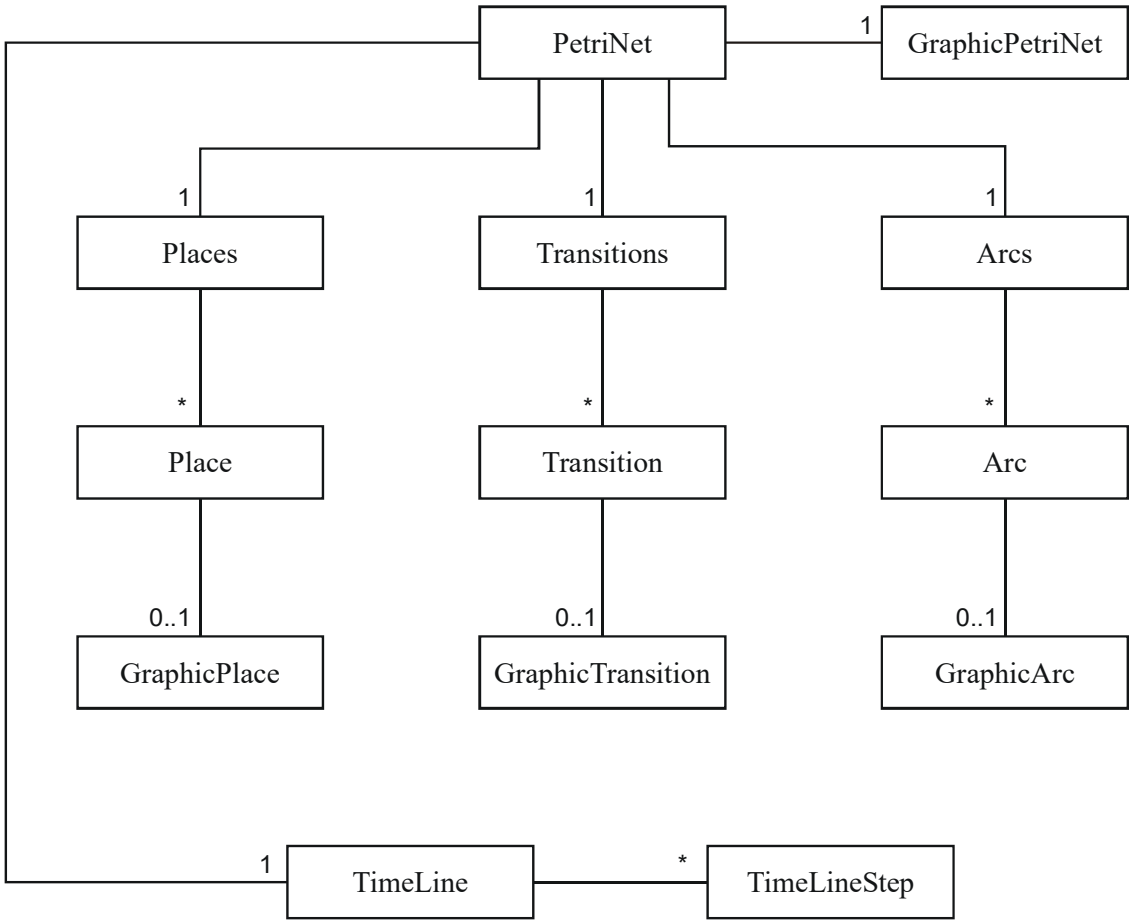


Fig. 40. Struttura delle classi di `AB.PetriNets`

7.2.2 IL PROGETTO SCORESYNTH

Il progetto ScoreSynth incorpora sia l'interfaccia che un componente che interagisce con essa e che fornisce la rappresentazione della Timeline grafica su cui vengono presentate in ordine temporale le tracce costituite dai documenti MX che vengono eseguiti. In Fig. 41 viene mostrata l'interfaccia completa.

La zona centrale della finestra d'interfaccia offre un'area per il disegno delle PNs, secondo lo standard MDI (Multiple Document Interface), ovvero avendo la possibilità di lavorare con più finestre aperte.

Nella parte sinistra ed in basso sono invece presenti una serie di finestre mobili, che possono essere ancorate ai bordi della finestra principale, spostate, chiuse o venire chiuse e ripristinate automaticamente al passaggio del mouse.

Project Explorer. La finestra contiene la lista delle reti aperte e consente di definire la rete principale da eseguire.

Nets. In questa finestra viene presentata una struttura ad albero che rappresenta la gerarchia di reti e sottoreti contenute nel progetto.

Properties. Visualizza le proprietà dell'oggetto selezionato o della PN in caso non sia selezionato nulla.

Places. Riepilogo dei posti con indicazione del nome, marche, capacità, PN di appartenenza, file associato. E' possibile scegliere se mostrare solo i posti della rete visualizzata correntemente oppure di tutte le reti del progetto.

Transitions. Riepilogo delle transizioni con indicazione del nome, della rete di appartenenza e del file associato. Come per i posti è possibile scegliere il tipo di visualizzazione.

Tokens. Riepilogo delle marche presenti nelle reti.

Timeline. Finestra che mostra le tracce che costituiscono l'esecuzione della PN principale.

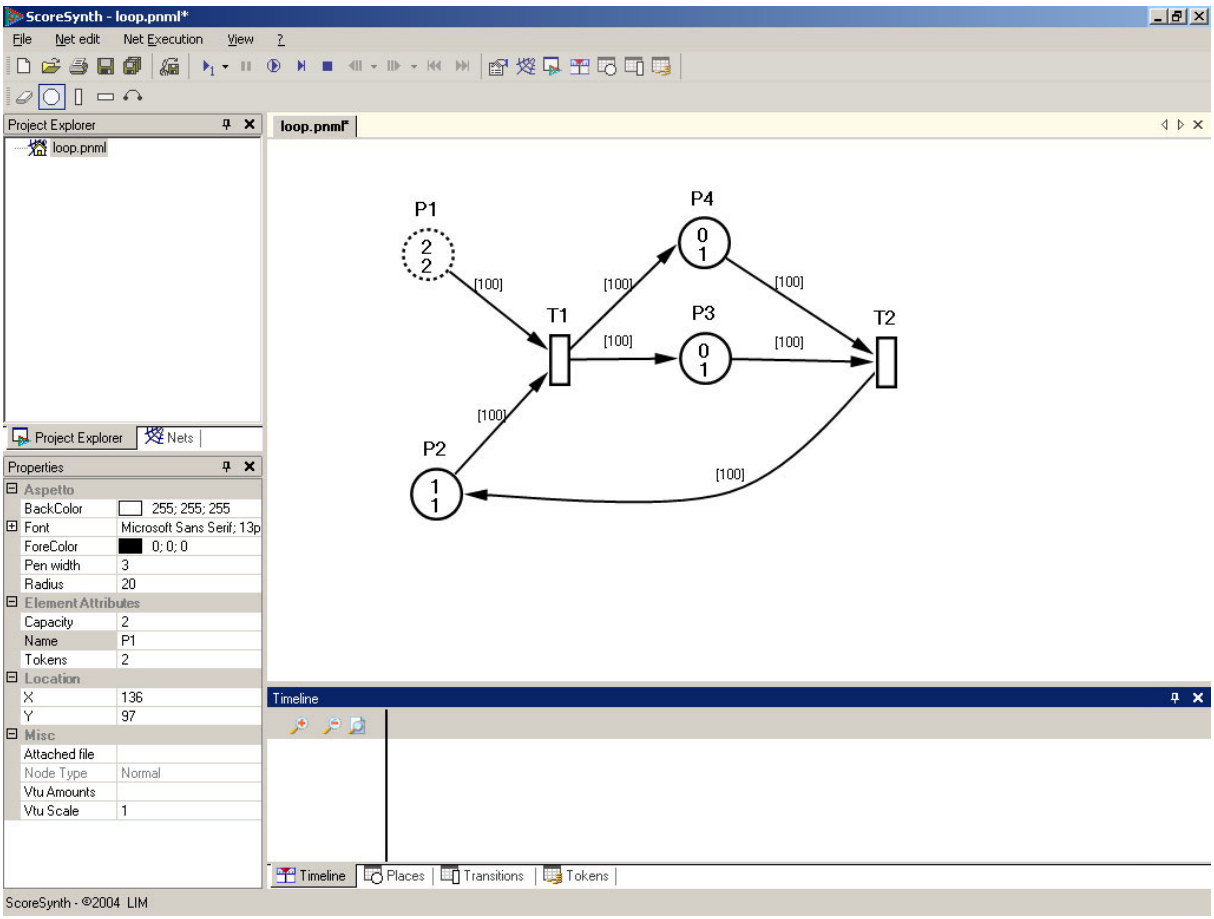


Fig. 41. Interfaccia di ScoreSynth

7.3 LE CLASSI DEL NAMESPACE AB.PETRINETS

Di seguito vengono esposte le classi del namespace `AB.PetriNets`, indicando i loro campi e proprietà (C) e metodi (M) principali; gli algoritmi fondamentali presenti vengono descritti in un apposito paragrafo più avanti.

7.3.1 PETRINET

- `PetriNet(Control parent)` : Costruttore della classe. `Parent` deve essere il controllo padre in cui visualizzare la PN graficamente, `null` se non si desidera una rappresentazione grafica.
- (C) `GraphicPetriNet Graphic` : Campo che si riferisce all'oggetto `GraphicPetriNet` nel caso sia stata creata una PN con rappresentazione grafica.
- (C) `Places Places, Transitions Transitions, Arcs Arcs` : Contengono posti, transizioni ed archi della PN.
- (C) `bool Compile` : Campo booleano che, se impostato su `true` genera il file MX in seguito all'esecuzione della PN; è impostato su `false` durante il debugging della rete.
- (C) `bool InNode, bool OutNode` : Riferimenti al nodo di input e di output nel caso la PN sia una sottorete.
- (M) `PetriNet Clone(bool GraphicClone)`: Crea un clone dello stesso oggetto `PetriNet`. Con `GraphicClone` a `false` non copia il corrispettivo oggetto grafico.
- (M) `Places InPlaces(Transition t)` : Restituisce un oggetto contenente la lista dei posti in ingresso alla transizione passata come parametro.
- (M) `Places OutPlaces(Transition t)` : Restituisce un oggetto contenente la lista dei posti in uscita alla transizione passata come parametro.
- (M) `Transitions InTransitions(Place p)` : Restituisce un oggetto contenente la lista delle transizioni in ingresso al posto passato come parametro.
- (M) `Transitions OutTransitions(Place p)` : Restituisce un oggetto contenente la lista delle transizioni in uscita al posto passato come parametro.
- (M) `Arc GetArc(object from, object to)` : Restituisce l'arco che collega il posto/transizione `from` al posto/transizione `to`.

- (M) `void LoadFromFile(string str)` : Carica la PN dal file PNML specificato.
- (M) `void SaveToFile(string str)` : Salva la PN sul file PNML specificato.
- (C) `TimeLine TimeLine` : L'oggetto `TimeLine` che tiene traccia dei passi dell'esecuzione della PN.
- (M) `bool ExecuteStep(int stepIndex)` : Esegue lo step `stepIndex` dell'esecuzione della PN, ritornando `true` se si verifica un errore o se la rete è bloccata definitivamente. Se esistono passi nella timeline successivi a `stepIndex` li cancella prima di eseguire il passo. Il metodo deve essere chiamato con parametro `0` prima dell'esecuzione vera e propria per inizializzare la timeline[17].
- (M) `bool SetStep(int stepIndex)` : Setta lo stato della PN al passo specificato, ripristinando i valori delle marche nei posti. Passando `-1` come parametro la timeline viene cancellata e la rete torna allo stato originale.

7.3.2 PLACES

- `Places()` : Costruttore della classe.
- (C) `ArrayList places` : ArrayList contenente gli oggetti `Place` della PN.
- (M) `Place Add(Control parent)` : Aggiunge un posto alla PN, con nome P*n*, dove *n* è il primo intero maggiore di 1 non corrispondente ad un posto già esistente. Passa `parent` al costruttore di `Place` e restituisce il posto creato.
- (M) `Place Add(Place p)` : Aggiunge il posto specificato alla PN.
- (M) `void Remove(Place p)` : Rimuove il posto specificato dalla lista dei posti della PN. Se non esiste viene generata un'eccezione.
- `Place this[int index]` : Indicizzatore che consente di riferirsi a `Places[n]` per indicare il posto di indice `n` nella lista dei posti della PN.
- `Place this[string id]` : Indicizzatore che consente di riferirsi a `Places[ID]` per indicare il posto avente l'`id` specificato[18].
- `IEnumerator GetEnumerator()` : Consente di utilizzare l'istruzione `foreach` per ciclare sui posti della PN.
- (C) `int Count` : Restituisce il numero di posti della PN.

[17] Il passo 0 della timeline corrisponde ad una copia della rete allo stato originale.

[18] L'`id` viene assegnato agli elementi di una PN quando viene salvata su file PNML e corrisponde proprio all'attributo `id` dell'elemento XML corrispondente.

- (M) `Places Clone()` : Crea un clone dello stesso oggetto `Places`.
- (M) `int IndexByName(string placeName)` : Restituisce l'indice nella lista dei posti del posto con nome passato come parametro. Se il posto non esiste restituisce `-1`.
- (M) `int IndexOf(Place p)` : Restituisce l'indice nella lista dei posti del posto passato come parametro. Se il posto non esiste restituisce `-1`.

7.3.3 TRANSITIONS

- `Transitions()` : Costruttore della classe.
- (C) `ArrayList transitions` : ArrayList contenente gli oggetti `Transition` della PN.
- (M) `Transition Add(Control parent)` : Aggiunge una transizione alla PN, con nome `T`*n*, dove *n* è il primo intero maggiore di 1 non corrispondente ad una transizione già esistente. Passa `parent` al costruttore di `Transition` e restituisce la transizione creata.
- (M) `Transition Add(Transition t)` : Aggiunge la transizione specificata alla PN.
- (M) `void Remove(Transition t)` : Rimuove la transizione specificata dalla lista delle transizioni della PN. Se non esiste viene generata un'eccezione.
- `Transition this[int index]` : Indicizzatore che consente di riferirsi a `Transitions[n]` per indicare la transizione di indice `n` nella lista delle transizioni della PN.
- `Transition this[string id]` : Indicizzatore che consente di riferirsi a `Transitions[ID]` per indicare la transizione avente l'`id` specificato[19].
- `IEnumerator GetEnumerator()` : Consente di utilizzare l'istruzione `foreach` per ciclare sulle transizioni della PN.
- (C) `int Count` : Restituisce il numero delle transizioni della PN.
- (M) `Transitions Clone()` : Crea un clone dello stesso oggetto `Transitions`.
- (M) `int IndexByName(string transitionName)` : Restituisce l'indice nella lista delle transizioni della transizione con nome passato come parametro. Se la transizione non esiste restituisce `-1`.

