

PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO

SOTTOPROGETTO 7
Sistemi di Supporto al Lavoro Intellettuale
Coordinatore Cesare Maioli

Alberto Sametti [^]

NOTE TECNICHE DEL MODULO
MODELSYNTH

N. M/46

12/93

Rapporto Tecnico Serie MUSIC

[^] LIM - Laboratorio di Informatica Musicale
Dipartimento di Scienze dell'Informazione
Università degli Studi di Milano.

1. Introduzione

1.1 Generalità sull'applicazione ModelSynth

1.2 Requisiti hardware/software per il corretto funzionamento

1.3 Note sulla fase di testing

1.4 Note per l'installazione

2. Strutture dati

2.1 Strutture dati interne

2.2 Strutture dati esterne

3. Struttura dell'applicazione

4. Algoritmi

1. Introduzione

Per una comprensione delle funzionalità di ModelSynth è opportuna comunque la presa visione dei Rapporti Tecnici relativi ai moduli ScoreSegmenter e ScoreSynth.

1.1 Generalità sull'applicazione ModelSynth

In questo paragrafo analizzeremo quella che è l'architettura software più generale, cioè il contesto in cui si inserisce il modulo ModelSynth. Come si vede in figura 1, direttamente correlati con esso esistono altri due moduli facenti parte della SLMI. Essi sono lo ScoreSegmenter e lo ScoreSynth.

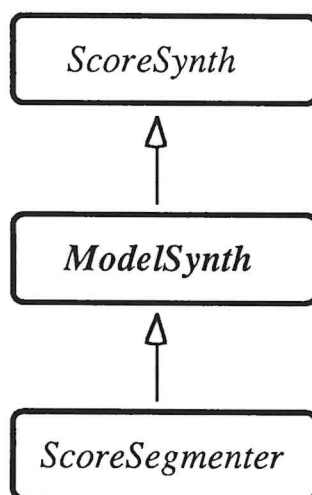


Fig. 1: architettura moduli s/w.

Illustriamo brevemente le peculiarità dei tre moduli. Lo ScoreSegmenter si pone l'obiettivo della segmentazione di brani musicali quale primo passo per una futura loro strumentazione automatica proposta dall'elaboratore. Si tratta cioè della ricerca dei vari oggetti musicali con cui è composto il brano, dove per oggetti intendiamo quei fraseggi o pensieri musicali che l'autore ha espresso e che ha ripreso e trasformato, secondo i vari canoni musicali, in relazione al periodo storico e alla forma compositiva. E' il primo tentativo al fine di realizzare uno strumento informatico che renda possibile l'accostarsi ai testi musicali non come pura sequenza di note, bensì vedendo i testi stessi come un edificio espressivo fatto da alcuni elementi di base composti ed armonizzati fra loro in vario modo. Il modulo ScoreSegmenter è quindi essenzialmente uno strumento di analisi.

Lo ScoreSynth al contrario è uno strumento di sintesi. Esso rende disponibile un ambiente integrato per la creazione, debugging ed esecuzione di modelli di

partiture musicali basati sul formalismo delle reti di Petri. L'obiettivo primario del modulo ScoreSynth è quello di mettere a disposizione del 'musicista' uno strumento molto potente ed efficace che permette di considerare l'attività compositiva da un punto di vista architeturale: il 'musicista' manipola sequenze, funzioni di trasformazione e strutture.

Infine il ModelSynth permette di tradurre, in termini di modelli di reti di Petri eseguibili da ScoreSynth, i risultati dell'analisi operata dal modulo ScoreSegmenter.

Considerando i tre moduli da un punto di vista funzionale vediamo ora quali trasformazioni definiscono e cosa costituisce il dominio e il codominio per ciascuno di essi.

