UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze e Tecnologie
Corso di Laurea Triennale in Informatica Musicale

PROTOTIPO JAVA PER LA CONVERSIONE DA MIDI A IEEE1599

Relatore: Prof. Luca Andrea LUDOVICO

Correlatore: Adriano BARATÈ

Tesi di Laurea di:
Fabio LEARDINI
Matricola 760071

Anno Accademico 2012 - 2013

Indice

Introduzione

Questo elaborato ha l'obiettivo di progettare e implementare un prototipo software per la conversione automatica di file dal formato MIDI a IEEE1599.

Nella prima parte sono trattati il linguaggio Java, il metalinguaggio XML e il suo utilizzo nello standard IEEE 1599 e il protocollo MIDI, le tre tecnologie utilizzate per sviluppare il prototipo.

Java è un linguaggio di programmazione multipiattaforma, ossia indipendente dalla piattaforma di esecuzione. È un linguaggio orientato agli oggetti, sviluppato nel 1995 da Sun Microsystems e acquistato nel 2010 da Oracle. Nell'elaborato si utilizzano ampiamente classi e strutture dati dinamiche.

Lo standard MIDI è stato sviluppato per consentire l'iterazione tra strumenti musicali elettronici. Non distingue i semitoni cromatici da quelli diatonici ma usa lo stesso valore a parità di frequenza, inoltre non ha una gestione della durata delle note in valori ritmici, ma codifica la distanza tra eventi in tick, cioè in impulsi. Nell'elaborato viene descritto il protocollo facendo riferimento ai messaggi e agli Standard MIDI File.

Lo standard IEEE1599 è stato sviluppato fornire una rappresentazione completa di un brano musicale. Esso supporta file audio, video e grafici e utilizza XML, un metalinguaggio che consente di creare tag per descrivere elementi. Nel prototipo viene utilizzato per descrivere i layer, i loro attributi e i vari parametri del file convertito nello standard IEEE1599. Nell'elaborato si spiega come XML gestisce gli elementi del protocollo IEEE1599 e quali tag usa per farlo.

Nella seconda parte viene descritto il programma sviluppato facendo riferimento alle problematiche incontrate, tra cui la gestione della durata delle note e i problemi di sincronizzazione.

Lo scopo del prototipo è di convertire un file dallo standard MIDI al formato IEEE1599, utilizzando le librerie per la gestione dei file MIDI implementate in Java e risolvendo i problemi riguardanti la tonalità, la durata delle note e le limitazioni del protocollo. Il programma, inoltre, gestisce la mappatura tra il file XML e quello MIDI. Il linguaggio di programmazione scelto per sviluppare il progetto è stato Java.

Oltre alla sua caratteristica di linguaggio multipiattaforma, esso è stato scelto per la presenza della libreria javax.sound.midi che permette di usufruire di un insieme di classi e di funzioni che consentono una gestione del protocollo più facile, agevolando la gestione degli eventi, del-

le tracce e delle sequenze. La libreria fornisce, inoltre, accesso ai dati relativi al formato e alle caratteristiche del MIDI.

Un'altra libreria rivelatasi fondamentale durante lo sviluppo del prototipo è org.jdom2, che consente di gestire agevolmente i file XML.

Tecnologie utilizzate

MIDI

Il MIDI è un protocollo standard che è stato sviluppato negli anni ottanta. Il suo scopo è di far interagire tra loro gli strumenti musicali. MIDI è acronimo di Musical Instrument Digital Interface, cioè interfaccia digitale per strumenti musicali.

Il protocollo MIDI serve a definire le regole per lo scambio di dati tra periferiche. Si determinano quanti e quali byte servono per effettuare istruzioni riconoscibili dalle apparecchiature collegate, quale deve essere la velocità, che circuito deve avere l'interfaccia e tutto ciò che serve per effettuare una comunicazione corretta. Inoltre consente di controllare tutta una serie di apparecchi contemporaneamente facendoli lavorare in modo sincronizzato.

Il protocollo non è stato modificato dall'anno della sua creazione e questo fa sì che dispositivi molto vecchi siano in grado di comunicare con i dispositivi moderni e continuino dunque ad essere utilizzabili senza problemi.

La comunicazione avviene attraverso un'interfaccia seriale in modalità binaria e questo consente di realizzare interfacce MIDI molto economiche. La comunicazione seriale non è un grosso problema in quanto la quantità di dati spediti è relativamente ridotta. La velocità di trasferimento dei dati non è molto alta ma è sufficiente allo scopo. Essa è di 31250 bit/sec ossia 3906.25 bytes/s ossia 3.8 Kb/s.

Le prese MIDI, che vengono anche definite porte, sono tre:

- MIDI In che riceve i dati MIDI in entrata;
- MIDI Out che invia i dati MIDI;
- MIDI Thru che invia una copia esatta dei dati entrati dalla porta MIDI In.

I connettori sono DIN a cinque poli dei quali ne sono utilizzati tre: uno per la schermatura (2) e due per il trasferimento dei dati (4 e 5). Gli altri due sono inutilizzati.

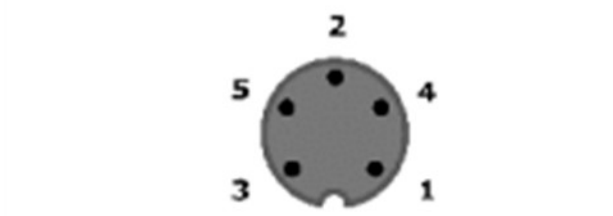

FIGURA 1.1 Immagine di un connettore DIN.

I cavi non devono avere una lunghezza superiore ai 15 metri.

La comunicazione è di tipo asincrono ossia non vengono impiegati segnali di temporizzazione ma sono utilizzati bit di start e bit di stop.

Messaggi: Come sono composti

I messaggi sono composti da due tipi di byte:

- Status byte che indica il tipo di messaggio;
- Data byte che trasporta le informazioni numeriche relative al messaggio.

Un messaggio è composto da tre byte: uno di status e due di data.

Il primo bit del byte di status e posto a 1. I tre bit seguenti servono a definire il tipo di messaggio mentre gli ultimi quattro servono a definire il canale.

I data byte contengono le informazioni riguardanti alle note e suonate e alla velocità della nota cioè l'intensità con cui viene premuto il tasto. Il primo bit è posto a 0 per cui avremo la possibilità di suonare 128 note.

Un esempio di messaggio è visibile nella figura 1.2 dove *t* sta per il tipo, *c* sta per il canale, *n* sta per la nota e *v* sta per velocity, che rappresenta l'intensità.

STATUS Byte	DATA Byte	DATA Byte
1ttt cccc	0nnn nnnn	0vvv vvvv

FIGURA 1.2 Esempio di messaggio

Tipi di messaggi

I messaggi sono di due tipi: Channel messages, divisi a loro volta in Channel voice messages e Channel mode messages; System messages, divisi a loro volta in System common, System real time e System Exclusive

Channel voice messages

Nella tabella 1.1 si possono vedere i channel voice messages con il relativo status byte.

I messaggi note on e note off servono per accendere o spegnere la nota cioè per farla suonare o farla smettere. Contengono le seguenti informazioni:

- Informazioni di canale, da 1 a 16
- Numero di nota espressa in valori da O a 127.

- Velocità di pressione del tasto, cioè la dinamica di esecuzione, sempre con un'escursione numerica da 0 a 127.