[19] L'`id` viene assegnato agli elementi di una PN quando viene salvata su file PNML e corrisponde proprio all'attributo `id` dell'elemento XML corrispondente.

- (M) `int IndexOf(Transition t)` : Restituisce l'indice nella lista delle transizioni della transizione passata come parametro. Se la transizione non esiste restituisce `-1`.

7.3.4 ARCS

- `Arcs()` : Costruttore della classe.
- (C) `ArrayList arcs` : ArrayList contenente gli oggetti `Arc` della PN.
- (M) `Arc Add(Place p, Transition t, ArcType arcType, Control parent)` : Aggiunge un arco alla PN che connette il posto `p` con la transizione `t`, di verso specificato da `arcType` (v. classe `Arc`). Passa `parent` al costruttore di `Arc` e restituisce l'arco creato. Se l'arco è già presente incrementa il suo peso.
- (M) `Arc Add(Arc a)` : Aggiunge l'arco specificato alla PN.
- (M) `void Remove(Arc a)` : Rimuove l'arco specificato dalla lista degli archi della PN. Se non esiste viene generata un'eccezione.
- `Arc this[int index]` : Indicizzatore che consente di riferirsi a `Arcs[n]` per indicare l'arco di indice `n` nella lista degli archi della PN.
- `IEnumerator GetEnumerator()` : Consente di utilizzare l'istruzione `foreach` per ciclare sugli archi della PN.
- (C) `int Count` : Restituisce il numero di archi della PN.
- (M) `Arcs Clone()` : Crea un clone dello stesso oggetto `Arcs`.
- (M) `int ArcIndex(Place p, Transition t, ArcType arcType)` : Restituisce l'indice nella lista dei posti del posto con nome passato come parametro. Se l'arco non esiste restituisce `-1`.
- (M) `int IndexOf(Arc a)` : Restituisce l'indice nella lista dei posti del posto passato come parametro. Se l'arco non esiste restituisce `-1`.

7.3.5 PLACE

- `enum PlacType`[20] : Indica se il posto è un sottonodo. I valori sono `Normal` e `Subnet`.
- `Place(Control parent)` : Costruttore della classe. `parent` è il controllo padre che ospita la parte grafica del posto, `null` se non si desidera crearla. Nel progetto viene passato il campo `Graphic` di `PetriNet`.
- (C) `GraphicPlace Graphic` : Campo che si riferisce all'oggetto `GraphicPlace` nel caso sia stata creata una PN con rappresentazione grafica.
- (C) `string Name` : Ottiene o imposta il nome del posto.
- (C) `int NTokens` : Ottiene o imposta il numero di marche del posto.
- (C) `int Capacity` : Ottiene o imposta la capacità del posto.
- (C) `string File` : Ottiene o imposta il nome del file associato al posto. Se l'allegato è un file MX aggiorna il campo `VtuAmountsArray`.
- (C) `ArrayList VtuAmountsArray` : Lista dei tempi musicali del file MX associato al posto con la quantità `vtu_amount` associata.
- (C) `int VtuScale` : Ottiene o imposta il fattore di scala da applicare ai vtus presenti nella spine del file MX associato al posto durante l'esecuzione della PN.
- (C) `object InputObject` : In fase di esecuzione della PN, contiene l'oggetto che il posto riceve in input. Nel presente progetto è un documento MX, trasferito dallo scatto della transizione in ingresso, nel caso il posto non abbia già associato un file MX.
- (C) `string InputObjectType` : Stringa per identificare il tipo di oggetto presente in input: in ScoreSynth vale sempre `"mx"`.
- (C) `PlaceType Type` : Indica il tipo di posto (`Normal` o `Subnet`).
- (C) `long Duration` : Proprietà di sola lettura che restituisce la durata dell'oggetto associato al posto.
- (C) `bool IsInput` : Proprietà di sola lettura uguale a `true` se il posto è il nodo di input di una sottorete.
- (C) `bool IsOutput` : Proprietà di sola lettura uguale a `true` se il posto è il nodo di output di una sottorete.

[20] L'enumerazione è dichiarata nel file `Base.Place.cs`, esternamente alla classe `Place`.

- (C) `int freeTokens` : Durante l'esecuzione della PN, indica il numero di marche liberatesi dopo l'esecuzione dell'oggetto associato al posto e disponibili in output.
- (C) `int inTokens` : Durante l'esecuzione della PN, indica il numero massimo di marche che il posto può ricevere.
- (C) `int outTokens` : Durante l'esecuzione della PN, indica il numero massimo di marche che il posto può fornire in output.
- (C) `object Track` : Oggetto che incorpora informazioni sulla traccia occupata nella timeline[21] dall'oggetto musicale associato al posto durante l'esecuzione della PN.
- (C) `string ID` : Un id unico assegnato al posto durante il salvataggio su file PNML della PN. All'interno della codifica XML corrisponde all'attributo `id` dell'elemento `place` (v. cap. PNML più sopra).
- (C) `string PNMLFileName` : Il nome del file PNML contenente la PN di cui fa parte il posto.
- (M) `Place Clone()` : Crea un clone dello stesso oggetto `Place`.
- (M) `void Copy(Place p)` : Copia l'oggetto stesso nell'oggetto `Place` specificato.
- (M) `string ToString()` : Effettua l'override del metodo `ToString` restituendo il nome del posto.
- `event EventHandler Execution` : Evento invocato all'esecuzione del posto durante la procedura di esecuzione della PN.
- (M) `void Execute()` : Invoca l'evento `Execution`.

7.3.6 TRANSITION

- `enum TransitionType`[22] : Indica se la transizione è un sottonodo. I valori sono `Normal` e `Subnet`.
- `enum TransitionStatus` : Durante l'esecuzione della PN, indica lo stato della transizione. I valori sono: `Null`, `Alternative`, `Executable`, `NotExecutable`, `Conflict`, `AlternativeConflict`.

[21] In questo caso con il termine *timeline* non si indica l'oggetto associato a `PetriNet` ma la timeline presente nell'interfaccia grafica di ScoreSynth.

[22] Le enumerazioni `TransitionType` e `TransitionStatus` sono dichiarate nel file `Base.Transition.cs`, esternamente alla classe `Transition`.

- `Transition(Control parent)` : Costruttore della classe. `parent` è il controllo padre che ospita la parte grafica della transizione, `null` se non si desidera crearla. Nel progetto viene passato il campo `Graphic` di `PetriNet`.
- (C) `GraphicTransition Graphic` : Campo che si riferisce all'oggetto `GraphicTransition` nel caso sia stata creata una PN con rappresentazione grafica.
- (C) `string Name` : Ottiene o imposta il nome della transizione.
- (C) `string File` : Ottiene o imposta il nome del file associato alla transizione. Se l'allegato è un file MX aggiorna il campo `VtuAmountsArray`.
- (C) `ArrayList VtuAmountsArray` : Lista dei tempi musicali del file MX associato alla transizione con la quantità `vtu_amount` associata.
- (C) `int VtuScale` : Ottiene o imposta il fattore di scala da applicare ai vtus presenti nella spine del file MX associato alla transizione durante l'esecuzione della PN.
- (C) `object InputObject` : In fase di esecuzione della PN, contiene l'oggetto che la transizione riceve in input. Nel presente progetto è un documento MX, trasferito dal posto in ingresso quando la sua esecuzione è conclusa, nel caso la transizione non abbia già associato un file MX.
- (C) `string InputObjectType` : Stringa per identificare il tipo di oggetto presente in input: in ScoreSynth vale sempre `"mx"`.
- (C) `TransitionType Type` : Indica il tipo di transizione (`Normal` o `Subnet`).
- (C) `TransitionStatus status` : Indica lo stato della transizione durante l'esecuzione.
- (C) `bool IsInput` : Proprietà di sola lettura uguale a `true` se la transizione è il nodo di input di una sottorete.
- (C) `bool IsOutput` : Proprietà di sola lettura uguale a `true` se la transizione è il nodo di output di una sottorete.
- (C) `string ID` : Un id unico assegnato alla transizione durante il salvataggio su file PNML della PN. All'interno della codifica XML corrisponde all'attributo `id` dell'elemento `transition` (v. cap. PNML più sopra).
- (C) `string PNMLFileName` : Il nome del file PNML contenente la PN di cui fa parte la transizione.
- (M) `Transition Clone()` : Crea un clone dello stesso oggetto `Transition`.

- (M) `void Copy(Transition t)` : Copia l'oggetto stesso nell'oggetto `Transition` specificato.
- (M) `string ToString()` : Effettua l'override del metodo `ToString` restituendo il nome della transizione.
- `event EventHandler Execution` : Evento invocato all'esecuzione della transizione durante la procedura di esecuzione della PN.
- (M) `void Execute()` : Invoca l'evento `Execution`.

7.3.7 ARC

- `enum ArcType`[23] : Indica il verso dell'arco che collega un posto con una transizione. I valori sono `FromPlaceToTransition` e `FromTransitionToPlace`.
- `Arc(Place p, Transition t, ArcType arcType, Control parent)` : Costruttore della classe, crea un arco che collega `p` con `t` e con verso specificato da `arcType`. `parent` è il controllo padre che ospita la parte grafica dell'arco, `null` se non si desidera crearla. Nel progetto viene passato il campo `Graphic` di `PetriNet`.
- (C) `GraphicArc Graphic` : Campo che si riferisce all'oggetto `GraphicArc` nel caso sia stata creata una PN con rappresentazione grafica.
- (C) `Place Place` : Ottiene o imposta il posto connesso dall'arco.
- (C) `Transition Transition` : Ottiene o imposta la transizione connessa dall'arco.
- (C) `int TokensWeight` : Indica il peso in marche dell'arco.
- (C) `int ProbWeight` : Indica il peso probabilistico dell'arco.
- (C) `ArcType Type` : Indica il tipo di arco (`FromPlaceToTransition` o `FromTransitionToPlace`).
- (C) `string PNMLFileName` : Il nome del file PNML contenente la PN di cui fa parte l'arco.
- (M) `Arc Clone()` : Crea un clone dello stesso oggetto `Arc`.
- (M) `void Copy(Arc a)` : Copia l'oggetto stesso nell'oggetto `Arc` specificato.

[23] L'enumerazione è dichiarata nel file `Base.Arc.cs`, esternamente alla classe `Arc`.

- (M) `string ToString()` : Effettua l'override del metodo `ToString` resituendo una stringa con "NomePosto -> NomeTransizione" oppure "NomeTransizione -> NomePosto".

7.3.8 GRAPHICPETRINET

- `enum PNAction` : Tipo di azione grafica corrente. I valori sono `AddPlace`, `AddVerticalTransition`, `AddHorizontalTransition`, `AddArc`, `Eraser`.
- `GraphicPetriNet()` : Costruttore della classe.
- (C) `PetriNet ReferredPetriNet` : La PN di cui l'oggetto è rappresentazione grafica.
- (C) `bool IsInExecution` : Indica se la PN è in fase di esecuzione: viene utilizzato in quel caso per vietare l'editing della rete.
- (C) `PNAction CurrentAction` : Imposta o ottiene il tipo di azione di editing in corso.
- (C) `ArrayList arcPoints` : Durante il disegno di un arco tiene traccia dei punti intermedi.
- (C) `Point oldArcPoint` : Durante il disegno di un arco tiene traccia del punto intermedio precedente.
- (C) `object arcFrom` : Durante il disegno di un arco indica il posto o la transizione di partenza.
- (C) `object arcTo` : Durante il disegno di un arco indica il posto o la transizione di arrivo.
- (C) `bool arcCreationInProgress` : Indica se è in corso il disegno di un arco.
- (M) `void PlaceAdd(GraphicPlace p)` : Aggiunge la rappresentazione grafica di un posto alla rete grafica corrente.
- (M) `void PlaceRemove(GraphicPlace p)` : Rimuove un posto dalla PN.
- (M) `void TransitionAdd(GraphicTransition t)` : Aggiunge la rappresentazione grafica di una transizione alla rete grafica corrente.
- (M) `void TransitionRemove(GraphicTransition t)` : Rimuove una transizione dalla PN.

- (M) `void ArcAdd(GraphicArc a)` : Aggiunge la rappresentazione grafica di un arco alla rete grafica corrente.
- (M) `void ArcRemove(GraphicArc a)` : Rimuove un arco dalla PN.
- (M) `void RemoveArcsFromOrTo(object obj)` : Rimuove gli archi connessi al posto o alla transizione passati come parametro.
- (M) `void NodeLocationChanged(object obj, EventArgs ea)` : Intercetta l'evento invocato quando un posto o una transizione vengono spostati, ridisegnando gli archi connessi.
- (M) `void OnPaint(PaintEventArgs pea)` : Effettua l'override del metodo `OnPaint` per disegnare il controllo, ovvero la PN.
- (M) `void OnClick(EventArgs ea)` : Effettua l'override del metodo `OnClick` per gestire la creazione di posti, transizioni e punti intermedi di archi.
- (M) `void ChildOnClick(object obj, EventArgs ea)` : Gestisce l'evento invocato al click del mouse su posti, transizioni ed archi, eseguendo l'operazione correntemente selezionata.
- (M) `void ChildOnKeyDown(object obj, KeyEventArgs kea)` : Gestisce l'evento invocato alla pressione di un tasto. Le operazioni eseguite sono l'annullamento della procedura di disegno di archi (tasto ESC) e la cancellazione degli elementi selezionati (tasto CANC).
- (M) `void OnMouseMove(MouseEventArgs mea)` : Effettua l'override del metodo `OnMouseMove` per disegnare la traccia durante la creazione dei punti intermedi degli archi.
- (M) `int InitDialogPT(Form dialog, string labelText, string defaultText, int defaultWidth, Point defaultLocation)` : Inizializza il dialogo utilizzato per settare nomi di posti, nomi di transizioni, marche e capacità.
- (M) `void PlaceNameLabelOnDoubleClick(object obj, EventArgs ea)` : Gestisce la creazione del dialogo di settaggio del nome dei posti, visualizzato al doppio click sul nome stesso.
- (M) `void PlaceNTokensCapacityOnDoubleClick(object obj, EventArgs ea)` : Gestisce la creazione del dialogo di settaggio di marche e capacità di un posto, visualizzato al doppio click su di esso.