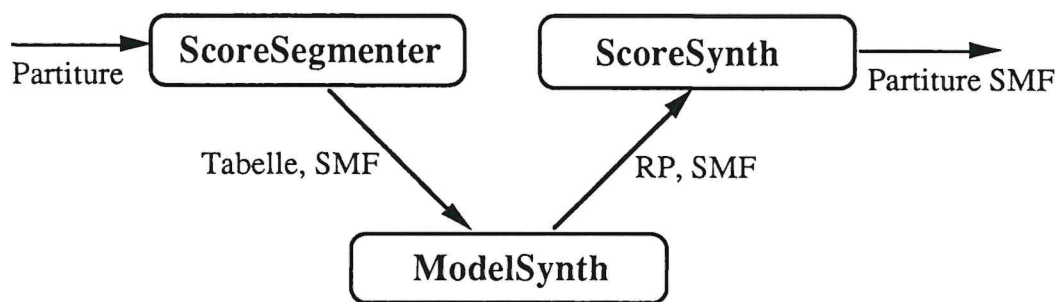


Fig. 2: flussi di Input/Output.

ScoreSegmenter

La funzionalità principale correlata con le caratteristiche di ModelSynth è quella della segmentazione automatica di un brano.

ScoreSegmenter è in grado di leggere una partitura espressa in notazione musicale tradizionale e generare un file di supporto (detto 'file di lavoro'), corrispondente al file in ingresso, tramite cui operare la propria analisi, i vari temi riconosciuti in formato SMF e, più importante, una tabella testuale contenente i risultati delle ricerche.

Quest'ultima tabella per ogni tema o frammento trovato riporta le seguenti informazioni: la voce, la nota iniziale e la nota finale all'interno della partitura, l'operatore eventualmente applicato e la prima nota riconosciuta del tema. D'ora in poi faremo riferimento ad ogni riga della tabella con il termine 'atomo'.

Affinchè le tabelle dello ScoreSegmenter siano significative fissare i parametri di impostazione della segmentazione in modo da effettuare ricerche esclusive su frammenti e temi. In modo da evitare che una sequenza di note, in una certa posizione nella partitura, appartengano a più temi o frammenti.

ModelSynth

In sostanza ciò che ModelSynth compie è una operazione inversa rispetto a quella dello ScoreSegmenter. Infatti cerca di ricostituire la struttura del brano musicale analizzato e atomizzato. Chiaramente questa operazione per non essere un passaggio ridondante deve conferire alla struttura finale del brano alcune caratteristiche significative. Queste caratteristiche si possono riassumere in una sola: rappresentare con modelli flessibili il contenuto informativo del brano, dove per 'informativo' intendiamo le relazione tra gli atomi costituenti e le loro trasformazioni nel corso dello svolgimento. Ciò che viene realizzato è quindi l'analisi di una analisi, allo scopo di, da una parte, fornire una rappresentazione alternativa, dall'altra estrarre e codificare l'informazione.

ScoreSynth

Come detto ScoreSynth permette la creazione, il debugging e la esecuzione di modelli di reti di Petri orientati alla sintesi di partiture musicali MIDI (Musical Instruments Digital Interface). Per una dettagliata descrizione delle funzionalità si rimanda ai documenti relativi facenti parte della medesima collana.

1.2 Requisiti hardware/software per il corretto funzionamento

Per il corretto funzionamento dell'applicazione è richiesto qualunque modello della linea Apple Macintosh in grado di supportare una versione di sistema operativo uguale o successiva alla versione 7.0.

1.3 Note sulla fase di testing

La fase di test è stata effettuata con Sistema Operativo versione 7.1 su modelli Apple Macintosh vi / vx / Quadra 700 / Quadra 950 / Centris 650 non riscontrando incompatibilità di alcun tipo.

1.4 Note per l'installazione

Il modulo ModelSynth è costituito da un file di applicazione con nome appunto "ModelSynth" e una cartella con nome "MS/NetLib". Per una corretta installazione ed esecuzione questi due elementi devono risiedere nella medesima cartella (directory) del Finder in uno qualsiasi dei volumi di file system presenti. Non necessariamente nel volume di avvio.

La cartella “MS/NetLib” relativa alla versione attuale di ModelSyth, cioè la versione v. 1.0.0, deve contenere i seguenti file:

- *ObjAlgSel(1-5)*
- *Object Alg*
- *Place Loop*
- *SubPlace Loop*
- *TopNet*

È opportuno non togliere e non aggiungere file estranei da e in questa cartella. Vedere il paragrafo relativo ai messaggi di errore per segnalazioni di installazione scorretta per quanto riguarda questa cartella.