Il Polyfonic after touch indica il valore di pressione che viene applicato sul tasto dopo che è stato premuto. Questo messaggio fa aumentare gli effetti applicati alla nota come per esempio il vibrato e il variare del volume. Viene raramente implementato a causa del suo costo elevato, infatti prevede un sensore sotto ogni tasto.

Il Channel pressure aftertouch, come il precedente, indica il valore di pressione dopo che un tasto è stato premuto ma questa volta viene implementato un solo sensore per cui il movimento di un tasto varierà anche quello degli altri tasti.

Il Program change indica il variare del suono di un canale.

Il pitch bender permette di eseguire l'effetto bending della chitarra. Esso provoca come effetto sonoro la modifica dell'intonazione della nota suonata in maniera tanto più rilevante, quanto più ampia è l'azione esercitata sulla ruota o leva adibita al controllo.

DEC	**MESSAGGIO**
128 - 143	NOTE OFF canale midi 1 – 16
144 - 159	NOTE ON canale midi 1-16
160 – 175	POLYFONIC AFTER TOUCH (key pressure) canale midi 1 – 16
176 - 191	CONTROL CHANGE canale midi 1 – 16
192 - 207	PROGRAM CHANGE canale midi 1 – 16
208 - 223	AFTER TOUCH (channel pressure) canale midi 1 – 16
224 - 239	PITCH BENDER canale midi 1 – 16

TABELLA 1.1 Tipi di channel voice message

Channel mode messages

Riguardo al control change dei Channel Voice Messages possiamo avere 127 diversi codici (sono i 7 bit del primo data byte). I primi 120 sono relativi a parametri tipici come volume, pan ecc. Gli ultimi 7 sono messaggi che agiscono sulla modalità di funzionamento di un canale e sono:

- 121 - Reset all controllers che imposta tutti i parametri del dispositivi vengono posti pari al loro valore iniziale cioè quello definito all'accensione del dispositivo;
- 122 - Local control on/off;
- 123 - All notes off che manda un messaggio di tipo note off a tutte le note;
- 124-125-126-127: Gli ultimi 4 messaggi definiscono il cosiddetto MIDI mode ossia il modo in cui la tastiera-sintetizzatore interpreta i comandi MIDI.
- 124-125 Omni mode off-on che attivano o disattivano la modalità Omni. Questa, se attivata, fa in modo che nei messaggi MIDI venga ignorata l'informazione relativa al canale e dunque ogni messaggio viene applicato a tutti i canali. Viceversa, quando la modalità è disattivata, ogni messaggio viene applicato al canale indicato nello status byte;
- 126-127 Mono On - Poly On: la modalità Mono on impone sul canale la possibilità di eseguire una sola nota per volta. La modalità Poly on consente di eseguire più note contemporaneamente sul singolo canale.

System messages

Questi messaggi sono spediti contemporaneamente su tutti i canali MIDI.

Un esempio di messaggio di sistema è visibile nella figura 1.3. *t* indica il tipo di messaggio di sistema (4 bit) e *d* indica i dati

1 1 1 1 t t t t	0 d d d d d d d	0 d d d d d d d

FIGURA 1.3 Esempio di messaggi di sistema

Vi sono tre tipi di messaggio di sistema:

- System common;
- System real time;
- System Exclusive.

I system common comprendono:

- MTC: Midi Time Code (temporizzazione);
- Song Position Pointer : Identificativo temporale di un punto della canzone;
- Song select: selezione di una canzone nella memoria del dispositivo;
- End of exclusive: Fine della trasmissione di dati di sistema.

I system real time sono messaggi brevi (1 byte) che vengono spediti mischiati ai normali messaggi. Servono per mantenere la sincronizzazione tra i sistemi collegati via MIDI. Questi messaggi vengono spediti continuamente ad un ritmo di 24 ogni quarto di nota quindi il numero di messaggi spediti nell'unità di tempo aumenta col bpm del brano. Se la dimensione di dati MIDI da spedire è molto grande, può convenire spedire i messaggi di temporizzazione su un'uscita MIDI separata.

I messaggi System Exclusive (Sys-ex) trasportano dati specifici del dispositivo in uso. Ogni dispositivo MIDI professionale è dotato di questa funzionalità che permette di scaricare via MIDI tutta la memoria. In questo modo tutti i settaggi che sono stati fatti sul dispositivo per un certo lavoro vengono spediti per esempio su un computer dove vengono memorizzati. Nella necessità di impostare il dispositivo su configurazioni completamente diverse è molto utile e immediato utilizzare i settaggi salvati su computer sotto forma di file ritrasferendoli nel dispositivo sempre via MIDI.

Metaeventi

I metaeventi rappresentano informazioni aggiuntive ad un midifile. Non sono eventi midi, ma messaggi che contengono informazioni di vario tipo come copyright, nome delle tracce, la misura del tempo.

I metaeventi sono informazioni non imprescindibili e non fondamentali del file MIDI, che, quando sono presenti, sono contenute nell'intestazione delle traccie. Essi si definiscono dal primo byte di valore sempre uguale a FF, e sono costituiti da altri byte. La lunghezza del messaggio cambia in base al tipo.

I metaeventi sono composti anche da un byte che identifica il tipo, uno che indica la lunghezza , cioé il numero di byte che segue, e una serie di byte che contengono i dati

FF	Tipo di metaevento	Lunghezza	Byte 1	...	Byte N

FIGURA 1.3 Esempio di meta evento

Tipo di Metaevento	Commento
01	Evento di testo per descrivere il nome di una traccia o il nome della voce usata, può essere utilizzato anche per inserire il testo.
02	Evento di copyright, il testo deve contenere la © seguita dall'anno di creazione e dal nome dell'autore. Dovrebbe essere inserito come primo evento sulla traccia 1.
03	Evento di testo per indicare il nome della traccia. Se è inserito in un midifile in formato 0 o nella prima traccia di un midifile in formato 1, indica il nome del brano.
05	Evento di testo per inserire le liriche del brano.
2F	Evento che indica la fine di un blocco di traccia (Track-chunk).
51	Evento di velocità del brano, indica quanti microsecondi ci debbono essere in un quarto di nota.
58	Evento per descrivere la divisione del brano.
59	Evento che indica la tonalità.

TABELLA 1.2 Tipi di meta evento.

Struttura dei file

I file sono divisi in due blocchi:

- header chunk, cioè l'intestazione del file midi che contiene le informazioni riguardanti ad esso;
- track chunk, cioè l'intestazione della traccia che contiene le informazioni che la riguardano.

Ogni traccia usa un canale che varia da 1 a 16. Ogni canale identifica una funzionalità di un dispositivo MIDI, per esempio su una tastiera-sintetizzatore può identificare un particolare suono.

Header Chunk

L'Header-chunk (blocco d'intestazione) è il primo elemento inserito nel midifile, in quanto contiene le informazioni del formato, del numero di tracce e della temporizzazione. Inizia con quattro caratteri ASCII: MThd, che identificano il blocco.

Dati esadecimale	Commento
4D 54 68 64	ASCII=MThd
00 00 00 06	Lunghezza del blocco, in questo esempio 6 byte
00 00	Formato del midifile (0,1,2)
00 01	Numero delle tracce (da 1 a 16)
00 60	Tick per indicare un quarto di nota

TABELLA 1.3 Un esempio di Header Chunk.

Track Chunk

Si inserisce un track chunk per ogni traccia presente nel file.

All'interno del track chunk sono contenute le informazioni di messaggi, di metaeventi o di e-xclusive messages.