- (M) `void TransitionNameLabelOnDoubleClick (object obj, EventArgs ea)` : Gestisce la creazione del dialogo di settaggio del nome delle transizioni, visualizzato al doppio click sul nome stesso.
- (M) `int[] InitDialogA(Form dialog, string defaultText1, string defaultText2, int defaultWidth, Point defaultLocation)` : Inizializza il dialogo utilizzato per settare i pesi degli archi.
- (M) `void ArcWeightsLabelOnDoubleClick(object obj, EventArgs ea)` : Gestisce la creazione del dialogo di settaggio dei pesi in marche e probabilistici degli archi, visualizzato al doppio click sull'etichetta che li rappresenta.
- (M) `void Print(float scale)` : Stampa la PN sulla stampante predefinita, utilizzando il fattore di scala passato come parametro.

7.3.9 GRAPHICPLACE

- `GraphicPlace(Place p)` : Costruttore della classe, crea la controparte grafica del posto passato come parametro.
- (C) `Place ReferredPlace` : Il posto di cui l'oggetto è rappresentazione grafica.
- (C) `bool isInExecution` : Indica se la PN di cui fa parte il posto è in fase di esecuzione.
- `event EventHandler PlaceLocationChanged` : Evento invocato quando la posizione del posto viene modificata.
- (C) `Label nameLabel` : Etichetta che visualizza il nome del posto.
- (C) `Point oldPos` : Durante lo spostamento del posto tiene traccia della posizione precedente.
- (C) `int nameLabelOldAngle` : Durante la rotazione dell'etichetta del nome tiene traccia dell'angolo precedente.
- (C) `int NameLabelAngle` : Indica l'angolo di visualizzazione dell'etichetta del nome del posto. L'angolo è misurato in senso orario; con valore 0 si indica la posizione centrata sopra il cerchio del posto.
- (C) `bool bTracking` : Viene posto a `true` durante lo spostamento del posto.
- (C) `bool nameLabelBTracking` : Viene posto a `true` durante la rotazione dell'etichetta del nome.

- (C) `bool Selected` : Indica se il posto è selezionato.
- (C) `Point Center` : Indica il centro del cerchio che rappresenta il posto.
- (C) `int Radius` : Indica il raggio del cerchio che rappresenta il posto.
- (C) `int PenWidth` : Indica lo spessore del tratto del cerchio che rappresenta il posto. Il valore minimo è 2.
- (C) `bool ArcCreationInProgress` : Impostato esternamente a `true` se è in corso il disegno di un arco della PN.
- (M) `void Refresh()` : Effettua l'override del metodo `Refresh`; viene chiamato quando il posto è creato, ridimensionato, o il suo nome cambia.
- (M) `void OnPaint(PaintEventArgs pea)` : Effettua l'override del metodo `OnPaint`, disegnando il posto graficamente.
- (M) `void OnMouseDown(MouseEventArgs mea)` : Utilizzato per iniziare l'operazione di spostamento del posto.
- (M) `void OnMouseUp(MouseEventArgs mea)` : Utilizzato per terminare l'operazione di spostamento del posto.
- (M) `void OnMouseMove(MouseEventArgs mea)` : Durante lo spostamento del posto disegna la traccia.
- (M) `void NameLabelOnMouseDown(object obj, MouseEventArgs mea)` : Utilizzato per iniziare l'operazione di rotazione dell'etichetta del nome del posto.
- (M) `void NameLabelOnMouseUp(object obj, MouseEventArgs mea)` : Utilizzato per terminare l'operazione di rotazione dell'etichetta del nome del posto.
- (M) `void NameLabelOnMouseMove(object obj, MouseEventArgs mea)` : Durante la rotazione dell'etichetta del nome del posto disegna la sua traccia.
- (M) `void OnClick(EventArgs ea)` : Gestisce l'evento `Click` per selezionare e deselezionare il posto.
- (M) `void OnMouseWheel(MouseEventArgs mea)` : Quando il posto è selezionato consente di cambiarne il raggio agendo sulla rotellina del mouse.
- (C) `Point nameLabelCenter` : Proprietà di sola lettura che fornisce le coordinate del centro dell'etichetta del nome.

7.3.10 GRAPHICTRANSITION

- `GraphicTransition(Transition t)` : Costruttore della classe, crea la controparte grafica della transizione passata come parametro.
- (C) `Transition ReferredTransition` : La transizione di cui l'oggetto è rappresentazione grafica.
- (C) `bool isInExecution` : Indica se la PN di cui fa parte la transizione è in fase di esecuzione.
- `event EventHandler TransitionLocationChanged` : Evento invocato quando la posizione della transizione viene modificata.
- (C) `Label nameLabel` : Etichetta che visualizza il nome della transizione.
- (C) `Point oldPos` : Durante lo spostamento della transizione tiene traccia della posizione precedente.
- (C) `int nameLabelOldAngle` : Durante la rotazione dell'etichetta del nome tiene traccia dell'angolo precedente.
- (C) `int NameLabelAngle` : Indica l'angolo di visualizzazione dell'etichetta del nome della transizione. L'angolo è misurato in senso orario; con valore 0 si indica la posizione centrata sopra la transizione.
- (C) `bool bTracking` : Viene posto a `true` durante lo spostamento delle transizione.
- (C) `bool nameLabelBTracking` : Viene posto a `true` durante la rotazione dell'etichetta del nome.
- (C) `bool Selected` : Indica se la transizione è selezionata.
- (C) `Point Center` : Indica il centro del rettangolo che rappresenta la transizione.
- (C) `int LongSide` : Indica la misura del lato maggiore del rettangolo che rappresenta la transizione[24].
- (C) `int ShortSide` : Indica la misura del lato minore del rettangolo che rappresenta la transizione[24].
- (C) `int PenWidth` : Indica lo spessore del tratto del rettangolo che rappresenta la transizione. Il valore minimo è 2.
- (C) `bool Horizontal` : Vale `true` se la transizione è orizzontale[25].

[24] Il settaggio del lato maggiore/minore del rettangolo che rappresenta la transizione comporta automaticamente il riscalamento dell'altro lato.

[25] Il settaggio delle proprietà `Horizontal` e `Vertical` comporta automaticamente il settaggio opposto della proprietà contraria.

- (C) `bool Vertical` : Vale `true` se la transizione è verticale[25].
- (C) `bool ArcCreationInProgress` : Impostato esternamente a `true` se è in corso il disegno di un arco della PN.
- (M) `void Refresh()` : Effettua l'override del metodo `Refresh`; viene chiamato quando la transizione è creata, ridimensionata, o il suo nome cambia.
- (M) `void OnPaint(PaintEventArgs pea)` : Effettua l'override del metodo `OnPaint`, disegnando la transizione graficamente.
- (M) `void OnMouseDown(MouseEventArgs mea)` : Utilizzato per iniziare l'operazione di spostamento della transizione.
- (M) `void OnMouseUp(MouseEventArgs mea)` : Utilizzato per terminare l'operazione di spostamento della transizione.
- (M) `void OnMouseMove(MouseEventArgs mea)` : Durante lo spostamento della transizione disegna la traccia.
- (M) `void NameLabelOnMouseDown(object obj, MouseEventArgs mea)` : Utilizzato per iniziare l'operazione di rotazione dell'etichetta del nome della transizione.
- (M) `void NameLabelOnMouseUp(object obj, MouseEventArgs mea)` : Utilizzato per terminare l'operazione di rotazione dell'etichetta del nome della transizione.
- (M) `void NameLabelOnMouseMove(object obj, MouseEventArgs mea)` : Durante la rotazione dell'etichetta del nome della transizione disegna la sua traccia.
- (M) `void OnClick(EventArgs ea)` : Gestisce l'evento `Click` per selezionare e deselezionare la transizione.
- (M) `void OnDoubleClick(EventArgs ea)` : Gestisce l'evento `DoubleClick` per cambiare l'orientamento della transizione (orizzontale/verticale).
- (M) `void OnMouseWheel(MouseEventArgs mea)` : Quando la transizione è selezionata consente di cambiarne le dimensioni agendo sulla rotellina del mouse.
- (C) `Point nameLabelCenter` : Proprietà di sola lettura che fornisce le coordinate del centro dell'etichetta del nome.

7.3.11 GRAPHICARC

- `GraphicArc()` : Costruttore della classe.
- `Arc ReferredArc` : L'arco di cui l'oggetto è rappresentazione grafica.
- (C) `bool isInExecution` : Indica se la PN di cui fa parte l'arco è in fase di esecuzione.
- (C) `Label weightsLabel` : Etichetta che visualizza i pesi dell'arco.
- (C) `Point oldPos` : Durante lo spostamento di un punto intermedio dell'arco tiene traccia della sua posizione precedente.
- (C) `int draggedPointIndex` : Indica, all'interno della lista dei punti intermedi, l'indice del punto in corso di spostamento.
- (C) `int weightsLabelOldAngle` : Durante la rotazione dell'etichetta dei pesi dell'arco tiene traccia dell'angolo precedente.
- (C) `int WeightsLabelAngle` : Indica l'angolo di visualizzazione dell'etichetta dei peso dell'arco. L'angolo è misurato in senso orario; con valore 0 si indica la posizione centrata sopra il centro di rotazione.
- (C) `bool bTracking` : Viene posto a `true` durante lo spostamento di un punto intermedio dell'arco.
- (C) `bool weightsLabelBTracking` : Viene posto a `true` durante la rotazione dell'etichetta dei pesi dell'arco.
- (C) `bool Selected` : Indica se l'arco è selezionato.
- (C) `bool ArcCreationInProgress` : Impostato esternamente a `true` se è in corso il disegno di un arco della PN.
- (C) `int PenWidth` : Indica lo spessore del tratto dell'arco. Il valore minimo è 2.
- (C) `ArrayList Points` : Lista delle coordinate dei punti intermedi dell'arco.
- (C) `float Tension` : Indica il grado di curvatura della spline che collega i punti intermedi dell'arco. Può variare tra 0 e 1, con 0 che indica una spezzata semplice.
- (M) `void Refresh()` : Effettua l'override del metodo `Refresh`; viene chiamato quando l'arco è creato, un suo punto intermedio viene spostato, o un suo peso cambia.
- (M) `void OnPaint(PaintEventArgs pea)` : Effettua l'override del metodo `OnPaint`, disegnando l'arco graficamente.
- (M) `void OnMouseDown(MouseEventArgs mea)` : Utilizzato per iniziare lo spostamento di un punto intermedio dell'arco.

- (M) `void OnMouseUp(MouseEventArgs mea)` : Utilizzato per terminare lo spostamento di un punto intermedio dell'arco.
- (M) `void OnMouseMove(MouseEventArgs mea)` : Durante lo spostamento di un punto intermedio disegna la traccia dell'arco.
- (M) `void WeightsLabelOnMouseDown(object obj, MouseEventArgs mea)` : Utilizzato per iniziare l'operazione di rotazione dell'etichetta dei pesi dell'arco.
- (M) `void WeightsLabelOnMouseUp(object obj, MouseEventArgs mea)` : Utilizzato per terminare l'operazione di rotazione dell'etichetta dei pesi dell'arco.
- (M) `void WeightsLabelOnMouseMove(object obj, MouseEventArgs mea)` : Durante la rotazione dell'etichetta dei pesi dell'arco disegna la sua traccia.
- (M) `void OnClick(EventArgs ea)` : Gestisce l'evento `Click` per selezionare e deselezionare l'arco.
- (M) `void OnDoubleClick(EventArgs ea)` : Gestisce l'evento `DoubleClick` per aggiungere o rimuovere punti intermedi dell'arco.
- (M) `void OnMouseWheel(MouseEventArgs mea)` : Quando l'arco è selezionato consente di cambiarne la proprietà `Tension` agendo sulla rotellina del mouse.
- (C) `Point weightsLabelCenter` : Proprietà di sola lettura che fornisce le coordinate del centro dell'etichetta dei pesi dell'arco.

7.3.12 TIMELINE

- `TimeLine(PetriNet PN, bool compile)` : Costruttore della classe. Espande la PN passata come parametro sostituendo i sottonodi con le relative reti. Il parametro `compile` indica se i documenti MX presenti in posti e transizioni devono essere processati.
- (C) `ArrayList timeLine` : Lista dei passi successivi dell'esecuzione della PN.
- (C) `long CurTime` : Contiene il tempo in vtus trascorso dall'inizio dell'esecuzione della PN.

- `TimeLineStep this[int index]` : Indicizzatore che consente di riferirsi a `TimeLine[n]` per indicare il passo di indice `n` nella sequenza di esecuzione della PN.
- (M) `void Expand(PetriNet toExpandPN, PetriNet globalPN)` : Espande `toExpandPN` sostituendo i sottonodi con le relative reti. `globalPN` contiene la PN espansa.
- (C) `bool compile` : Vale `true` se i documenti MX della PN devono essere processati.
- (M) `void ScaleVtus(XmlElement xRoot, int scale)` : Scala i vtus della spine del documento MX avente radice `xRoot` del fattore di scala specificato.
- (C) `int Count` : Restituisce il numero di passi della sequenza di esecuzione della PN.
- (M) `int AddStep()` : Esegue un passo nell'esecuzione della PN. Restituisce il numero dell'ultimo passo aggiunto[26].
- (M) `bool RemoveSteps(int firstIndex, int count)` : Rimuove `count` passi dell'esecuzione della PN, partendo da `firstIndex`.
- (C) `XmlDocument outputDoc` : Documento MX generato dall'esecuzione della PN.
- (C) `int NumMX` : Numero di documenti MX processati durante l'esecuzione[27].
- (M) `void OnPlaceExecution(object sender, EventArgs ea)` : Intercetta l'evento `Execution` degli oggetti `Place`, processando (se `compile` è settato a `true`) il documento MX associato, e trasferendolo alle transizioni in uscita non aventi files MX associati.
- (M) `void OnTransitionExecution(object sender, EventArgs ea)` : Intercetta l'evento `Execution` degli oggetti `Transition`, trasferendo ai posti in uscita il documento MX associato.
- (M) `void UpdateIDs(XmlElement root, string prefix)` : Modifica gli id del documento MX con radice `root` anteponendo `prefix`.
- (M) `void AppendElements(string elPrefix, string elName, XmlElement inputRoot, XmlDocument outputDoc)` : Durante il

[26] Nel metodo `ExecuteStep` della classe `PetriNet` il numero del passo restituito viene confrontato con il numero del passo precedente per comprendere se la PN è in uno stato di stallo senza posti attivi, nel qual caso l'esecuzione è conclusa.