2. Strutture dati

L'intero progetto è stato realizzato utilizzando il linguaggio Pascal fornito disponibile nell'ambiente di sviluppo MPW v. 3.2 di Apple.

2.1 Strutture dati interne

Nel seguito forniamo l'elenco delle variabili globali più significative. Ne diamo la definizione, così come nel codice sorgente, e una breve descrizione testuale. Tutte le variabili globali, le costanti e i tipi sono contenute nel file "MS_Globals.p". I tipi utilizzati derivati dai moduli ScoreSegmenter e ScoreSynth non verranno dettagliati. Viene comunque segnalata la loro specificità con un rimando ai documenti tecnici relativi.

Variabili generiche

gMainDialog: DialogPtr - Dialogo principale;

gParseProgress: DialogPtr - Dialogo di stato dell'operazione di 'parsing' delle tabelle;

gBuildModelProgress: DialogPtr - Dialogo di stato dell'operazione di generazione del modello di RPM;

gBuildNetLibProgress: DialogPtr - Dialogo di stato dell'operazione di caricamento delle RPM della libreria "MS/NetLib";

gGluesCreatorProgress: DialogPtr - Dialogo di stato dell'operazione di creazione dei MIDI 'glue';

gWriteSortedAtoms: Boolean - Variabile di stato che specifica se salvare su disco la lista ordinata degli atomi riconosciuti durante la fase di 'parsing' delle tabelle;

Variabili descrittive del modello

Sono stati aggiunti i seguenti tipi:

Cowtry = (english, italian) - Lingua utilizzata nelle tabelle testuali di ScoreSegmenter;

ParseTableStatus = (parseAll, parseTheme, parseAtomList, parseAbstract, parsePause, parseEnd, parseError, parseTonality) - Gli stati possibili durante l'operazione di parsing delle tabelle;

AtomType = (unknown, fragment, theme) - Tipi di atomi;

BasicOperator = (realTransp, tonalTransp, realInv, tonalInv, retrogRealTransp, retrogTonalTransp, retrogRealInv, retrogTonalInv) - Operatori ScoreSegmenter;

Descrittore di un atomo:

```
AtomSpec = record
    atomFlags: integer;
    themeCode: integer;
    typo: AtomType;
    refNum: integer;
    voice: integer;
    startMeas: integer;
    startNote: integer;
    startNoteSpec: integer;
    endMeas: integer;
    endNote: integer;
    endNoteSpec: integer;
    operator: BasicOperator;
    operatorData: integer;
    ref: integer;
end;
```

AtomSpecArray = array[1..MAXINT] of AtomSpec;

AtomSpecArrayPtr = ^AtomSpecArray;

AtomSpecArrayHandle = ^AtomSpecArrayPtr - Lista dinamica di atomi;

Descrittore di un tema:

```
ThemeSpec = record
    themeName: Str31;
    instanceNum: integer;
    themeLen: integer;
end;
```

ThemeSpecTable = array[1..MAXINT] of ThemeSpec;

TS_ptr = ^ThemeSpecTable;

TS_handle = ^TS_ptr- Lista dinamica di atomi;

Descrittore del modello corrente:

```
CurModelSpec = record
    modelName: Str63;
    vRefNum: integer;
    tableNum: integer;
    wfDataHndl: Handle;
    voiceNum: integer;
    parseStatus: ParseTableStatus;
    language: Cowntry;
    themeNum: integer;
    atomNum: longint;
    atomHandle: AtomSpecArrayHandle;
    validAtoms: integer;
    themeHandle: TS_handle;
    division: integer;
    BPM: integer;
    tonality: Str31;
end;
```

gCurModel: *CurModelSpec* - Modello corrente;

gTableLine: *CharLine* - Buffer per le letture a linee delle tabelle;

gNetHandle: *NS_handle* - Lista delle RPM del modello; le prime sono quelle di libreria;

gNetNum: integer - Numero di RPM nel modello;

gCurNet: integer - Indice della RPM corrente;

gSavedNetNum: integer Numero di RPM salvate su disco;