Dati esadecimali	Commento
4D 54 72 6B	ASCII=MTrk
00 00 00 00	Lunghezza del blocco, in questo esempio 0 byte

TABELLA 1.4 Un esempio di Track Chunk.

I messaggi contengono i campi indicanti l'evento e un campo per il delta time che è l'intervallo di tempo espresso in ticks che divide due eventi. Se il messaggio in questione dice di suonare una nota per alcuni secondi, le informazioni relative al tempo vengono codificate nel delta-time, mentre le informazioni relative alla nota da suonare vengono memorizzate nei byte successivi.

valore di delta time	lunghezza del messaggio	messaggio

FIGURA 1.4 Composizione di un messaggio MIDI.

Il delta time viene scritto su gruppi di byte così suddivisi:

Il bit più significativo di ogni byte è il primo partendo da sinistra e serve ad indicare se c'è un altro byte successivo. È a 1 se c'è o a 0 in caso contrario.

L’informazione riguardante il delta-time sta nei restanti 7 bit di ogni gruppo di byte.

Se due eventi devono essere suonati contemporaneamente il valore di delta-time per quegli eventi è 0.

Il delta time rappresenta solo un modo per ricavare un evento midi ma non è un evento midi.

I meta eventi servono a memorizzare: i nomi delle tracce, la divisione del tempo, la velocità, la tonalità, il testo del brano, il copyright e altro.

I Meta-eventi vengono memorizzati nello SMF (Standard MIDI File) allo stesso modo dei Sys-Ex, ma iniziano con l'esadecimale FF.

Gli Exclusive messages servono per mandare messaggi midi esclusivamente ad un apparecchiatura o ad un gruppo di apparecchiature dello stesso tipo. Molte volte vengono utilizzati per poter programmare apparecchiature midi. Si possono, infatti con questi messaggi, impostare alcuni parametri proprietari di uno strumento.

Formati di file

I file MIDI si dividono in tre formati:

- 0, nel quale le tracce di un brano vengono mixate in una singola traccia che contiene però tutte le informazioni degli eventi di tutte le altre.
- 1, nel quale le tracce vengono memorizzate in modo singolo e contengono gli stessi valori di tempo e metrica. La velocità del brano viene inserita nella prima traccia che fa da riferimento a tutte le altre.
- 2, nel quale le tracce vengono gestite indipendenti l'una da l'altra con valori anche diversi di tempo e metrica.

I più utilizzati sono il formato 0 per i sequencer e il formato 1 per i programmi di editazione.

GM - GS - XG

GM, GS e XG sono degli standard.

Il general midi (GM) definisce una serie di convenzioni introdotte per unificare i parametri nell’esecuzione di una canzone. Per esempio, tutti gli strumenti che aderiscono a queste specifiche hanno come primo suono il pianoforte (classico), come secondo un pianoforte più " brillante”.

Lo standard GS è un evoluzione dello standard general MIDI level 1 ed è stato introdotto dalla Roland. Questo standard permette infatti di utilizzare più strumenti rispetto ai 128 consentiti

dal GM, e introduce nuovi tipi di messaggi e di controller. Il GS introduce il concetto di Bank Select, esso unito al program change permette di avere 128 variazioni per ogni singolo strumento. Il numero di suoni a disposizione in un sintetizzatore dipende esclusivamente dal produttore dello strumento e non dalle specifiche GS. Inoltre lo standard GS permette di modificare effetti audio come il chorus e il reverbero.

Lo standard XG fu introdotto dalla Yamaha nel 1994 e rappresenta un ulteriore evoluzione del GM e GS. Esso è lo standard più evoluto attualmente disponibile: è aumentato il numero di strumenti e il numero di effetti. Ovviamente lo standard XG è compatibile con lo standard GM, ossia se suoniamo una sequenza che rispetta le regole GM su di una periferica XG essa verrà suonata con gli stessi strumenti che gli erano stati fissati per il GM. Anche il viceversa vale, cioè suonare una base MIDI XG su di una periferica GM, ma si perdono tutti i vantaggi offerti dallo standard XG. Inoltre un sintetizzatore XG è in grado di riprodurre correttamente le sequenze GS, previa trasmissione di un comando GS Reset.

Metodi di collegamento

Una periferica MIDI può essere collegata:

- a PC
- Tramite porta giochi;
- Tramite interfaccia USB;
- ad altre periferiche
- direttamente
- Tramite Midibox

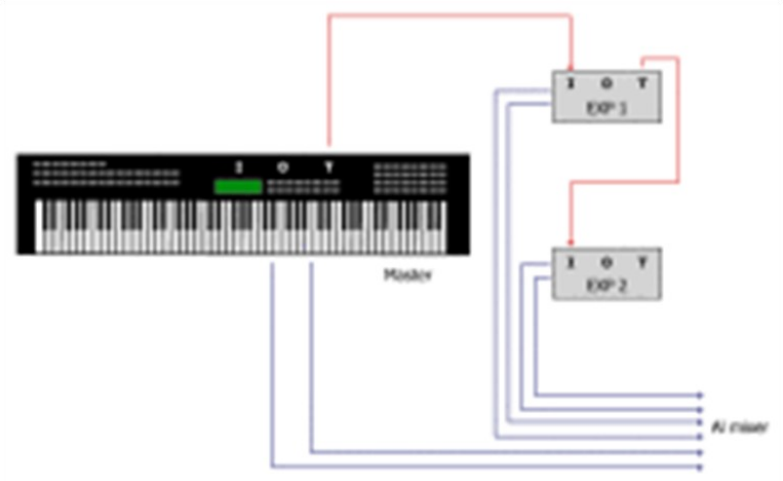

FIGURA 1.5 Un esempio di Daisy Chaining.

Esistono diversi metodi di collegamento per poter comunicare tramite midi:

- Daisy chaining o concatenazione a margherita nel quale vi è un dispositivo Master e una serie di dispositivi slave disposti in cascata;
- Configurazione con MIDI Thru Splitter Box che preleva il segnale in ingresso e lo invia su più uscite.

FIGURA 1.6 Un esempio di MIDI Thru Splitter Box

XML

XML è un linguaggio di markup basato su un meccanismo sintattico che consente di definire e controllare il significato dei contenuti. È un'evoluzione del linguaggio SGML che consente di definire in modo semplice nuovi linguaggi di markup. XML è acronimo di eXtensible Markup Language.

A differenza di HTML, che definisce una grammatica per la descrizione e la formattazione delle pagine web con dei tag limitati, XML è un metalinguaggio estendibile perché consente di crearne di nuovi.

I tag e le etichette

XML utilizza dei marcatori chiamati tag, o etichette, che possono contenere informazioni. Queste possono essere definite in due modi: racchiudendole in dei paramenti o come testo.

Le etichette sono delimitate da delle parentesi angolari <> e la chiusura del tag, come in HTML è data dalla barra /. Devono essere sempre aperti e chiusi e non possono cominciare con caratteri speciali o numeri.

```
<tagEsempio parametro1="etichetta di esempio">contenuto</tagEsempio>

<tagesempio2 parametro1="autochiudente" caratteristiche="senza chiusura" />
```

Due esempi di etichette. La seconda non ha bisogno di un'etichetta per la chiusura perché è gia presente alla fine.

XML è case sensitive.

Per essere ben formattato un file deve contenere un prologo e un unico nodo radice e tutti i tag devono essere bilanciati.