[27] Agli id degli elementi contenuti negli MX processati durante l'esecuzione della PN viene aggiunto il prefisso "`mx`*n*`_`", dove n corrisponde a `NumMX`.

processing di un posto, recuperando i valori dall'MX avente radice `inputRoot`, appende in `outputDoc` l'elemento specificato a quelli dello stesso tipo.

- (M) `void CopyMXData(XmlElement inputRoot, XmlDocument outputDoc)` : Copia gli elementi MX dal documento di input con radice `inputRoot` nel documento `outputDoc`.

7.3.13 TIMELINESTEP

- `TimeLineStep(PetriNet pn, TokensMap tm)` : Costruttore della classe. Imposta i campi con i parametri passati.
- (C) `PetriNet PetriNet` : Indica la PN in esecuzione.
- (C) `TokensMap tokensMap` : Oggetto che contiene la lista delle marche attive.
- (C) `Places PlacesBak` : Contiene una copia di backup dei posti della rete per consentire di ripristinare lo stato della PN ad un passo presecendemente eseguito.
- (C) `TokensMap TokensMapBak` : Copia di backup della lista delle marche attive.
- (C) `long CurTime` : Copia di backup del tempo di esecuzione trascorso.
- (C) `ArrayList AltOrConflTransitions` : Lista delle transizioni in alternativa o in conflitto.
- (M) `void executeTransition(Transition t)` : Esegue la transizione passata come parametro, trasferendo le marche dai posti in entrata a quelli in uscita.
- (M) `void checkStatus(Place p, transitionStatus status)` : Verifica l'esistenza di transizioni aventi lo stato specificato in ingresso o uscita al posto `p`.
- (M) `bool isTransitionExecutable(Transition t)` : Restituisce `true` se la transizione specificata può scattare.
- (M) `bool fire()` : Vengono fatte scattare tutte le transizioni possibili.
- (M) `long Execute()` : Esegue lo step corrente dell'esecuzione, restituendo il tempo trascorso dallo step precedente.

7.3.14 TOKENSMAP

- `struct TokensMapRecord` : Struttura che contiene una marca attiva con informazioni relative (posto, tempo rimanente alla conclusione dell'oggetto musicale associato al posto)

- `TokensMap` : Costruttore della classe.
- (C) `ArrayList map` : Lista delle marche attive.
- `TokensMapRecord this[int index]` : Indicizzatore che consente di riferirsi a `TokensMap[n]` per indicare il record di indice `n` nella lista delle marche attive della PN.
- (M) `void Init(Places places)` : Inizializza la lista delle marche attive.
- (M) `void Add(Place p)` : Aggiunge una marca alla lista delle marche attive, recuperando le informazioni dal posto specificato.
- (M) `void Add(TokensMapRecord rec)` : Aggiunge una marca alla lista delle marche attive, copiando le informazioni dal record passato come parametro.
- (M) `long GetMinTime()` : Restituisce il tempo minore fra le marche attive.
- (M) `void FreeTokens(long deltaT)` : Toglie dalla lista delle marche attive quelle con tempo rimanente alla conclusione dell'oggetto musicale uguale a 0.
- (C) `int Count` : Proprietà di sola lettura che restituisce il numero di marche attive.
- (M) `TokensMap Clone()` : Crea un clone dello stesso oggetto `TokensMap`.

7.3.15 PNEXCEPTION

- `PNException(string Message)` : Genera un'eccezione visualizzando il messaggio specificato e specificando il namespace generante (`"AB.PetriNets"`).

7.4 ALGORITMI FONDAMENTALI

7.4.1 ESPANSIONE DELLA PN

I parametri passati al metodo `Expand` sono rispettivamente la PN da espandere e il riferimento alla rete che conterrà la PN espansa.

```
Expand(PetriNet ToExpandPN, PetriNet GlobalPN)
{
     Copia posti, transizioni ed archi da ToExpandPN a GlobalPN
     Per ogni sottonodo di ExpandPN
     {
          Carica la rete associata al sottonodo in newPN
```

```
        Expand(newPN, GlobalPN)
        Aggiorna i riferimenti degli archi connessi al
sottonodo
        facendoli puntare ai nodi di ingresso/uscita di newPN
        Pone a null i nodi di ingresso/uscita di newPN
    }
}
```

Fig. 42. Algoritmo di espansione della PN

All'uscita vengono cancellati da GlobalPN i sottonodi espansi.

7.4.2 PROCESSING DI DOCUMENTI MX

Il processing di documenti MX associati ai posti avviene nell'event handler `OnPlaceExecution` della classe `TimeLine`. La variabile `NumMX` indica il numero di documenti MX processati, mentre `outputDoc` è il documento MX globale generato dall'esecuzione della PN.

```
OnPlaceExecution
{
    Se il posto ha associato un file MX
    {
        XmlElement inputRoot = nodo radice dell'MX associato
        Aggiunge il prefisso mx_NumMX_ agli id dell'MX
        Se NumMX == 0
        {
            Inserisce tutto l'MX in outputDoc
        }
        altrimenti
        {
            Carica nell'XmlNodeList globalList gli elementi 'event'
            dell'outputDoc
            Carica nell'XmlNodeList inputList gli elementi 'event'
            del documento MX associato
            Avanza in globalList fino al raggiungimento di CurTime
            o fino alla fine della lista
            Se è stata raggiunta la fine di globalList
            {
                Aggiorna l'attributo 'timing' del
```

```
                primo evento dell'inputList a
                CurTime-(tempo totale della globalList)

                Copia la spine in outputDoc

            }

            altrimenti

            {

                Crea un XmlDocumentFragment xDocFrag da outputDoc

                Copia tutti gli eventi di outputDoc precedenti
                CurTime in xDocFrag

                Aggiorna l'attributo 'timing' del
                primo evento dell'inputList a CurTime –
                (tempo della globalList fino a CurTime)

                i = puntatore al primo evento in globalList
                successivo a CurTime

                j = puntatore al primo evento in inputList

                Fintanto che non si è raggiunta la fine di
                inputList o di globalList

                {

                    Se il timing dell'evento j di inputList
                    è minore del timing dell'evento
                    i di globalList

                    {

                        Copia l'evento j da inputList in xDocFrag

                        Aggiorna il timing dell'evento i di
                        globalList sottraendo il timing
                        dell'evento j di inputList

                        Incrementa j

                    }

                    altrimenti

                    {

                        Copia l'evento i da globalList a xDocFrag

                        Aggiorna il timing dell'evento j di
                        inputList sottraendo il timing
                        dell'evento i di globalList

                        Incrementa i

                    }

                }

                Copia gli eventi rimanenti di inputList o
                di globalList

                Sostituisci la spine di outputDoc con xDocFrag

            }

            Copia tutti gli elementi dell'MX in outputDoc

        }

        Incrementa NumMX

    }
```

```
}
```

Fig. 43. Algoritmo di processing del documento MX associato ad un posto

7.4.3 ESECUZIONE DI UNO STEP

Vedere il paragrafo successivo per la procedura di scatto delle transizioni.

Il "tempo di fine" si riferisce al campo corrispondente nella lista delle marche attive: quando un processo viene eseguito e `CurTime` incrementato, il campo viene decrementato del tempo corrispondente.

```
Execute
{
     Fa scattare tutte le transizioni abilitate
     Se non ci sono stati scatti e non ci sono posti attivi
     {
          deltaT = tempo di fine minimo tra i posti attivi
          Se deltaT > 0
               Esegue ogni posto nella lista dei posti attivi
               avente durata uguale al tempo di fine
          altrimenti
               Esegue ogni posto nella lista dei posti attivi
               avente durata = 0
          Decrementa di deltaT il tempo di fine per ogni posto
          nella lista dei posti attivi e rimuove
          quindi quelli con tempo = 0
     }
}
```

Fig. 44. Algoritmo di esecuzione di un passo

7.4.4 SCATTO DELLE TRANSIZIONI ABILITATE

Forniamo innanzitutto un riepilogo di alcune variabili utilizzate spiegandone il significato:

- `bool atLeastOneTransitionCanFire`: Viene impostata su `true` quando viene trovata la prima transizione abilitata.
- `ArrayList AltOrConfTransitions`: Lista delle transizioni in alternativa o in conflitto.
- `ArrayList AltOrConflArcs`: Lista degli archi in alternativa o in conflitto.

Vedere i paragrafi successivi per gli algoritmi relativi a `isTransitionExecutable` e per la scelta pesata tra le transizioni in alternativa o conflitto.

```
Fire
{
     atLeastOneTransitionCanFire = false
     Azzera le liste AltOrConflArcs e AltOrConflTransitions
     Per ogni posto della PN azzera i campi inTokens e outTokens
     Per ogni transizione della PN imposta il suo stato a Null
     Per ogni transizione t della PN
     {
          Se t non è eseguibile (!isTransitionExecutable(t))
               Imposta lo stato di t a NotExecutable
          altrimenti
          {
               atLeastOneTransitionCanFire = true
               Per ogni posto p in entrata a t
               {
                    Incrementa p.outTokens del peso in marche
                    dell'arco che connette p a t
                    Se p.freeTokens < p.outTokens (ovvero se non
                    ci sono abbastanza marche libere in p
                    per tutte le transizioni in uscita
                    {
                         Per ogni transizione outT in uscita a p
                         {
                              Se outT è eseguibile
                              (isTransitionExecutable(outT))
                              {
                                   Aggiunge l'arco tra p e outT a
                                   AltOrConflArcs se non già presente
                                   Switch sullo stato di outT
```

```
                    {
                        = Null: aggiunge outT a
                        AltOrConflTransitions ed imposta
                        lo stato ad Alternative
                        = Executable: aggiunge outT a
                        AltOrConflTransitions ed imposta
                        lo stato ad Alternative
                        = Conflict: imposta lo stato
                        di outT ad AlternativeConflict
                        altrimenti: imposta lo stato
                        di outT ad Alternative
                    }
                }
                altrimenti
                    Imposta lo stato di outT
                    a NotExecutable
            }
        }
    }
    Per ogni posto p in uscita a t
    {
        Incrementa p.inTokens del peso in marche
        dell'arco che connette t a p
        Se (p.inTokens+p.NTokens) > p.Capacity (ovvero se
        il posto non può ospitare tutte le marche che le
        transizioni in ingresso potrebbero fornire)
        {
            Per ogni transizione inT in entrata a p
            {
                Se inT è eseguibile
                (isTransitionExecutable(inT))
                {
                    Aggiunge l'arco tra p e inT a
                    AltOrConflArcs se non già presente
                    Switch sullo stato di inT
                    {
                        = Null: aggiunge inT a
                        AltOrConflTransitions ed imposta
                        lo stato ad Alternative
                        = Executable: aggiunge inT a
                        AltOrConflTransitions ed imposta
                        lo stato ad Alternative
                        = Alternative: imposta lo stato
                        di inT ad AlternativeConflict
                        altrimenti: imposta lo stato
                        di inT ad Conflict
```

```
                                }
                            }
                            altrimenti
                                Imposta lo stato di
                                inT a NotExecutable
                        }
                    }
                }
                Se lo stato di t è Null non ci sono alternative o
                conflitti: imposta quindi lo stato a Executable
            }
        }
        Se atLeastOneTransitionCanFire è false esce dalla procedura
        Per ogni posto p della PN
        {
            p.outTokens = p.freeTokens
            p.inTokens = p.Capacity - p.NTokens
        }
        Fintanto che ci sono transizioni in AltOrConflTransitions
        {
            Sceglie in AltOrConflTransitions la transizione t
            da far scattare
            Rimuove t da AltOrConflTransitions
            Rimuove gli archi connessi a t da AltOrConflArcs
            Esegue t
            Per ogni posto in ingresso a t controlla se esistono
            transizioni in alternativa
            Per ogni posto in uscita a t controlla se esistono
            transizioni in conflitto
        }
        Esegue tutte le transizioni aventi stato Executable
}
```

Fig. 45. Algoritmo di scatto delle transizioni abilitate

7.4.5 SCELTA PESATA DI TRANSIZIONI IN ALTERNATIVA/CONFLITTO

Una caratteristica sostanziale innovativa di questa versione di ScoreSynth rispetto alle precedenti è la possibilità di attribuire agli archi di una PN un peso probabilistico che intervenga nella scelta di transizioni in alternativa e/o in conflitto.

Per chiarezza di esposizione riportiamo il codice commentato C#, invece di fornire una versione dell'algoritmo in metalinguaggio.

```
// t contiene la transizione scelta per essere eseguita
Transition t = null;
// Somma dei pesi probabilistici degli archi in alternativa/conflitto
int TotProbWeights = 0;
foreach (Arc a in AltOrConflArcs)
     TotProbWeights += a.ProbWeight;
// Genera un intero da 0 a TotProbWeight-1
int rndInt = (new Random()).Next(TotProbWeights);
// Reimposta TotProbWeights a 0
TotProbWeights = 0;
// Trova la transizione da eseguire
for (int i = 0; i < AltOrConflArcs.Count; i++)
{
     TotProbWeights += ((Arc)AltOrConflArcs[i]).ProbWeight;
     if (rndInt < TotProbWeights)
     {
          t = ((Arc)AltOrConflArcs[i]).Transition;
          break;
     }
}
// Se tutti gli archi hanno peso probabilistico nullo sceglie
// la transizione puntata dal primo arco della lista
if (t == null)
     t = ((Arc)AltOrConflArcs[0]).Transition;
```

Fig. 46. Algoritmo di scelta fra transizioni in alternativa/conflitto

L'algoritmo è abbastanza semplice e certamente non ottimizzato in velocità: tuttavia appare inutile cercare questo tipo di caratteristica in un codice eseguito pochissime volte all'interno dell'esecuzione della rete. Interessanti studi su algoritmi di scelta pesata sono comunque reperibili in generale in [Knu98] e più in particolare in [RR92].