Variabili descrittive della libreria di reti

gNetLibVRefNum: integer - Riferimento della cartella con le reti di libreria in formato ScoreSynth;

gLibNetNum: integer - Numero di RPM in libreria;

Variabili gestione 'glue'

gGlueHandle: *GS_handle* - Lista con le specifiche dei 'glue'; anche se solo pause;

gGlueNum: integer; - Numero di 'glue' nella lista;

Variabili parsing file di lavoro

CurWF_Lenght: Longint - Lunghezza del WF corrente;

CurWF_MeasuresNum: Integer - Numero di misure del WF corrente;

CurWF_VoicesNum: Integer - Numero di voci del WF corrente;

CurWF_Measure: Integer - Battuta corrente;

CurWF_Note: Integer - Nota corrente;

CurWF_MeasureLenght: Integer - Lunghezza in byte della battuta corrente;

CurWF_LastMeasure: Integer - Ultima battuta del brano;

CurWF_LastNote: Integer - Ultima nota del brano;

CurWF_Voice: Integer - Numero di Voce corrente;

CurWF_Pos: Longint - Posizione corrente di lettura del WF;

CurWF_Ptr: ptrSB - 'Punta' alla posizione corrente del WF;

EndOfCurVox: Boolean - Segnalatore della fine di voce;

GlueIndex: Integer - Indice del 'glue' corrente;

2.2 Strutture dati esterne

La descrizione del formato delle partiture, dei file di lavoro, delle tabelle testuali degli atomi si trovano nel documento "Note Tecniche" del modulo ScoreSegmenter. La descrizione del formato dei file delle RPM si trova nel documento "Note Tecniche" del modulo ScoreSynth. Per il formato di tutti gli standard MIDI file in gioco si faccia riferimento alla relativa documentazione ufficiale; si tenga comunque presente che essi sono del tipo denominato "Formato 0".

L'unico file con un formato 'proprietario' è il file che riporta l'elenco di tutti gli atomi riconosciuti ordinati secondo voce, misura e nota di inizio. Si veda come esempio la figura seguente:

IND	TYP	OK	REC	THE	VC	MEA	NTS	MEA	NTE
1	T	Yes	1	1	1	1	2	2	1
2	T	No	1	2	1	1	2	1	9
3	T	Yes	2	2	1	2	2	2	9
4	T	Yes	1	3	1	2	10	3	1

5	T	Yes	1	4	1	11	2	11	5
6	T	Yes	2	4	1	11	6	12	1
7	T	Yes	3	4	1	12	2	12	5
8	T	Yes	3	2	1	20	2	20	9
9	T	Yes	4	2	2	1	3	2	1
10	T	Yes	5	2	2	2	5	3	1
11	T	Yes	4	4	2	3	2	3	5
12	T	Yes	5	4	2	3	6	4	1
13	T	Yes	6	4	2	4	2	4	5
14	T	Yes	6	2	2	5	2	5	9
15	T	Yes	7	4	2	5	10	6	1
16	T	No	8	4	2	5	12	6	3
17	T	No	7	2	2	7	2	7	9
18	T	Yes	2	1	2	7	2	8	1
19	T	Yes	8	2	2	8	2	8	9
20	T	Yes	9	4	2	19	2	19	5
21	T	Yes	10	4	2	19	6	20	1

Fig. 8: esempio di file con gli atomi ordinati.

Esso è un comune file ASCII con separatore di campo il carattere di tabulazione.

Le colonne indicano rispettivamente: un indice progressivo crescente; il tipo di atomo, tema(T) o frammento(F); se l'atomo è valido(Yes) o no(No); il numero di riferimento dell'atomo, come generato ScoreSegmenter; l'indice del tema a cui si riferisce; la voce; la misura di inizio; la nota di inizio; la misura di terminazione; la nota di terminazione.

3. Descrizione dei moduli software

Tutti il codice sorgente è stato suddiviso nei seguenti moduli:

MS_Globals.p

Contiene le definizioni di costanti, tipi e variabili globali.

MS_MainProc.p

Funzioni dedicate alla gestione 'main event loop'.