Tecnologie di supporto

Nel corso degli anni sono stati sviluppati diversi linguaggi schema, utilizzati per crearne nuovi XML. Uno di questi è il DTD, Document Type Definition, che serve a specificare l'insieme degli elementi, le relazioni gerarchiche, l'ordine di apparizione e quali elementi e attributi sono obbligatori o opzionali tramite una serie di regole.

Sono state sviluppate anche alcune interfacce di programmazione per consentire una più semplice creazione, lettura e gestione del file XML. Tra queste è presente SAX, Simple API for XML, che consente la lettura e l'elaborazione dei documenti. SAX è implementata in diversi

linguaggi ed è event based e processa ogni linea dei documenti. Il flusso di dati è unidirezionale in modo tale che i dati non possano essere letti senza elaborare l'intero documento.

```
<!ELEMENT graphic_instance (graphic_event+, rights?)>
<!ATTLIST graphic_instance
     description CDATA #IMPLIED
     position_in_group CDATA #REQUIRED
     file_name CDATA #REQUIRED
     file_format %formats; #REQUIRED
     encoding_format %formats; #REQUIRED
     measurement_unit (centimeters | millimeters | inches | decimal_inches
| points | picas | pixels | twips) #REQUIRED>
```

Un esempio preso dal DTD dello standard IEEE 1599

Un'altra API utilizzata è DOM, Document Object Model. Rappresenta i documenti come modelli orientati agli oggetti ed è tree based perché gestisce il documento xml come un albero in cui ogni nodo corrisponde a un elemento del file. DOM è lo standard W3C ed è implementata in diversi linguaggi di programmazione. Le specifiche DOM sono divise in 4 livelli che vanno da 0 a 3. Per appartenere a un livello l'applicazione deve rispettare tutti i suoi requisiti e di quelli inferiori.

Lo standard IEEE1599 e l'XML musicale

Lo standard IEEE1599 è stato sviluppato fornire una rappresentazione completa in tutti i suoi livelli di un brano musicale e utilizza il metalinguaggio XML. Supporta file audio, grafici e video.

Ha lo scopo di consentire la sincronizzazione di un file sonoro con uno spartito.

È diviso in più livelli. Il primo contiene tutti i dati generali del brano, tra cui il nome del brano, l'autore, la pubblicazione. Questo è detto <general>.

Il secondo contiene tutte le informazioni riguardanti gli eventi. È divisa a sua volta in due sottosezioni: <spine> e <los>. La prima è la spina dorsale della sezione, contiene tutti gli eventi precisandone il tipo, il timing e la posizione. La seconda specifica il tipo di evento, alla battuta di appartenenza, la durata, le note che ne fanno parte, se sono legate alla nota successiva e se sono puntate.

Seguono il livello <structural> e <notational>. In quest'ultimo tutti gli eventi vengono messi in relazione con la parte di spartito alla quale fanno riferimento.

Il quinto è <performance> e infine gli eventi sono sincronizzati con la traccia nel livello <audio>.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "http://standards.ieee.org/downloads/1599/1599-
2008/ieee1599.dtd">
<ieee1599 version="1.0">
     <general>
          <description>
          </description>
     </general>
     <logic>
          <spine>
          </spine>
          <los>
          </los>
     </logic>
     <audio>
     </audio>
</ieee1599>
```

Struttura di un file XML vuota.

<general>

Nel layer general contiene i metadati riguardanti la musica, al catalogo e le informazioni che descrivono il brano. Alcune di queste sono il titolo del brano e il compositore che sono racchiuse nel tag <description>.

Il titolo ha etichetta <main_title> mentre il compositore ha tag <author> con parametro type settato a composer. Altri tag presenti nello standard sono il <work_title>, che serve a impostare il titolo del movimento, il <work number>, il <date>, che contiene la data di pubblicazione, i generi con etichetta <genres>, che può contenere più generi che sono racchiusi in altri tag <genre>.

```
<general>
     <description>
          <main_title>esempio3</main_title>
          <author type="composer" />
     </description>
</general>
```

Parte di un layer general compilato.

<logic>

Il layer logic fornisce una descrizione logica dei simboli del pentagramma ed è diviso in due parti: <spine> e <los>.

Nello spine sono presenti tutti gli eventi con i relativi parametri di timing, che rappresenta lo sfasamento a livello temporale, e hpos, che rappresenta l'allineamento verticale e lo sfasamento orizzontale.

```
<spine>
     <event id="TimeSignature_track_1" timing="0" hpos="0" />
     <event id="KeySignature_track_1" timing="0" hpos="0" />
     <event id="Clef_track_1" timing="0" hpos="0" />
     <event id="track_1_measure_0_ev_0" timing="0" hpos="0" />
     <event id="track_1_measure_0_ev_1" timing="0" hpos="0" />
     <event id="track_1_measure_0_ev_2" timing="0" hpos="0" />
     <event id="track_1_measure_0_ev_3" timing="0" hpos="0" />
     <event id="track_1_measure_0_ev_4" timing="0" hpos="0" />
     <event id="track_1_measure_1_ev_0" timing="0" hpos="0" />
     <event id="track_1_measure_1_ev_1" timing="0" hpos="0" />
     <event id="track_1_measure_2_ev_0" timing="0" hpos="0" />
     <event id="track_1_measure_3_ev_0" timing="0" hpos="0" />
     <event id="track_1_measure_3_ev_1" timing="0" hpos="0" />
     <event id="track_1_measure_4_ev_0" timing="0" hpos="0" />
     <event id="track_1_measure_5_ev_0" timing="0" hpos="0" />
</spine>
```

Un esempio di spine compilato.

All'interno del tag <los>, Logically Organised Symbols, vengono descritti i simboli degli eventi del brano. Questa parte è divisa a sua volta in più sottosezioni. La prima è la <staff_list> che contiene la descrizione del tempo nella <time_signature>, della chiave in <clef> e delle alterazioni della scala in <key_signature>. In tutti e tre i tag si deve settare il parametro event_ref che specifica l'evento dello spine al quale si riferisce.

All'interno del tag <time_signature> si deve specificare il <time_indication>, cioè la durata della battuta, tramite i parametri num e den.

Nella <key_signature> si devono specificare, tramite il parametro number delle etichette <sharp_num> e <flat_num>, il numero di diesis o di bemolle presenti nella scala.

Infine, nel tag <clef>, vengono specificati la chiave, nel parametro shape, il paramentro staff_step, che indica la posizione verticale della chiave, ed octave_num che codifica il number che c'è sopra o sotto alla chiave.

La seconda sottosezione è contenuta nell'etichetta <part>, che contiene l'abbinamento tra la voce e lo spartito e la descrizione di tutti gli accordi e le pause.

Per l'abbinamento viene utilizzato il tag <voice_list> che contiene al suo interno un altro tag, il <voice_item>, nel quale si devono settare i parametri id, che identifica la voce, e staff_ref, che fa riferimento alla riga sullo spartito.

Gli accordi sono specificati tramite l'etichetta <chord>. Al loro interno vengono definite la durata, tramite i parametri num e den del tag <duration>, i punti delle note, tramite il parametro numbrer di <augmentation_dots>, e le note, tramite il tag <notehead> che a sua volta contiene il tag <pitch>, nel quale si devono specificare i parametri octave, che indica l'ottava, step, che indica la nota e actual_accidental, che specifica l'alterazione della nota e può prendere i valori natural, flat, sharp, double_flat, double_sharp. Sono considerate accordi anche le note singole.

Le pause sono rappresentate dal tag <rest> e, come per gli accordi, deve essere impostata la durata tramite i parametri num e den del tag <duration>.

Sia per le pause, che per gli accordi deve essere specificato l'evento a cui si riferiscono tramite il parametro event_ref.