CAPITOLO 8

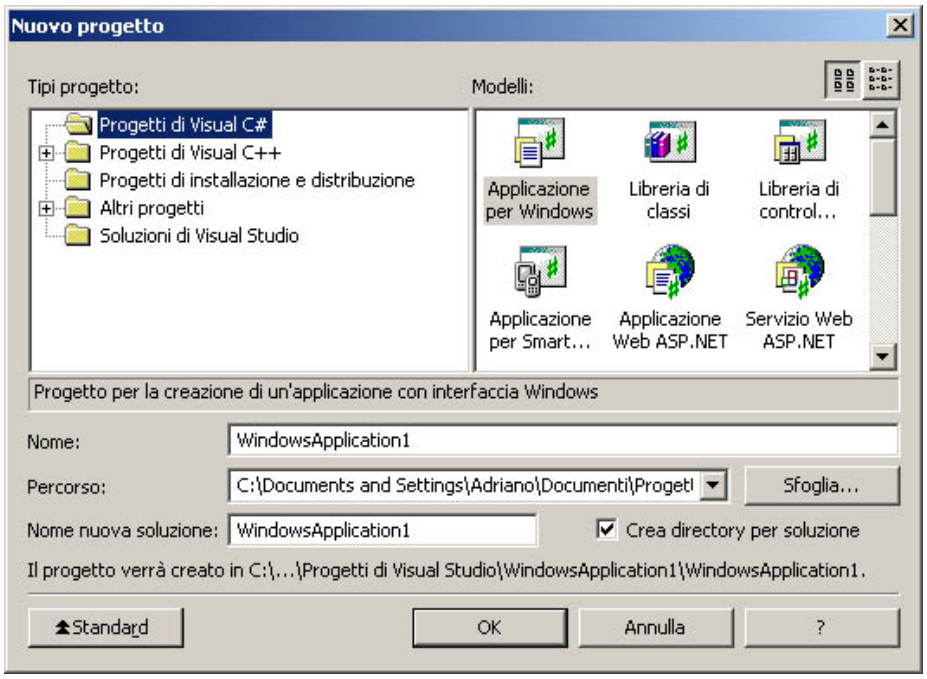
ESEMPI DI RIUTILIZZO DEL CODICE

8.1 APPLICAZIONE DI DISEGNO E STAMPA DI PNS

In questo capitolo verranno presentati due esempi che illustrano la semplicità di riutilizzo del codice di ScoreSynth, utilizzando l'ambiente Microsoft Visual Studio .NET 2003.

Nel primo esempio verrà creata un'applicazione che, semplicemente, permetta di disegnare e stampare una PN.

- Creiamo un nuovo progetto di applicazione Windows:



- In **Esplora Soluzioni** aggiungiamo ai riferimenti del progetto creato un riferimento alla dll **AB.PetriNets.dll**.
- Dalla **Casella degli strumenti** creiamo un **Panel** nel form principale.
- Con un doppio click sull'area vuota del form principale si apre la finestra del codice e il cursore si posiziona sull'event handler creato automaticamente di load del form. Qui creiamo nella classe un campo che conterrà la nostra PN:

```
AB.PetriNets.PetriNet PN;
```

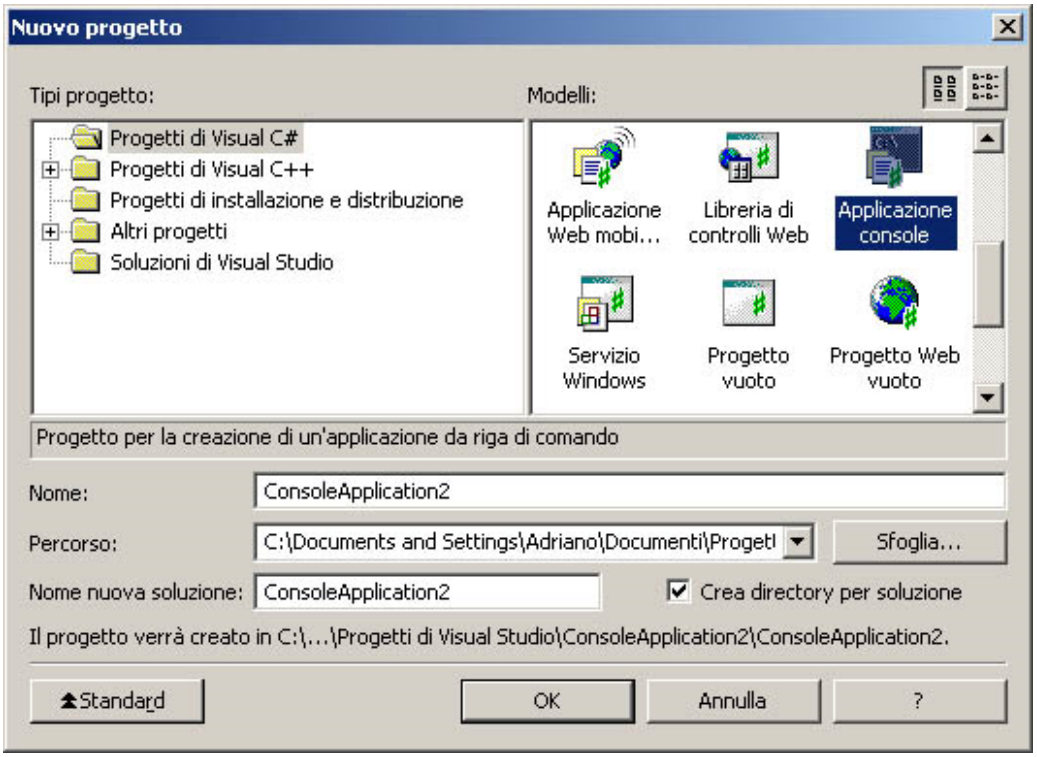
- Nell'event handler di load del form scriviamo creiamo la PN:
  ```
  PN = new AB.PetriNets.PetriNet(panel1);
  ```
  e impostiamo la grandezza a copertura dell'interno controllo **panel1**:
  ```
  PN.Graphic.Size = panel1.Size;
  ```
- Dalla **Casella degli strumenti** creiamo un **Button** sul form principale.
- Con un doppio click sul pulsante creato viene creato un event handler che gestisce il click sul pulsante, qui chiamiamo il comando di stampa della PN, con fattore di scala del 50%:
  ```
  PN.Graphic.Print(0.5f);
  ```
- Avviamo la soluzione

In pochi passaggi è stata creata un'applicazione completamente funzionante che consente di disegnare e stampare una PN.

8.2 ESECUTORE DI PNS A RIGA DI COMANDO

Nel secondo esempio verrà creata un'applicazione console che carica dal file PNML specificato una PN, la esegue, e salva l'output sul file specificato.

- Creiamo un progetto di applicazione console:



- In **Esplora Soluzioni** aggiungiamo il progetto **AB.PetriNets.csproj** alla soluzione.
- Nelle proprietà del progetto **AB.PetriNets** aggiungiamo alle **Costanti di compilazione condizionale** il simbolo **CONSOLE**. Questo fa sì che la dll compilata del progetto **AB.PetriNets** non contenga riferimenti a componenti grafici.
- Aggiungiamo ai riferimenti di **ConsoleApplication1** un riferimento alla dll del progetto **AB.PetriNets.**
- Nel metodo **Main** di **Class1** aggiungiamo:

```
AB.PetriNets.PetriNet PN = new AB.PetriNets.PetriNet(null);
PN.LoadFromFile(args[0]);
PN.Compile = true;
PN.File = args[1];
int step = 0;
while (PN.ExecuteStep(step++));
```

  In questo modo viene creata la PN senza rappresentazione grafica, viene caricato il file specificato dal primo parametro, viene impostata la generazione del file di output dell'esecuzione, viene impostato il nome del file di output, viene eseguita la rete, incrementando ogni volta lo step ed uscendo quando il metodo **ExecuteStep** restituisce **false**.
- Generiamo la soluzione.

Questi due esempi fanno capire quanto è semplice riutilizzare il codice di ScoreSynth, ed in generale la dll di gestione delle PNs.

Naturalmente negli esempi non abbiamo aggiunto funzionalità agli elementi esistenti. Ciò è comunque molto semplice, grazie alla modularità del progetto ed alla tecnologia ad oggetti.

CAPITOLO 9

MX, SCORESYNTH E POLIMETRIA

9.1 GRANULARITÀ DEI VTUS

9.1.1 IL PROBLEMA

Nel capitolo sugli esempi applicativi di ScoreSynth gli esempi di files MX associati ai posti delle PNs erano costruiti ad hoc; questo ha permesso di tralasciare un problema insito nella codifica stessa del formato MX in rapporto alle operazioni eseguite da ScoreSynth: vediamo di cosa si tratta.

In un file MX tutti gli eventi musicali sono "indicizzati" nella spine, una struttura iniziale del file che associa ad ogni evento un nome (**id**), un parametro spaziale (**hpos**, in vpxs) e un parametro temporale (**timing**, in vtus). Questi due parametri indicano quanto dista l'evento in oggetto rispettivamente in spazio ed in tempo dall'evento precedente.

Nel processing dei files MX da parte di ScoreSynth, il parametro **hpos** è lasciato inalterato, anche in considerazione del fatto che ulteriori modifiche in questo ambito sono in corso di definizione da parte del team di sviluppo del formato.

Il parametro **timing**, invece, viene massicciamente preso in considerazione, visto che in caso di sovrapposizione temporale di frammenti MX, gli eventi delle spines devono essere interfogliati e, visto che l'attributo è relativo all'evento precedente, esso deve essere ricalcolato in caso di inserimento di un evento tra due già presenti[28].

Il problema che emerge è che i vtus sono unità di misura relative: nelle prime stesure del formato MX non esisteva un parametro che "attualizzasse" i vtus, la cui granularità era scelta dal compilatore[29] del documento.

Vediamo con un esempio quello che comporta questa mancanza di attualizzazione dei vtus: supponiamo che due compilatori **A** e **B** debbano codificare in MX la stessa partitura,

[28] Per una descrizione dettagliata del procedimento si veda il capitolo dedicato al formato MX.
[29] Per "compilatore" nella parte seguente si intende la persona o l'applicazione che crea un documento MX.

costituita semplicemente da due battute di pausa da 4/4: il compilatore **A** produce un file MX con la spine seguente:

```
<spine>
     <event id="clef_0" hpos="4"/>
     <event id="timesig_0" hpos="9"/>
     <event id="rest_0" timing="0" hpos="9"/>
     <event id="barline_0" hpos="10"/>
     <event id="rest_1" timing="100" hpos="9"/>
     <event id="barline_1" hpos="10"/>
</spine>
```

Il secondo compilatore invece, scegliendo una granularità di vtus diversa produce la spine:

```
<spine>
     <event id="clef_0" hpos="4"/>
     <event id="timesig_0" hpos="9"/>
     <event id="rest_0" timing="0" hpos="9"/>
     <event id="barline_0" hpos="10"/>
     <event id="rest_1" timing="40960" hpos="9"/>
     <event id="barline_1" hpos="10"/>
</spine>
```

Come si può vedere l'attributo **timing** dell'evento **rest_1** (la seconda pausa) nel primo caso vale 100 mentre nel secondo 40960. E' da notare che nel resto del file MX non esiste nulla che possa far capire quanto vale un vtu.

Questo approccio non crea grossi problemi in fase di compilazione del documento MX, ma, se ad esempio un'applicazione dovesse tradurre in file MIDI il documento MX, occorrerebbe che l'utente specifichi quanto far durare un vtu.

Il problema incontrato in ScoreSynth, che non produce un output udibile dei files MX presenti nelle PNs, è legato al fatto che l'applicazione offre la possibilità, eseguendo una rete creata, di sovrapporre files MX. In questo caso la semplice sovrapposizione dei frammenti appena visti, senza un opportuno trattamento del problema della granularità dei vtus, non produrrebbe una sovrapposizione di parti con ognuna due battute di pausa dello stessa durata, come auspicabile, ma una parte che, temporalmente, durerebbe più di 400 volte l'altra, qualsiasi sia la durata attualizzata.

9.1.2 UNA SOLUZIONE

Per cercare di ovviare al problema, su nostro suggerimento è stato introdotto nella codifica MX un attributo, associato all'elemento **time_signature**, sottoelemento di **staff**, chiamato **vtu_amount**, che indica in quanti vtus è suddivisa una battuta[30].

In questo modo si risolverebbe il problema presentatosi nell'esempio precedente, visto che i vtus della spine potrebbero essere scalati in modo da far collimare i frammenti MX sovrapposti.

Anche con questa soluzione rimangono però altri tipi di problemi, relativi alla sovrapposizione di frammenti MX con indicazioni metriche diverse.

9.2 POLIMETRIA

9.2.1 INTRODUZIONE

Prima di approfondire il problema esposto alla fine del precedente paragrafo, poniamo alcune precisazioni terminologiche.

Riprendendo la terminologia adottata da Rudziński [Rud93], con *multimetria* si intende un andamento orizzontale della musica sottoposto a cambiamenti successivi nell'ambito dell'organizzazione metrica, mentre con *polimetria* si intende una sovrapposizione di flussi metrici diversi ma simultanei.

Per anni la multimetria è stata presente in tutta la musica, sia in forma palese che più o meno celatamente. Nella musica barocca si trovano frequentemente esempi di multimetria palese: si prenda ad esempio la melodia di corale di Hans Leo Hassler (1608) in Fig. 47.