Principali funzioni:

procedure Initialize - Inizializzazione dei manager del Toolbox utilizzati e delle variabili di gestione del loop degli eventi;

procedure EventLoop - Loop di riconoscimento degli eventi;

procedure DoEvent (event: EventRecord) - Processing di un evento prelevato dalla coda;

procedure DoDialogEvent (theEvent: EventRecord) - Processing di un evento relativo al dialogo principale;

function ReadNetLib: integer - Crea la libreria interna di RPM sulla base dei file ScoreSynth contenuti nella cartella "MS/NetLib";

MS_NetProc.p

Funzioni dedicate alla gestione delle strutture che implementano la rappresentazione Reti di Petri.

Principali funzioni:

procedure ReleaseNet (netInd: integer) - Rilascia la memoria allocata da una RPM;

procedure NetDefault (var theNet: R_Record) - Inizializza i campi del record di descrizione di una rete;

procedure WriteNet (netind: integer; toFromf: integer) - Salva una rete su disco;

procedure ReadNet (netind: integer; toFromf: integer) - Carica una rete da disco;

function DoInserPlace (netInd: integer; thePlace: P_record): integer - Aggiunge un posto alla rete specificata;

procedure PlaceDefault (var thePlace: P_Record; theLocation: Point; netInd: integer) - Inizializza i campi del record di descrizione di un posto;

function DoInsertTrans (netInd: integer; theTrans: T_Record): integer - Aggiunge una transizione alla rete specificata;

procedure TransDefault (var theTrans: T_Record; pos: integer; theLocation: Point; netInd: integer) - Inizializza i campi del record di descrizione di una transizione;

function DoInsertTPArc (netInd: integer; startObj: integer; endObj: integer; theArc: A_Record): integer - Aggiunge un arco *transizione->posto* alla rete specificata;

function DoInsertPTArc (netInd: integer; startObj: integer; endObj: integer; theArc: A_Record): integer - Aggiunge un arco *posto->transizione* alla rete specificata;

procedure ArcDefault (var theArc: A_Record; startPt: Point; endPt: Point) - Inizializza i campi del record di descrizione di un arco;

function FindPlace (netInd: integer; placeName: Str255; var pInd: integer): Boolean - Cerca un posto per nome all'interno della rete specificata;

function FindTrans (netInd: integer; transName: Str255; var tInd: integer): Boolean - Cerca una transizione per nome all'interno della rete specificata;

MS_ParseProc.p

Funzioni dedicate all'operazione di 'parsing' delle tabelle testuali di ScoreSegmenter. Principali funzioni:

function DoSelectNewSEGM: boolean - Seleziona il modello corrente;

procedure DoClearCurModel - Deselezione il modello corrente;

function DoParseSEGMTTable: integer - Opera il parsing su tutte le tabelle relative al file di lavoro selezionato;

function GetVoiceNum: integer - Calcola il numero di voci nella partitura;

procedure SortAtoms - Sort degli atomi per voce, misura e nota di inizio;

procedure VerifyAtoms - Verifica della consistenza degli atomi riconosciuti;

procedure WriteSort - Scrittura su disco di un file con la lista ordinata degli atomi in formato testo;

function InitParsing: integer - Inizializzazione delle variabili di stato dell'operazione di parsing;

procedure CloseParsing - Teminazione del parsing;

procedure SetAtomSpec (var theAtom: AtomSpec; theAtomStr: Str255) - Riempimento dei campi del record di descrizione di un atomo sulla base di una linea ricavata dalle tabelle testuali;

function ParseTableLine (theLine: CharLine): integer - Parsing di una linea di testo;

function ParseTableFile (fileRef: integer; totalLength: longint; var curPos: longint): integer - Parsing di una intera tabella;

function AddThemeSpec: integer - Aggiunge un tema alla lista;

function AddAtomSpec (theAtomStr: Str255): integer - Aggiunge un atomo alla lista;