```
<los>
  <staff_list>
    <staff id="track_1_staff" line_number="1">
      <time_signature event_ref="TimeSignature_track_1">
        <time_indication num="1" den="1" />
      </time_signature>
      <key_signature event_ref="KeySignature_track_1">
        <sharp_num number="0" />
      </key_signature>
      <clef shape="G" event_ref="Clef_track_1" staff_step="2" octa-
ve_num="0" />
    </staff>
  </staff_list>
  <part id="track_1">
    <voice_list>
      <voice_item id="track_1_voice" staff_ref="track_1_staff" />
    </voice_list>
    <measure number="0">
    <voice voice_item_ref="track_1_voice">
      <chord event_ref="track_1_measure_0_ev_0">
        <duration num="1" den="8" />
        <augmentation_dots number="1" />
        <notehead>
          <pitch octave="4" step="G" actual_accidental="natural" />
        </notehead>
      </chord>
      <rest event_ref="track_1_measure_0_ev_1">
        <duration num="1" den="16" />
      </rest>
```

Un esempio di los compilato.

<notational>

Il tag <notational> contiene la rappresentazione grafica dello spartito. Le rappresentazioni possono essere di due tipi: notational e graphical.

La prima fa riferimento al <notational_istance_group> che ha un parametro description e può contenere più <notational_istance> che a sua volta contiene più <notational_event>.

La seconda fa riferimento al <graphic_istance_group> che ha un parametro description e può contenere più <graphic_istance>. Queste hanno diversi parametri configurabili. Il primo è file-name che contiene il percorso del file al quale si vuole fare riferimento, il secondo è il file_format che permette di impostare il formato del file, il terzo è l'encoding_format che imposta la codifica e infine vi è position_in_group che indica la posizione nel gruppo di file che si sta sincronizzando.

All'interno di <graphic_istance> vi sono gli eventi che hanno tag <graphic_event>. I loro parametri sono l'evento di riferimento, event_ref, e l'area che occupano nel file descritta utilizzando le coordinate dei due angoli del rettangolo che racchiude la nota tramite i parametri upper_left_x, upper_left_y, lower_right_x e lower_right_y.

```
<notational>
  <graphic_instance_group description="Spariti">
    <graphic_instance
      file_name="C:\Users\Utente\Desktop\ProgettoA\ProgettoA.tif"
      file_format="image_tiff" encoding_format="image_tiff"
      position_in_group="1" measurement_unit="pixels">
        <graphic_event event_ref="track_1_measure_0_ev_0" upper_left_x="500"
          upper_left_y="354" lower_right_x="520" lower_right_y="390" />
        <graphic_event event_ref="track_1_measure_0_ev_1" upper_left_x="582"
          upper_left_y="334" lower_right_x="614" lower_right_y="388" />
        <graphic_event event_ref="track_1_measure_0_ev_2" upper_left_x="620"
          upper_left_y="332" lower_right_x="634" lower_right_y="386" />
        <graphic_event event_ref="track_1_measure_0_ev_3" upper_left_x="640"
          upper_left_y="332" lower_right_x="662" lower_right_y="392" />
        <graphic_event event_ref="track_1_measure_0_ev_4" upper_left_x="668"
          upper_left_y="334" lower_right_x="692" lower_right_y="388" />
        <graphic_event event_ref="track_1_measure_1_ev_0" upper_left_x="712"
          upper_left_y="334" lower_right_x="730" lower_right_y="388" />
        <graphic_event event_ref="track_1_measure_1_ev_1" upper_left_x="744"
          upper_left_y="350" lower_right_x="768" lower_right_y="390" />
    </graphic_instance>
  </graphic_instance_group>
</notational>
```

Un esempio di una parte di layer notational compilato.

<audio>

L'elemento <audio> contiene tutte le informazioni per sincronizzare gli eventi con la traccia audio del brano o con un video.

Ogni file viene abbinato inserendo il percorso nel parametro file_name del tag <track>. I parametri file_format e enconding_format servono per definire il formato e la codifica del file.

Segue il tag <track_indexing> che serve a definire, tramite il parametro timing_type, il tipo di parametro che serve a definire quando comincia l'evento.

All'interno del <track_indexing>, gli eventi vengono mappati con il file tramite il tag <track_event> che ha come parametri event_ref, l'evento a cui si riferiscono, e start_time, che contiene il secondo in cui comincia l'evento.

```
<audio>
  <track file_name="esempio3.mid" file_format="audio_x_midi"
    encoding_format="audio_x_midi">
      <track_indexing timing_type="seconds">
        <track_event event_ref="TimeSignature_track_1" start_time="0.00" />
        <track_event event_ref="KeySignature_track_1" start_time="0.00" />
        <track_event event_ref="Clef_track_1" start_time="0.00" />
        <track_event event_ref="track_1_measure_0_ev_0" start_time="0.0" />
        <track_event event_ref="track_1_measure_0_ev_1" start_time="0.09" />
        <track_event event_ref="track_1_measure_0_ev_2" start_time="0.12" />
        <track_event event_ref="track_1_measure_0_ev_3" start_time="0.3" />
        <track_event event_ref="track_1_measure_0_ev_4" start_time="0.36" />
        <track_event event_ref="track_1_measure_1_ev_0" start_time="0.48" />
        <track_event event_ref="track_1_measure_1_ev_1" start_time="0.84" />
        <track_event event_ref="track_1_measure_2_ev_0" start_time="0.96" />
        <track_event event_ref="track_1_measure_3_ev_0" start_time="1.44" />
        <track_event event_ref="track_1_measure_3_ev_1" start_time="1.68" />
        <track_event event_ref="track_1_measure_4_ev_0" start_time="1.92" />
        <track_event event_ref="track_1_measure_5_ev_0" start_time="2.4" />
      </track_indexing>
  </track>
</audio>
```

Un esempio di layer audio compilato.

<structural> e <performance>

Il tag <structural> identifica gli oggetti musicali e le loro relazioni. In questo layer è possibile collegare le parti del brano ai nodi di una rete di Petri.

L'elemento <performance> contiene gli elementi che forniscono una descrizione del brano tramite linguaggi o protocolli, come il MIDI o cSound.

Questi due layer e il livello notational non vengono utilizzati nel prototipo.

Java

Java è un linguaggio di programmazione multipiattaforma, cioè è indipendente dalla piattaforma di esecuzione. È un linguaggio orientato agli oggetti.

È stato sviluppato nel 1995 da Sun Microsystems ed è stato acquistato nel 2010 da Oracle.

Java deve gran parte della sua sintassi ad altri linguaggi come C e C++, al quale è molto simile fatta eccezione per l'aritmetica dei puntatori, l'eredità multipla delle classi e l'istruzione go to che non sono state implementate perché ritenute fonti di complessità non necessarie e che favoriscono l'introduzione di bug.

FIGURA 2.1 Un immagine del logo di Java

Java SE 7

Per lo sviluppo del prototipo si è utilizzata la versione di Java SE 7. Rilasciata il 28 luglio 2011, è la prima prodotta da Oracle dopo l'acquisto di Sun Microsystems e introduce numerose novità rispetto alle versioni precedenti, anche se, nonostante il ritardo della data di rilascio, sono stati apportati diversi tagli alle migliorie previste inizialmente.

Innanzitutto sono state migliorate le prestazioni, la stabilità e la sicurezza.

Inoltre sono stati introdotti dei miglioramenti al linguaggio di programmazione che permettono di scrivere e ottimizzare il codice con maggiore facilità. Un esempio è l'apertura dei file e la gestione delle eccezioni che ne sono connesse. I cambiamenti che riguardano l'ottimizzazione del codice sono parte del Progetto Coin.

Altre migliorie implementate riguardano la notazione numerica. È stata introdotta la notazione binaria togliendo la necessità di utilizzare il metodo parseByte() e introducendo il prefisso *0b* per poterne usufruire. Inoltre viene data la possibilità di utilizzare il carattere separatore '_' per