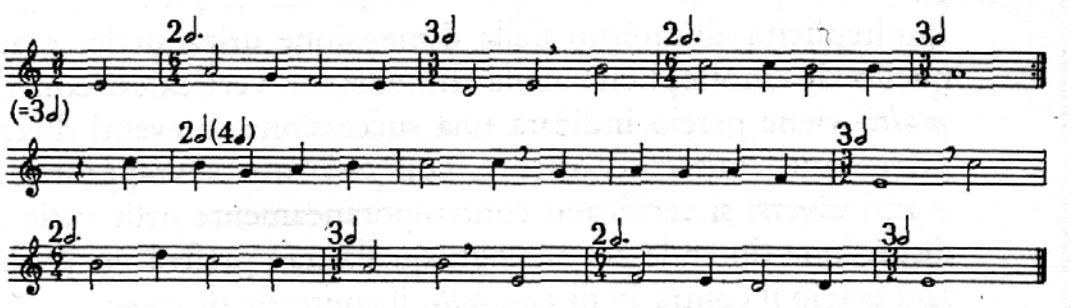


Fig. 47. Hassler

[30] Questa soluzione non risolve il problema dell'attualizzazione temporale necessaria nel caso di esecuzione MIDI di un documento MX, ad esempio. Questo perchè la codifica MX si propone di descrivere logicamente gli elementi partiturali di un frammento musicale, svincolandoli da una esecuzione qualsiasi del materiale.

Durante il periodo classico e romantico i compositori facevano largo uso della multimetria nascosta: quella, cioè, non evidenziata da indicazioni di battuta.

All'inizio del XIX secolo fa la sua ricomparsa la multimetria palese, dichiarata da indicazioni di battuta, che negli anni successivi si presenta sempre più frequentemente: prendendo ad esempio la scena *Jeux des cités rivales* della *Sacre du printemps* di Stravinskij, si contando 36 cambiamenti metrici nel corso di 76 battute, mentre nell'*Adoration de l'Elue* ci sono 48 cambiamenti in 58 battute.

Nella musica più recente spesso l'indicazione metrica è addirittura omessa, specialmente in pezzi per strumento solo o per piccoli organici.

La *polimetria*, come si è detto, indica invece una sovrapposizione simultanea di diversi flussi metrici. Uno dei primi e più rappresentativi esempi è il finale del primo atto del *Don Giovanni* di Mozart, in cui ogni stratificazione metrica ha la propria indicazione e la polimetria caratterizza un lungo tratto musicale.

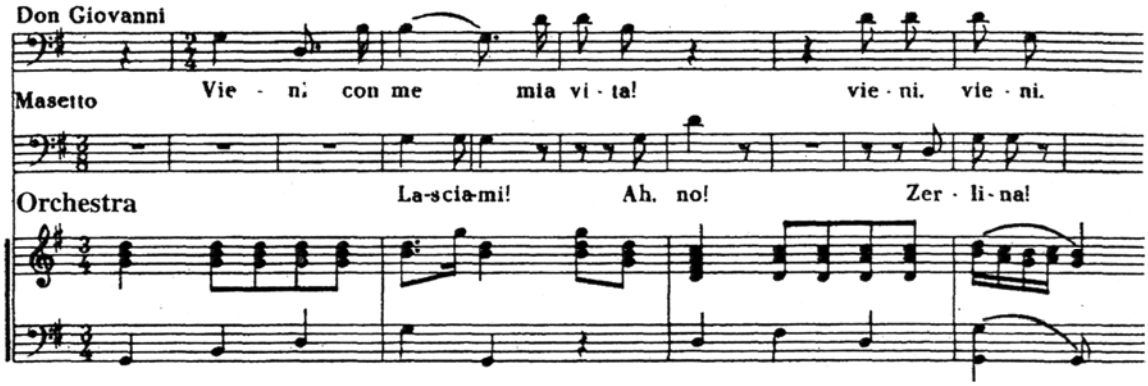


Fig. 48. Mozart, *Don Giovanni*

Per anni non vennero usate se non in casi sporadici strutture di questo genere.

La polimetria palese si riaffaccia nei compositori moderni, che ne fanno larghissimo uso: Rimskij-Korsakov nella *Terza Sinfonia* sovrappone battute di 5/4 e 2/2 (Fig. 49), Ravel nel *Trio per pianoforte, violino e violoncello* battute di 4/2 e 3/4, Webern nella *Seconda Cantata* op. 31 battute di 2/2, 3/2 e 4/2.

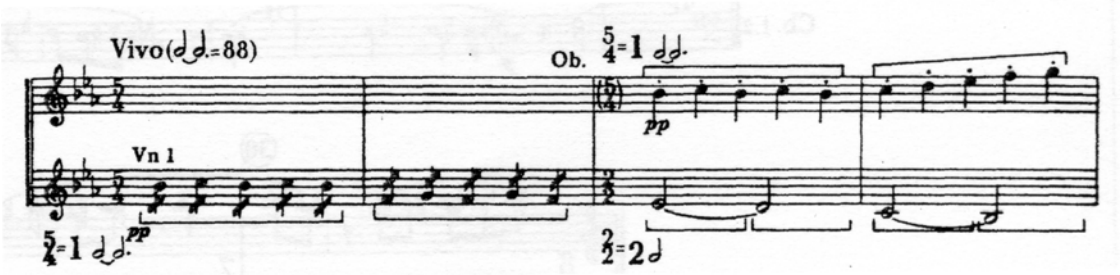


Fig. 49. Rimskij-Korsakov, *Terza Sinfonia*

Si arriva alla *polimetria asincronica*, quella cioè in cui non è prevista l'esistenza di alcuna fase di sincronismo tra le varie voci: nel *Quartetto per archi* di Lutosławski gli strumentisti suonano la loro parte senza curarsi di quella dei colleghi, rincontrandosi solo al termine di ogni sezione in cui è divisa la composizione, quando chi ha già terminato la propria parte aspetta gli altri per attaccare la sezione successiva.

Per strutture polimetriche particolarmente complesse spesso il compositore richiede la presenza di più di un direttore d'orchestra (Ives, Lutosławski, Stockhausen).

9.2.2 LA POLIMETRIA IN SCORESYNTH

La multimetria e la polimetria nella maggioranza dei casi non presentano grossi problemi per la codifica in formato MX[31]. Il compilatore di un documento MX in cui è presente la multimetria palese può semplicemente cambiare il parametro **vtu_amount** dell'indicazione metrica per semplificare il processo di codifica[32]. Anche in presenza di polimetria occorre soltanto trovare un valore di **vtu_amount** che consenta una granularità abbastanza piccola da codificare efficacemente tutti gli eventi delle indicazioni metriche sovrapposte.

Il problema in ScoreSynth interviene durante la sovrapposizione di documenti MX, che, come accennato sopra, diversi compilatori possono aver redatto scegliendo granularità diverse, o, più semplicemente che non sono stati codificati con lo scopo preventivo di essere sovrapposti. La soluzione al problema è già stata esposta, e consiste nel riscalare una o più spines dei documenti prima di effettuarne la sovrapposizione.

Una domanda però si presenta in questo caso: come decidere il fattore di scala?

Nell'esempio all'inizio del capitolo era chiaro che la scala doveva consentire semplicemente di sovrapporre le battute una ad una; supponiamo invece che si presentino due frammenti MX da sovrapporre, l'uno con indicazione metrica 5/4 e l'altra con 4/4: in questo caso è esatto sovrapporre le battute una ad una?

La risposta non è univoca: dipende dal volere del compositore, o, in questo caso, da chi ha modellato la PN che porta alla sovrapposizione del materiale.

In generale si sono individuate tra le scelte possibili di "riscalamento" tre casi base:

---

[31] Nell'ultimo secolo si sono sviluppati, insieme alla multimetria ed alla polimetria usate spesso contemporaneamente, i fenomeni della *notazione aleatoria,* della *notazione gestuale* e del *grafismo* [Azz97], che invece rappresentano problemi quasi insormontabili per una codifica efficace in formato MX.

[32] Per esempio, se si succedono delle battute di 2/4 e di 3/4 e si desidera avere una granularità che consenta di codificare almeno delle semibiscrome si potrebbe utilizzare per i 2/4 un **vtu_amount** = 128 e per i 3/4 un **vtu_amount** = 192 (o, in alternativa, mantenere invariato un **vtu_amount** = 384).

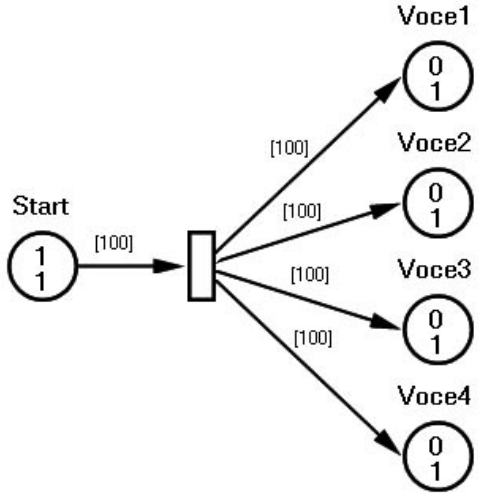
- Le sovrapposizioni possono essere eseguite "per battuta", facendo collimare quindi ogni battuta di ogni file da sovrapporre con quella degli altri files (es.: Fig. 49)
- Le sovrapposizione possono essere eseguite "per valore", facendo collimare le figure con lo stesso valore metrico (es.: prima, terza e quarta riga in Fig. 48)
- La sovrapposizione è più complessa e devono essere definiti degli strumenti per specificarne le modalità.

Per consentire un ampio ventaglio di scelte, in ScoreSynth è previsto il parametro intero **Vtu Scale**, associato a posti e transizioni. Il valore del parametro indica la scala che l'applicazione applica preventivamente agli eventi dei files MX associati ai nodi. In questo modo l'utente può specificare precisamente quale tipo di sovrapposizione ottenere. Il parametro è impostato di default a 1; questo consente di lavorare con files MX creati con lo stesso valore di **vtu_amount** senza intervenire in cambiamenti di scala non necessari.

Nei menu sono presenti due voci che aiutano l'utente nella scelta di **Vtu Scale**: **Align to measures** e **Align to values**, presenti in **Net Edit**. Scegliendo questi comandi, ScoreSynth calcola automaticamente, tenendo conto del parametro[33] **vtu_amount** di ogni file MX della PN, il fattore di scala da applicare ad ogni frammento MX, in linea con i primi due casi sopraesposti.

Chiariamo il funzionamento di questa opzione attraverso un esempio: supponiamo di voler codificare il frammento del *Don Giovanni* in Fig. 48, e di avere a disposizione le quattro voci già codificate in MX su files diversi. La PN che sovrappone le 4 voci è semplicemente quella proposta a sinistra: allegando ai posti le varie voci possiamo poi controllare i valori dei **vtu_amount** di ogni file dalla proprietà di sola lettura del posto: **Vtu Amounts**, che fa apparire la lista delle indicazioni metriche presenti nel file con i relativi **vtu_amount**. Supponiamo che le 4 voci abbiano i seguenti valori:

- Voce1: 2/4 : 192
- Voce2: 3/8 : 24



[33] L'applicazione controlla solo il primo **vtu_amount** presente in ogni f polimetria e multimetria presenti nella stessa PN. Ad esempio: se un frammento sovrapposto ad un altro file con 2 battute in 4/4, si potrebbero voler far collima questo caso occorrerebbe un fattore di scala diverso per ogni indicazione met preferito non appesantire inutilmente il lavoro richiesto all'utente, demandando l'eventuale creazione di una soluzione ben studiata a versioni successive di ScoreSynth.

- Voce3: 3/4 : 4608
- Voce4: 3/4 : 144

Aiutandoci con il menu **Align to values**, ScoreSynth imposta i 4 valori di **Vtu Scale** dei posti rispettivamente a 16, 96, 1, 32. Il valore relativo alla seconda voce deve però essere modificato, in quanto nel brano i suoi 3/8 corrispondono ad 1/4 delle altre voci. In questo caso basta dividere 96 per 1.5 (3/8 diviso per 1/4), ed impostare manualmente il valore di **Vtu Scale** del posto **Voce2** a 64.

9.2.3 CONCLUSIONI

Vogliamo far notare quali fattori positivi sono stati considerati per la soluzione scelta:

- L'utente che crea da solo i files MX da utilizzare nelle PNs modellate decide in fase di compilazione del documento MX i valori **vtu_amount**, in modo da ottenere le eventuali sovrapposizioni scelte. Nel caso di mancanza di polimetria può semplicemente utilizzare lo stesso valore di **vtu_amount** per tutti i frammenti MX, e non intervenire in ScoreSynth su **Vtu Scale**.
- In caso di polimetria semplice, cioè quando devono essere fatte collimare le battute una ad una oppure i valori metrici, l'utente può utilizzare i menu **Align to measures** e **Align to values**, che compilano automaticamente i campi **Vtu Scale**.
- Nei casi più complessi, l'utente può intervenire manualmente sulle scale da applicare ai files MX dei vari posti, anche aiutandosi con la proprietà **Vtu Amounts** che lista tutti i parametri delle indicazioni metriche per ogni file.

Due ultimi appunti relativi all'algoritmo di "riscalamento":

- Nel caso venga impostato **Vtu Scale** a 0, il frammento MX non viene trattato, ed il posto ha durata 0.
- Nel caso venga scelto uno dei due modi di calcolo automatico del fattore di scala, l'algoritmo prende in considerazione tutti i frammenti MX presenti nella rete, indipendentemente dal fatto che siano poi sovrapposti in fase di esecuzione, altrimenti l'applicazione dovrebbe prima eseguire la rete per capire quando i files si sovrappongono, ma la sovrapposizione potrebbe dipendere proprio dal fattore di scala!

CAPITOLO 10

IDEE PER SVILUPPI FUTURI

In questo capitolo forniamo una serie di idee per possibili sviluppi futuri di ScoreSynth:

10.1 INTERFACCIA

- Dal punto di vista dell'interfaccia si potrebbero aggiungere funzionalità di zoom della PN, strumenti di undo, i classici comandi taglia, copia ed incolla, strumenti per l'allineamento automatico dei nodi selezionati, una serie di opzioni di stampa, la memorizzazione della disposizione delle finestre disponibili, e molto altro ancora.
- Potrebbe essere utile visualizzare l'XML corrispondente ai documenti MX associati ai nodi della PN, meglio se in forma tabellare (come in Altova XmlSpy).
- Molto utile sarebbe includere una finestra di visualizzazione notazionale dei documenti MX della PN.
- Una re-ingegnerizzazione della componente grafica di AB.PetriNets per eliminare alcuni problemi di flickering e di risposta lenta ai movimenti del mouse (Microsoft ha annunciato che dei meccanismi ad hoc sono in corso di sviluppo e dovrebbero essere inclusi nelle prossime versioni del framework .NET).