MS_NetModelBuilder.p

Funzioni dedicate alla generazione del modello di Reti Di Petri Musicali. Principali funzioni:

procedure DoBuildNetModel - Genera il modello corrente;

procedure DoReleaseAllNet (lastLibNet: integer) - Rilascia la memoria di tutte le RPM del modello escluse le reti di libreria;

function InitBuilding: Boolean - Inizializzazione delle variabili;

function DoVoiceNet: Boolean - Creazione della RPM relativa alle voci;

function DoThemeNet: Boolean - Creazione della RPM relativa ai temi;

function DoTopNet: Boolean - Creazione della RPM 'top';

function BuldVoiceNet (parentName: Str255; parentNet: integer; parentObj: integer; voiceInd: integer): Boolean - Creazione della RPM relativa a una voce;

procedure BuildAtomNet (parentName: Str255; parentNet: integer; parentObj: integer; atomInd: integer) - Creazione della RPM relativa ad un atomo;

procedure BuildSubPlaceLoopNet (parentName: Str255; parentNet: integer; parentObj: integer; fromAtom, toAtom: integer; voiceInd: integer) - Creazione della RPM relativa ad un loop di atomi;

function NetLibClone (sourceNet: integer; destNet: integer): integer - Creazione della RPM relativa alle voci - Duplicazione di una RPM del modello;

function DoNewNet (netInd: integer; objInd: integer): integer - Aggiunge una RPM al modello;

function UseLibNet (netName: Str255): integer - Verifica la presenza di una RPM in libreria;

function DoWriteAllNet: Boolean - Salva tutte le RPM utilizzate per il modello corrente;

procedure AtomOperatorStr (theAtom: AtomSpec; var theAtomStr: Str255) - Converte un operatore ScoreSegmenter in uno ScoreSynth;

procedure RestValueToStr (theDivision: integer; theBPM: integer; theRestValue: integer; var theRestStr: Str255) - Converte una pausa della partitura in una pausa ScoreSynth;

MS_GlueProc.p

Funzioni dedicate alla creazione dei MIDI 'glue'.

Principali funzioni:

function LoadCurWF: integer - Carica il file di lavoro in memoria;

procedure SetInitWF - Inizializzazione delle variabili di posizione sul file di lavoro;

function NewGetNota (VNum: Integer; var Notes: TpPattern; var NotesNum: Byte): Boolean - Legge la prossima nota nel file di lavoro;

function VaiB_N (VNum: Integer; BNum: Integer; NNum: Integer): Boolean - Si posiziona sul file di lavoro alla battuta e note specificata;

procedure GluesCreator - Ricerca e scrittura su file di tutti i MIDI 'glue';

MS_WSMF.p

Funzioni di utilità dedicate alla scrittura degli Standard MIDI File.

Principali funzioni:

procedure CreateGlueFile (nomeFile: STR255) - Salva su disco il MIDI 'glue' corrente;

MS_Uutilities.p

Funzioni di utilità.

4. Algoritmi

Le fasi di selezione di un file di lavoro, parsing delle tabelle relative, generazione dei MIDI 'glue' e generazione del modello i RPM si possono così meglio specificare.

- 1 - Selezione di un WF;
- 2 - Parsing di tutte le tabelle associate: creazione della lista di tutti gli atomi riconosciuti;
- 3 - Ordinamento degli atomi secondo le seguenti chiavi: voce, misura di inizio, nota di inizio;
- 4 - Eliminazione degli atomi di tipo diverso da "T";
- 5 - Verifica della consistenza della lista degli atomi: se esistono due atomi sovrapposti temporalmente viene mantenuto solo quello con numero maggiore di note; se uguali, il primo incontrato nella lista.
- 6 - Identificazione del numero di temi;
- 7 - Identificazione del numero di voci;
- 8 - Creazione di una lista delle occorrenze dei temi;
- 9 - Creazione di una lista delle occorrenze dei 'glue';
- 10 - Creazione su disco dei MIDI file corrispondenti a 'glue' non vuoti: cioè escluse le pause;
- 11 - Generazione del modello di RPM; con una gerarchia di chiamate a funzione di questa natura:
 - DoTopNet;
 - DoThemeNet;
 - DoVoiceNet;
 - BuildVoiceNet;
 - BuildAtomNet;
 - BuildSubPlaceNet;
- 12 - Scrittura su disco di tutte le reti nel modello;