```
int i = 0b01010110;
long aLong = 100_000_000_000L;
```

Nel primo esempio si può vedere la nuova notazione binaria del numero 86, mentre nel secondo si può vedere come è possibile separare tramite il carattere _ il numero 100000000000.

migliorare la leggibilità dei numeri. In questo modo è possibile leggere più facilmente anche i grandi numeri come i miliardi.

Un altro miglioramento è dato dalla possibilità di utilizzare le stringe all'interno del costrutto switch. Ciò porta un alleggerimento del codice ma può introdurre nuovi bug. Per evitarli bisogna gestire delle eccezioni oppure limitare all'utente le scelte di stringhe da utilizzare tramite delle enumerazioni, dove si hanno dei valori conosciuti a priori perché definiti nell'enumerazione stessa.

```
public static boolean getBoolean(String string)
{
     switch(string)
     {
     case "YES": return true;
    case "NO": return false;
  }
  throw new IllegalArgumentException(string);
}
```

Un esempio di codice in cui viene utilizzato un costrutto switch con delle stringhe.

Per la gestione delle eccezioni sono stati introdotti il multicatch, cioè la possibilità di racchiuderne diverse in un'unica istruzione separandole con un operatore OR, e delle migliorie nel comportamento al runtime.

```
try{
      midifile = MidiSystem.getMidiFileFormat(infile);
}
catch (InvalidMidiDataException | IOException e)
{
     e.printStackTrace();
     System.exit(1);
}
```

Un esempio di multi catch utilizzato nel prototipo quando si tenta di ottenere il formato del file MIDI.

Un'altra caratteristica introdotta in Java 7 è la "Type inference for generic istance creation", cioè la deduzione automatica di tipo per la creazione di un'istanza generica, che riduce anch'essa la scrittura di codice quando si dichiara una variabile di tipo generico utilizzando

l'operatore diamond "<>", che può essere tradotto come rombo, ed evitando di ripetere i parametri per gli oggetti già istanziati.

```
ArrayList<nota> note;
note = new ArrayList<>();
```

Un esempio di Type inference for generic istance creation utilizzato nel prototipo quando si tenta di ottenere il formato del file MIDI.

Strutture dati

Come anticipato in Java non viene data la possibilità di usufruire di alcuni tipi di variabile come i puntatori. Per consentire di utilizzare alcune strutture di dati derivate da questi ultimi, per esempio le liste, sono stati implementati altri tipi di variabili che consentono di ovviare al problema. Tra queste si ricorda arraylist.

Un altro tipo di variabile non implementato è la struttura. Per creare elementi che contengono più campi di tipi diversi bisogna utilizzare le classi.

ArrayList

ArrayList è un tipo di variabile che viene utilizzato per creare liste dinamiche.

Per poterne usufruire bisogna importare java.util.Arraylist, una classe di Java che contiene tutte le funzioni.

L'istruzione di dichiarazione è:

```
ArrayList<<tipo>> <nome>
```

dove <tipo> è riferito al tipo di variabile scelto per i nodi della lista e <nome> è il nome della variabile.

La classe contiene anche diverse funzioni che ne semplificano l'utilizzo. Una di queste è add() che serve per aggiungere degli elementi. L'istruzione per utilizzarla è:

```
<variabile>.add(<elemento>[, <indice>])
```

dove per <variabile> si intende il nome della lista alla quale si vuole aggiungere l'elemento, indicato da <elemento>, e indice è il numero della posizione dove si vuole inserire. Quest'ultimo è facoltativo e deve essere compreso tra 0 e la dimensione della lista decrementata di 1.

Altre due funzioni utilizzate nel prototipo sono remove e get. Entrambe servono per estrarre degli elementi dalla lista ma lo fanno in modo diverso: la prima li rimuove fisicamente, la seconda li esporta ma ne lascia una copia. Entrambi necessitano di un indice come parametro ma per remove() è possibile utilizzare anche l'elemento da rimuovere.

Inoltre all'interno della classe ArrayList c'è una funzione che serve a sapere quanti elementi sono presenti al suo interno e una che serve a sapere se è vuota. La prima è size() e che ha in uscita un intero, mentre la seconda è isEmpty() che rimanda un booleano. Entrambe le funzioni non necessitano di parametri.

Classi

La classe è una struttura astratta che consente di creare nuovi oggetti e di definire le loro proprietà, dette attributi, e le loro funzioni, dette metodi, che verranno utilizzate per agire su di essi.

Le classi hanno uno o più metodi costruttori che devono avere lo stesso nome della classe e che sono usati per inizializzare gli attributi quando una classe viene istanziata. Nel caso in cui ce ne siano più di uno, questi devono avere comunque lo stesso nome, questo è possibile perché Java consente l'overloading di funzioni, la possibilità di avere più finzioni con lo stesso nome ma con parametri diversi. Questo vale per ogni metodo.

```
Class <nomeclasse>{
      [<tipo> <attributo1>
      <tipo> <attributo2>
      …
      <tipo> <attributoN>]

      [<tipo>  <nome_metodo>([<tipo><parametro1>,  <tipo><parametro2>,  …  ,
<tipo><parametroN>])
      {
          <istruzioni>
      }
}
```

Un esempio generico di una classe.

L'istruzione utilizzata per creare una classe è class seguita dal nome della classe e dalle parentesi graffe. All'interno di queste ultime si potranno inserire gli attributi e i metodi.

Librerie

Java implementa diverse librerie per gestire diversi protocolli e diversi tipi di file. Tra queste si sottolinea javax.sound.midi e org.jdom2.

La prima serve ad accedere a una serie di classi e di funzioni utili per creare, leggere e modificare file MIDI e gli eventi e utilizzare sequencer.

La seconda consente di usufruire di classi e funzioni che servono a creare, leggere e modificare file XML.

Javax.sound.midi

La libreria javas.sound.midi contiene le classi e interfacce utili per la gestione del protocollo MIDI. All'interno di queste sono presenti attributi e funzioni che semplificano la gestione dei file e della comunicazione. Nel prototipo sono utilizzate maggiormente le prime ed in particolare MidiEvent, MidiFileFormat, MidiMessage, Sequence e track.

MidiEvent serve per creare, leggere e gestire gli eventi. Contiene tre metodi: getMessage(), getTick(), SetTick(). Il primo ottiene un vettore di byte che compongono il messaggio, il secondo e il terzo servono per ottenere o settare il momento in cui si verifica l'evento espresso in tick.