10.2 PNS

- Nell'esecuzione della PN, sarebbe utile poter inserire dei breakpoint nei nodi, raggiunti i quali l'esecuzione venga messa in pausa.
- All'inizio del lavoro di tesi sembrava assai interessante l'estensione delle PN colorate, poi tralasciata per semplicità: il passaggio a questo modello dovrebbe essere attentamente studiato.

- Nella versione di ScoreSynth di A. Sametti [Sam88] alle transizione potevano essere associati algoritmi di trasformazione degli oggetti in ingresso. Sarebbe interessante riproporre un'opzione di questo tipo, aggiornandola in modo che sia più adatta al trattamento di files MX.
- Gli archi in questa versione hanno un parametro di peso probabilistico che interviene in situazioni di alternativa/conflitto: sarebbe interessante ampliare il concetto potendo definire delle distribuzioni di probabilità da applicare in questi casi. Per esempio, Iannis Xenakis utilizza la distribuzione poissoniana per molte sue tecniche compositive [Xen92], ed altri compositori hanno teorizzato metodi simili [Lan91] [Kàr98].
- Potrebbe essere interessante ampliare il formato di salvataggio dei files delle PNs utilizzando il PNML strutturato e/o modulare.
- Potrebbe essere utile un "listato" testuale che descriva l'esecuzione della PN, in cui vengano presentati gli eventi di esecuzione come: transizioni in alternativa/conflitto, arrivo di una marca, esecuzione di un posto, scatto di una transizione, ecc…

10.3 MX

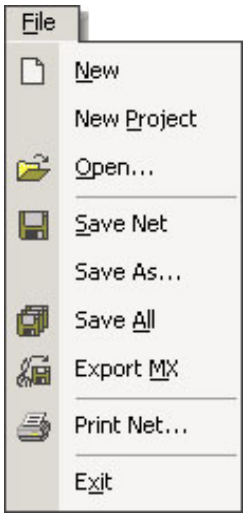
- E' doveroso l'aggiornamento delle funzionalità di ScoreSynth in relazione a future modifiche della codifica MX.
- I parametri **hpos** dei files MX vengono nella presente versione soltanto copiati inalterati in output: sarebbe utile uno studio di un loro trattamento più efficace.
- Potrebbe essere utile poter escludere scegliendoli dalla spine alcuni eventi non necessari (per esempio alcuni eventi che hanno solo un parametro spaziale).
- L'elemento layout dei documenti MX non viene processato, anche perché non ancora pienamente definito dallo standard: sarebbe interessante studiare modi di trattarlo.
- Sarebbe utile poter unire documenti MX giustapposti creando un solo elemento **staff**.
- Il processing dei documenti MX durante l'esecuzione richiede un tempo non indifferente, in caso di file di grosse dimensioni: questo è principalmente dovuto al modo di gestire l'XML da parte del framework .NET. Sarebbe utile una re-ingegnerizzazione degli algoritmi per aumentarne la velocità.
- Sarebbe utile poter trattare i casi più particolari di multimetria/polimetria, studiando una possibile soluzione definitiva.

APPENDICE A
GUIDA OPERATIVA

A.1 IL MENU PRINCIPALE

Verranno ora descritte le funzionalità associate ad ogni comando del menu principale. Si noti che tutti i pulsanti delle barre degli strumenti corrispondono a comandi presenti nel menu, quindi le spiegazioni che seguono si riferiscono anche ad essi.

A.1.1 MENU FILE



New. Crea una nuova PN, settandola come rete principale (Main Net) dell'albero gerarchico delle reti.

New Project. Chiude tutte le PNs aperte, resettando le finestre di riepilogo.

Open. Apre una PN salvata in formato PNML. Se nessuna rete era presente prima dell'apertura, la setta come rete principale. Quando la rete aperta contiene sottonodi vengono aperte anche tutte le PNs associate.

Save Net. Salva la rete visualizzata correntemente, con il nome datole in precedenza.

Save As. Salva la rete visualizzata correntemente, chiedendo il nome del file PNML.

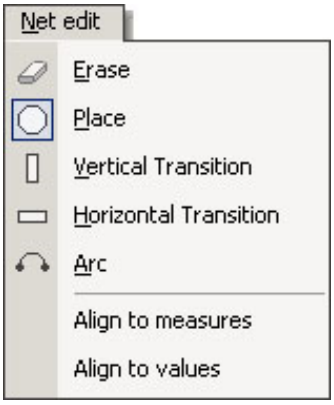
Save All. Salva tutte le reti del progetto in corso. Se alcune di esse non hanno ancora un nome, lo richiede (come per **Save As**)

Export MX. Esegue la rete principale del progetto corrente, salvando il documento MX creato su un file.

Print Net. Stampa la rete corrente sulla stampante predefinita.

Exit. Esce dall'applicazione. Se alcune reti non sono ancora state salvate un avviso viene presentato all'utente, consentendogli di farlo.

A.1.2 MENU NET EDIT



Erase. Cancella l'elemento della PN su cui si clicca.

Place. Al click sull'area di disegno viene aggiunto un posto alla PN.

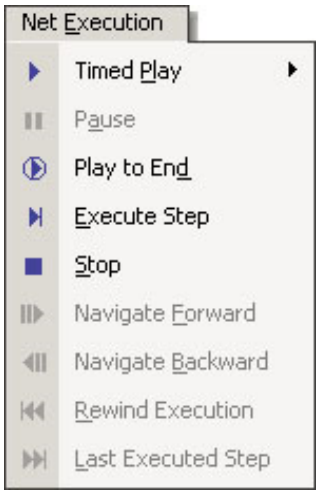
Vertical Transition. Horizontal Transition. Al click sull'area di disegno viene aggiunta una transizione verticale o orizzontale.

Arc. Viene impostata l'azione di disegno archi. Per disegnare un arco si clicca sul nodo di partenza, poi si possono aggiungere punti intermedi cliccando sull'area vuota e infine si clicca sul nodo finale.

(Per selezionare un elemento della PN si può selezionare da questo menu una voce qualsiasi tra ***Place, Vertical Transition o Horizontal Transition****, ma* ***non Arc.****)*

Align to measures. Align to values. Setta automaticamente i parametri VtuScale che si riferiscono ai files MX associati ai nodi ad un valore appropriato (si veda il capitolo 10).

A.1.3 MENU NET EXECUTION



Quando viene avviata l'esecuzione della PN con uno dei comandi di questo menu, lo sfondo della rete cambia ed è inibita la possibilità di modificarla.

Timed Play. Avvia l'esecuzione temporizzata della PN principale: è possibile scegliere il tempo che intercorre fra i passi dell'esecuzione, consentendo una visione rallentata dell'evoluzione della rete.

Pause. Blocca temporaneamente l'esecuzione temporizzata della PN.

Play to End. Esegue la PN principale completamente, senza pause tra un passo ed il successivo.

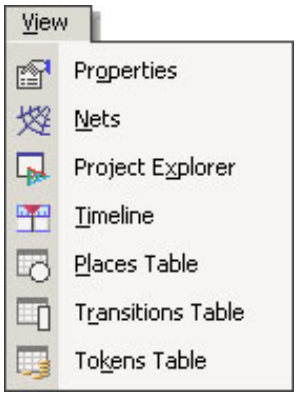
Execute Step. Viene eseguito un passo dell'esecuzione, per consentirne il debug.

Stop. Blocca l'esecuzione della PN. Ripristina la rete allo stato iniziale e ripristina le possibilità di editing.

Navigate Backward. Navigate Foreward. Durante l'esecuzione step-by-step della PN consente di spostarsi tra i passi già eseguiti. Da notare che **Navigate Foreward** non esegue un passo dell'esecuzione ex novo, ma semplicemente avanza di un passo tra quelli già eseguiti. Per spostarsi velocemente tra i passi eseguiti si utilizzino i pulsanti della barra degli strumenti, in cui è presente una lista dei passi.

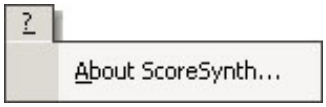
Rewind Execution. Last Executed Step. Analogo ai comandi appena precedenti, porta la rete al primo passo o all'ultimo dell'esecuzione.

A.1.4 MENU VIEW



I comandi del menu View consente di visualizzare o nascondere le finestre corrispondenti alle singole voci.

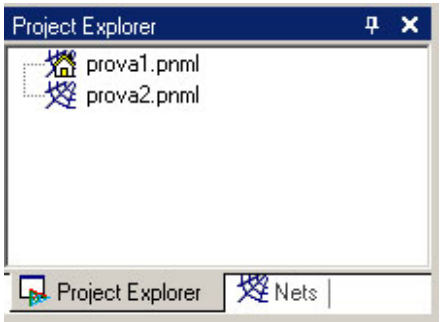
A.1.5 MENU ?



About ScoreSynth. Visualizza una schermata di informazioni sull'applicazione.

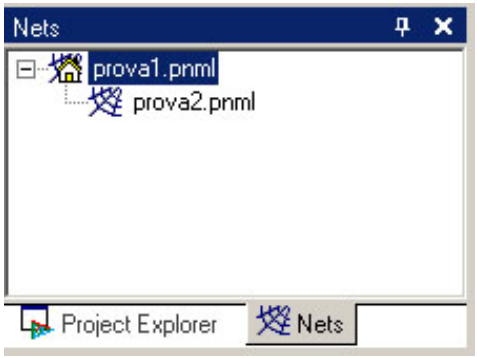
A.2 FINESTRE MOBILI

A.2.1 PROJECT EXPLORER



Project Explorer. Elenca le PNs del progetto corrente, consentendo di selezionare dal menu contestuale visualizzato alla pressione del tasto destro del mouse: selezione della rete principale (Main Net), esclusione di una rete dal progetto, creazione di un nuovo progetto, con la chiusura delle reti aperte.

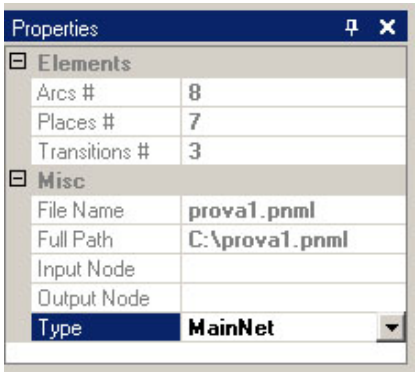
A.2.2 NETS



Nets. Visualizza l'albero gerarchico delle PN appartenenti al progetto corrente, rappresentando la struttura delle sottoreti relative a sottonodi.

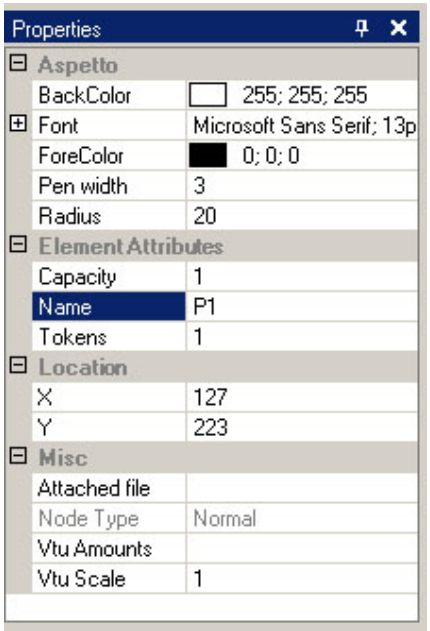
A.2.3 PROPERTIES

Il contenuto della finestra **Properties** cambia concordemente con l'elemento della rete selezionato.



Rete

- *Arcs # :* Numero degli archi della PN.
- *Places # :* Numero dei posti della PN.
- *Transitions # :* Numero delle transizioni della PN.
- *File Name :* Nome del file PNML della PN.
- *Full Path :* Percorso completo del file PNML della PN.
- *Input Node, Output Node :* Nodi di input/output di una sottorete.
- *Type :* Tipo di PN: indica se è la rete principale del progetto corrente.



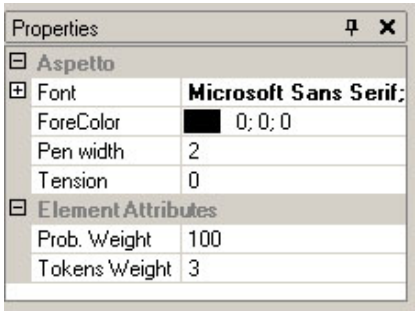
Posto - Transizione

- *BackColor, ForeColor :* Colore di sfondo e del tratto, rispettivamente.
- *Font :* E' possibile selezionare il carattere del testo (nome, marche, capacità); la grandezza è scalata automaticamente alla grandezza del posto.
- *Pen Width :* Spessore del tratto del cerchio che rappresenta il posto.
- *Radius :* Raggio del cerchio.

- *Name, Tokens, Capacity :* Nome del posto, marche e capacità *(marche e capacità non esistono fra le proprietà delle transizioni).*
- *X, Y :* Le coordinate del centro del cerchio.
- *Attached file :* Il nome del file MX associato al posto o del file PNML contenente una sottorete.
- *Node Type :* Indica se il posto è un nodo normale oppure di input o di output di una sottorete.
- *Vtu Amounts :* Lista le segnature di tempo del file MX associato, con il valore dell'elemento *vtu_amount* corrispondente.
- *Vtu Scale :* La scala da applicare ai vtus degli eventi contenuti nel file MX associato, oppure 0 se non si desidera che il posto venga eseguito.

Quando viene allegato un file MX viene visualizzata anche la proprietà Track:

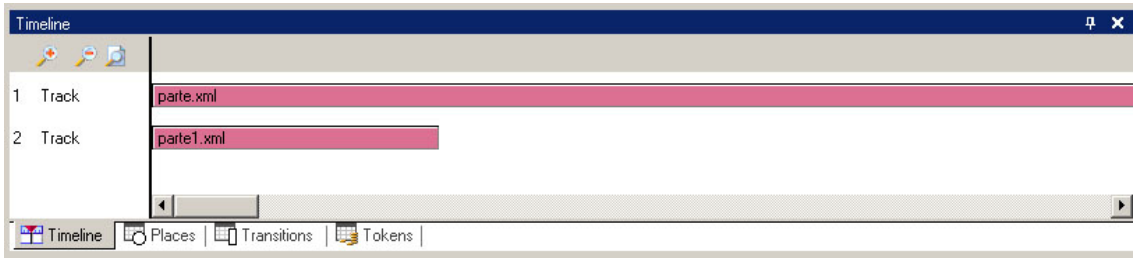
- *Track :* Indica la traccia della timeline in cui deve essere inserito l'oggetto MX eseguito. E' possibile specificare l'id della traccia, il nome e una priorità (non utilizzata in ScoreSynth).