```
MidiEvent evento = null;
…
evento = tracks[i].get(j);
messaggio=evento.getMessage();
…
tick=evento.getTick();
```

Parti del codice del prototipo in cui è possibile vedere come è possibile utilizzare la classe MetaEvent e i suoi metodi.

MidiFileFormat serve per ottenere informazioni in merito al formato, alla lunghezza del file, alla risoluzione, la durata in tick della nota da un quarto, e alle proprietà del brano. La classe contiene al suo interno sei attributi: byteLength che contiene la lunghezza del brano in byte; microsecondLength che contiene la durata in microsecondi; UNKNOW_LENGTH che si usa per segnalare che la lunghezza del file è sconosciuta; divisionType che indica com'è stato divi-

so il file; resolution che indica la durata della nota da un quarto; type che indica il tipo di file MIDI.

```
MidiFileFormat midifile = null;
Try
{
        midifile = MidiSystem.getMidiFileFormat(infile);
}
catch (InvalidMidiDataException | IOException e)
{
        e.printStackTrace();
        System.exit(1);
}
ppq=midifile.getResolution();
```

Parti del codice del prototipo in cui è possibile vedere come è possibile dichiarare e utilizzare la classe MIDIFileFormat e i suoi metodi.

All'interno di questa classe vi sono presenti anche sette metodi utilizzati per ottenere i valori degli attributi di cui si è appena parlato e delle proprietà del brano, tra cui il titolo e il nome dell'autore.

La terza classe serve per ottenere le informazioni di un messaggio. I suoi attributi sono data, un vettore di byte che contiene i dati del messaggio, e length, che contiene la lunghezza del messaggio. MidiMessage contiene inoltre cinque metodi: getMessage(), getSttus() e getLength() che permettono di ottenere i valori degli attributi; setMessage() che permette di creare un nuovo messaggio; clone() che crea un nuovo messaggio identico.

```
MidiMessage messaggio = null;
...
messaggio=evento.getMessage();
bytemsg = messaggio.getMessage();
msg = new int[bytemsg.length];
...
status = messaggio.getStatus();
```

Parti del codice del prototipo in cui è possibile vedere come è possibile dichiarare e utilizzare un MidiMessage e i suoi metodi.

La classe Sequence contiene le informazioni riguardanti le tracce. Contiene otto attributi: resolution e divisiontype che sono simili a quelli contenuti in MifiFileFormat; PPQ, SMPTE_24, SMPTE_25, SMPTE_30, SMPTE_30DROP che indicano il valore del timing a seconda del tipo scelto; tracks, il vettore che contiene le tracce con i loro eventi. Sono presenti inoltre otto metodi: getDivisionType() e getRisolution() che possono essere usati per ottenere il valori degli attributi; getMicrosecondLength() che, come in MidiFileFormat, ottiene la durata in microsecondi; getTickLength() che ottiene la durata della sequenza in tick; getPatchList(), che ottie-

ne una lista delle patch usate nella sequenza; getTracks(), che serve per ottenere un vettore contenente le tracce del brano; createTrack(), che serve per creare una nuova traccia e deleteTrack () che, invece serve per cancellarla.

```
Sequence sequence = null;
try {
     sequence = MidiSystem.getSequence(infile);
}
catch (InvalidMidiDataException | IOException e)
{
     e.printStackTrace();
     System.exit(1);
}
Track[] tracks = sequence.getTracks();
```

Parti del codice del prototipo in cui è possibile vedere come è possibile dichiarare e utilizzare la classe Sequence e i suoi metodi.

Infine Track contiene i metodi per gestire una traccia. In totale sono cinque: add() che permette di aggiungere un evento; get() che ottiene l'evento che si trova nella posizione che viene specificata; remove() che serve per rimuovere un evento; size() che ottiene in numero di eventi presenti nella traccia; ticks() che ottiene la durata in tick della traccia.

Org.jdom2

La libreria org.jdom2 contiene delle classi che consentono per creare, modificare e gestire un file XML sotto forma di albero. Quelle utilizzate per il prototipo sono: Document, DocType, Element e XMLoutputter.

Document consente di creare il documento a livello logico. Il suo metodo costruttore accetta in ingresso il nodo radice dell'albero. Nel prototipo è stato usato anche setDocType() per poter impostare il tag DOCTYPE del documento XML.

DocType è una classe che contiene i metodi per creare il doctype. Il metodo costruttore accetta in ingresso il nome, il valore di systemID e quello del publicID. Nessuno di questi attributi è obbligatorio. Nel progetto sono stati settati il nome e sytemID con l'indirizzo del dtd contenente le specifice per lo standard IEEE 1599.

```
Element root = new Element("ieee1599");
Document document = new Document(root);
DocType dt1 = new DocType("ieee1599",
          "http://standards.ieee.org/downloads/1599/1599-2008/ieee1599.dtd");
document.setDocType(dt1);
```

Parti del codice del prototipo in cui è possibile vedere come è possibile dichiarare un nuovo documento e impostare il suo DocType e il suo elemento radice.

Element contiene i metodi utilizzati per gestire ogni nodo dell'albero. Nel suo costruttore si può specificare il nome dell'elemento. Gli altri metodi presenti nel prototipo sono addContent (), setAttribute() e setText().

```
Element descr1 = new Element("main_title");
descr.addContent(descr1);
if (!tracks[0].nome.isEmpty())
{
     String arg = tracks[0].nome;
     descr1.setText(arg);
}
```

Parti del codice del prototipo in cui è possibile vedere la dichiarazione di un nuovo elemento e l'utilizzo dei metodi setAttribue() e setText()

Il metodo addcontent() viene utilizzato per aggiungere un nodo figlio, mentre setAttribute() e setText() permettono di impostare gli attributi degli elementi o il loro testo.

XMLoutputter è la classe che serve per creare fisicamente il file XML. I metodi che vengono utilizzati nel prototipo sono: setFormat() e output(). Il primo serve per impostare il formato di scrittura del file con le impostazioni per l'identazione. Quello utilizzato per il progetto è il PrettyFormat che usa due spazi per l'identazione, la codifica UTH-8 e non espande le dichiarazioni vuote.

Il secondo metodo serve per creare il file o per stampare a video un documento. Nel primo caso i parametri da inserire sono il nome del documento e il nome del file XML da creare, mentre nel secondo caso il nome del file dovrà essere System.out.