Arco

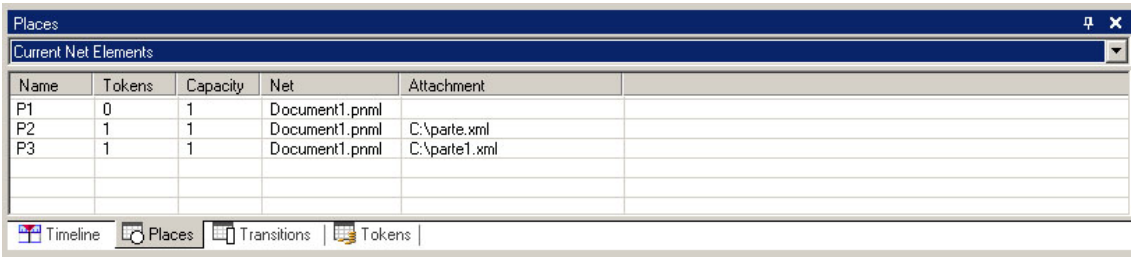
- *Font :* E' possibile selezionare il carattere dell'inscrizione dell'arco (in questo caso anche la grandezza).
- *ForeColor :* Colore del tratto dell'arco.
- *Pen Width :* Spessore del tratto dell'arco.
- *Tension :* Valore che indica la curvatura dell'arco in presenza di punti intermedi. Varia da 0 (spezzata) a 1.
- *Prob. Weight :* Peso probabilistico dell'arco.
- *Tokens Weight :* Peso in marche dell'arco.

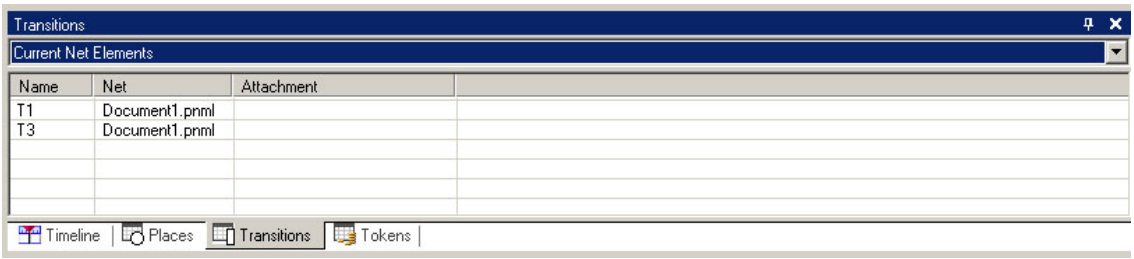
A.2.4 TIMELINE



Visualizza la timeline dell'esecuzione corrente, indicando con delle barre orizzontali gli oggetti MX eseguiti (la lunghezza della barra è proporzionale alla durata).

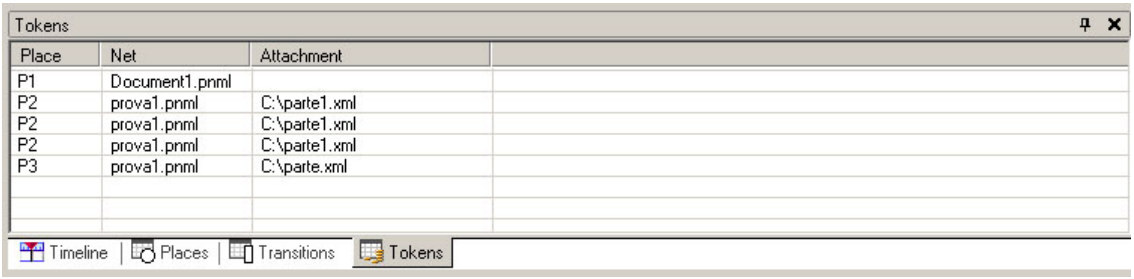
A.2.5 PLACES - TRANSITIONS





Visualizza il riepilogo dei posti/transizioni del progetto corrente o della PN in primo piano (selezionando l'opzione nella parte superiore della finestra), con i parametri principali, utili particolarmente durante il debug.

A.2.6 TOKENS



Visualizza la lista delle marche presenti nelle reti in esecuzione, con l'indicazione del posto in cui è presente la marca, la PN, ed il file allegato al posto.

A.3 IL DISEGNO DELLA RETE

Selezionando dai menu, dalla barra degli strumenti, oppure dal menu contestuale della rete, uno strumento di disegno, è possibile costruire graficamente delle PNs. Vediamo quali sono le azioni possibili.

Click del tasto destro su una parte vuota / su un arco. Viene presentato il menu contestuale per la scelta dello strumento di disegno.

Click del tasto destro su un nodo. Al menu contestuale di cui sopra vengono aggiunte le opzioni di settaggio del nodo come input/output di una sottorete.

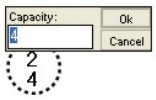
Click del tasto sinistro su una parte vuota. Viene aggiunto alla rete un oggetto in concordanza con lo strumento di disegno selezionato (posto/arco). Se è stato selezionato lo strumento *Arco* ed è stato già cliccato il nodo di partenza, aggiunge un nodo intermedio all'arco.

Click del tasto sinistro su un nodo/arco. Seleziona il nodo/arco (tenendo premuto CTRL viene aggiunto ai nodi/archi selezionati) se lo strumento attivo non è l'*Arco*, nel qual caso viene selezionato il nodo come partenza/arrivo dell'arco[34].

Doppio click su un posto. Edita il parametro su cui si è effettuato il doppio click: nome del posto, marche o capacità.

Doppio click sul nome di una transizione. Edita il nome della transizione.

Doppio click su una transizione. Cambia il verso della transizione (orizzontale/verticale).



Doppio click su un arco. Se il doppio click è effettuato su un punto intermedio di un arco viene cancellato dai punti intermedi. Se viene effettuato sulla parte dell'arco che congiunge due nodi viene aggiunto un punto intermedio.

Rotellina del mouse su un nodo selezionato. Cambia la grandezza del nodo.

Rotellina del mouse su un arco selezionato. Cambia il parametro *tension* (curvature dell'arco).

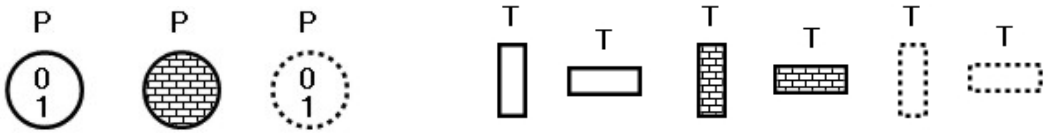
E' possibile, tenendo premuto il tasto sinistro del mouse, trascinare i nodi e i punti intermedi di un arco (quando selezionato).

Inoltre le etichette dei nomi di posti e transizioni e dei peso degli archi possono essere fatti ruotare per non sovrapporli ad altri elementi.

Usando la tastiera è possibile premere CANC per eliminare gli elementi selezionati, oppure ESC per abortire la procedura di creazione di un arco.

[34] Se l'arco creato esiste già, incrementa il suo peso in marche. Se vengono scelti due nodi dello stesso tipo per partenza/arrivo l'operazione viene scartata.

I nodi e gli archi vengono visualizzati diversamente per indicare quando sono selezionati o quando costituiscono un sottonodo:



(Nell'ordine: posto normale, sottoposto, posto selezionato, transizione verticale normale, orizzontale normale, sottotransizioni, transizioni selezionate).

BIBLIOGRAFIA

[ALPHA] *AlphaTech ALPHA/Sim®,* http://www.alphatech.com/secondary/techpro/alphasim/alphasim.html

[ATPN2003] *ICATPN2003*, Eindhoven, Olanda, http://www.tue.nl/atpn2003

[Azz97] **L. Azzaroni,** *Canone infinito. Lineamenti di teoria della musica,* CLUEB, Bologna, 1997.

[CPNTOOLS] *CPN Tools. Computer Tool for Coloured Petri Nets*, http://wiki.daimi.au.dk/cpntools/cpntools.wiki

[CSS2] *Cascading Style Sheets, level 2*, http://www.w3.org/TR/CSS2

[DeMH93] **A. De Matteis, G. Haus,** *Formalizzazione di strutture generative all'interno de La sagra della primavera,* Atti del 10° Colloquio di Informatica Musicale (1993), pp.48-54, Università di Milano, 1993.

[Des00] **J. Desel,** *Place/Transition Nets*, 21st International Conference on Application and Theory of Petri Nets, Introductive Tutorial, Catholic University in Eichstätt, 2000.

[DESCPN] *Design/CPN*, http://www.daimi.au.dk/designCPN

[FINALE] *Coda Music Finale*, http://www.finalemusic.com

[Hau84] **G. Haus,** *Elementi di Informatica Musicale*, Jackson, Milano, 1984.

[HL02] **G. Haus, M. Longari,** *Towards a Symbolic/Time-Based Music Language based on XML*, MAX2002.

[HR93] **G. Haus, A. Rodriguez,** *Formal Music Representation; a Case Study: the Model of Ravel's Bolero by Petri Nets,* in "*Music Processing*", G. Haus

Editor, Computer Music and Digital Audio Series, pp.165-232, A-R Editions, Madison, WI, 1993.

[JKW00] **M. Jüngel, E. Kindler, M. Weber,** *The Petri Net Markup Language,* Petri Net Newsletter*,* 59:24-29, 2000.

[Kàr98] **O. Kàrolyi,** *La musica moderna,* Mondadori, 1998.

[Knu98] **D. E. Knuth,** *The Art of Computer Programming, Volume 2: Seminumerical Algorithms,* Addison Wesley, 1998.

[Lan91] **A. Lanza,** *Il secondo Novecento,* EDT, Torino, 1991.

[LIM] *Laboratorio di Informatica Musicale dell'Università degli Studi di Milano*, http://www.lim.dico.unimi.it

[MAX2002] **AA.VV.,** *Proceedings of First International Conference MAX2002 – Musical Application using XML*, IEEE, 2002.

[MOSES] *The Moses Project*, http://www.tik.ee.ethz.ch/~moses

[MXML] *MusicXML DTD*, http://www.recordare.com/xml.html

[New95] **S. R. Newcomb,** *Standard Music Description Language*, ISO/IEC DIS 10743, ftp://ftp.ornl.gov/pub/sgml/WG8/SMDL/10743.ps, 1995.

[PACE] *IBE GmbH PACE*, http://www.ibepace.com

[PEP] *PEP (Programming Environment based on Petri Nets),* http://parsys.informatik.uni-oldenburg.de/~pep/

[Pet02] **C. Petzold,** *Programming Microsoft Windows with C#,* Microsoft Press, 2002.

[Pet76] **C.A Petri,** *General Net Theory*, Proceedings of the Joint IBM & Newcastle upon Tyne Seminar on Computer Systems Design, 1976.

[Pet81] **J.L. Peterson,** *Petri Net Theory and the Modeling of Systems,* Prentice Hall, New Jersey, 1981.

[PG02] **J. Price, M. Gunderloy,** *Mastering Visual C# .NET,* Sybex, 2002.

[PNK] *Petri Net Kernel,* http://www.informatik.hu-berlin.de/top/pnk/index.html

[PNML] *Petri Net Markup Language*, http://www.informatik.hu-berlin.de/top/pnml

[PNW] *Petri Nets World*, http://www.daimi.au.dk/PetriNets

[Rei00] **W. Reisig,** *An Informal Introduction to Petri Nets,* 21st International Conference on Application and Theory of Petri Nets, Introductive Tutorial, Humboldt University of Berlin, 2000.

[RELAXNG] **J. Clark, M. Murata** (eds.), *RELAX NG specification*, http://www.oasis-open.org/committees/relax-ng.

[RENEW] *Renew: The Reference Net Workshop*, http://www.renew.de

[RR92] **S. Rajasekaran, K. W. Ross,** *Fast Algorithms for Generating Discrete Random Variates with Changing Distributions*, Univ. of Pennsylvania, Philadelphia, 1992.

[Rud93] **W. Rudziński,** *Il ritmo musicale. Teoria e storia,* LIM, Lucca, 1993.

[Sam88] **A. Sametti,** *Un sistema per la sintesi di partiture musicali mediante esecuzione di Reti di Petri,* Tesi di Laurea in Scienze dell'Informazione, A.A. 1987-1988, Università degli Studi, Milano.

[Sam92] **A. Sametti,** *Manuale d'uso di ScoreSynth 3.0,* 1992.

[SLMI] **AA.VV. (G. Haus, Scientific Direction),** *The Intelligent Music Workstation (IMW) CD-ROM,* mixed mode CD-ROM (Macintosh HFS + CD-DA), IEEE Computer Society Press, 1994.

[SMDL] **S.R. Newcomb,** *Standard Music Description Language complies with hypermedia standard*, COMPUTER, vol. X, N.Y. 24:7, pp. 76-79, Luglio 1991.

[Ste02] **C. Stehno**, *Petri Net Markup Language: Implementation and Application*, in J. Desel, M. Weske (eds.), Promise 2002, Lecture Notes in Informatics P-21, pagg. 18-30, Gesellschaft für Informatik, 2002.

[SVG] **J. Ferraiolo, F. Jun, D. Jackson** (eds.), *Scalable Vector Graphics (SVG) 1.1 Specification*, http://www.w3.org/TR/SVG11

[Val78] **R. Valk,** *Self-Modifying Nets, a Natural Extension of Petri Nets,* ICALP 1978, Lecture Notes in Computer Science, N. 62, Springer, Berlin, pp. 464-476, 1978.

[Xen92] **I. Xenakis,** *Formalized Music,* Pendragon Press, 1992.

[XSCHEMA] **M. Sperberg-McQueen, H. Thompson** (eds.), *XML Schema*. http://www.w3.org/XML/Schema

[XSLT] **J. Clark** (ed.), *XSL Transformations (XSLT) Version 1.0,* http://www.w3.org/TR/XSLT/xslt.html