```
XMLOutputter outputter = new XMLOutputter();
outputter.setFormat(Format.getPrettyFormat());
outputter.output(document, new FileOutputStream("out.xml"));
out("File creato:");
outputter.output(document, System.out);
```

Parti del codice del prototipo in cui è possibile vedere la creazione di un nuovo file XML e l'utilizzo della classe XMLOutputter e dei metodi setFormat() e output()

Progetto

Lo scopo del prototipo è di convertire un file dallo standard MIDI a quello IEEE1599, utilizzando le librerie per la gestione dei file MIDI implementate in Java e risolvendo i problemi riguardanti la tonalità, la durata delle note e le limitazioni del protocollo. Il programma, inoltre, gestisce la sincronizzazione tra il file XML e quello MIDI.

Per farlo sono stati gestiti e risolti i problemi riguardanti la durata delle note utilizzando le funzioni messe a disposizione da Java. In particolare sono stati usati i due byte, presenti all'interno del blocco di intestazione del file MIDI, utilizzati per definire la risoluzione, cioè la durata della nota di un quarto.

Il linguaggio di programmazione scelto per sviluppare il progetto è stato Java. È stata fatta questa scelta perché è multipiattaforma.

Un'altra motivazione che ha portato a questa scelta è data dal fatto che Java contiene la libreria javax.sound.midi che permette di usufruire di un insieme di classi e di funzioni che permettono una gestione del protocollo più facile, agevolando la gestione degli eventi, delle tracce e dei sequencer. Consente, inoltre, di accedere facilmente anche ai dati relativi al formato e alle caratteristiche del midi.

Il programma doveva convertire un file MIDI in uno IEEE1599. I due standard presentano notevoli differenze dovute al diverso periodo di sviluppo e al loro scopo.

Limitazioni del protocollo MIDI

Lo standard MIDI è stato sviluppato per consentire l'iterazione tra più strumenti musicali. Non distingue i semitoni cromatici da quelli diatonici ma usa lo stesso valore a parità di frequenza, inoltre non ha una gestione della durata in quarti ma memorizza la distanza in tick, cioè in impulsi, tra un evento e il successivo per definire quando inizia o finisce una nota.

Lo standard IEEE1599 è stato sviluppato fornire una rappresentazione completa di un brano musicale e utilizza il metalinguaggio XML. Supporta file audio, video e grafici.

Funzionamento del prototipo

Il prototipo è composto da cinque funzioni principali: apri, analisi, segmentazione, creaxml e main. Inoltre sono presenti altre due funzioni utilizzate per stampare messaggi di errore.

Il programma funziona da linea di comando, l'istruzione è composta dal nome del programma e dal nome del file da convertire.

Il prototipo comincia controllando che l'istruzione inserita sia corretta. In caso positivo apre il file, lo analizza, lo divide e lo converte, altrimenti stampa un errore ed esce dal programma. Se anche la funzione apri va a buon fine, il programma inizia ad analizzare il file aperto ricavando la risoluzione della nota da un quarto, con la quale calcola la durata in tick di una battuta da 4 quarti, e la sequenza, dalla quale ricava un array contenente tutte le tracce con i relativi eventi. A questo punto comincia l'analisi dei messaggi di ogni singola traccia con cui si creano gli accordi che saranno poi codificati in XML. Una volta finita questa fase gli accordi vengono divisi, adattandoli alle battute, e viene raffinata la loro durata aggiungendo i punti di valore.

Infine si crea un file XML contenente gli accordi creati e sincronizzandoli con il file MIDI.

```
public static void main(String[] args)
     {
          //controllo l'istruzione inserita dall'utente
          if (args.length != 1)
          {
               errore();
          }
          File infile = apri(args[0]);
          MidiFileFormat midifile = null;
          try{
               midifile = MidiSystem.getMidiFileFormat(infile);
          }
          catch (InvalidMidiDataException | IOException e)
          {
               e.printStackTrace();
               System.exit(1);
          }
          ppq=midifile.getResolution();
          duratabattuta = ppq * 4;
          Sequence sequence = null;
          try {
               sequence = MidiSystem.getSequence(infile);
          }
          catch (InvalidMidiDataException | IOException e)
          {
               e.printStackTrace();
               System.exit(1);
          }
          Track[] tracks = sequence.getTracks();
          traccia[] tracce;
          //analisi del file e dei suoi eventi
          tracce = analisi(tracks);
          segmentazione(tracce);
          creaxml(midifile, tracce, args);
     }
```

Parte di codice in cui è possibile vedere il metodo main del programma.

apri

La funzione apri prende in ingresso una stringa, che contiene il nome del file, e in uscita manda il file aperto.

```
private static File apri(String nome)
{
     File inFile = new File(nome);
     if (!inFile.exists())
     {
          out("Il file '"+nome+"' non esiste!");
          errore();
     }
     return inFile;
}
```

Parte di codice in cui è possibile vedere il metodo apri del programma.

Serve ad aprire il file MIDI da convertire e a controllare che il file esista. Per farlo crea la variabile inFile nella quale crea il file logico tramite il percorso inserito dall'utente. In seguito controlla che il file esista e, in caso negativo, stampa un errore ed esce dal programma, altrimenti ritorna al main passando il file aperto.

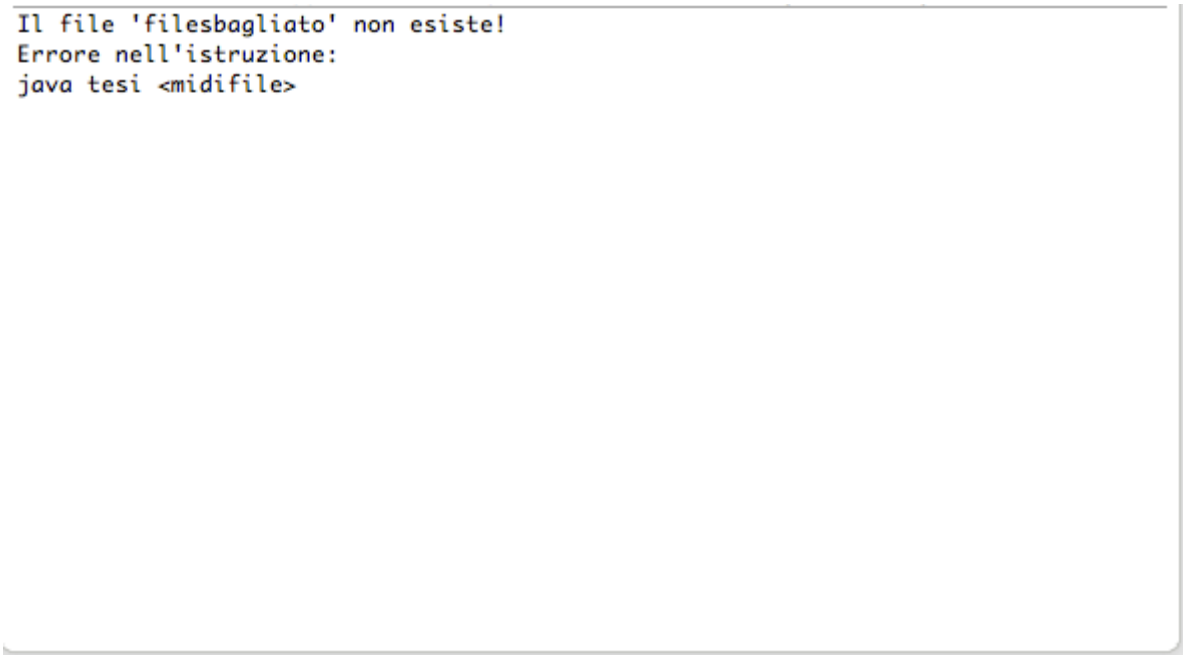

FIGURA 3.1 Messaggio di errore visualizzato quando il file non esiste.

analizza()

La funzione analizza prende in ingresso l'array contenente le tracce MIDI da analizzare e manda in uscita una lista dinamica di tracce contenenti i dati e gli accordi analizzati. Serve per analizzare gli eventi midi ed estrarne le informazioni utili a creare il file XML.

Il programma analizza per ogni traccia ogni evento che contiene. Quando incontra metaeventi, cioè quando lo status byte è uguale a 255, ne controlla il tipo, contenuto nel secondo byte del messaggio, e a seconda di esso si comporta in un modo diverso. Nel caso contenga le informazioni riguardanti il tempo metronomico, codice 81, converte i tre byte da esadecimale a decimale e salva il risultato, che verrà utilizzato in fase di sincronizzazione, in una variabile.

```
if (status == 255)
{
     switch (msg[1])
     {
          case 81:
               for (c=3;c<6;c++)
               {
                    hex += Integer.toHexString(msg[c]);
               }
               microtempo=Integer.parseInt(hex, 16);
               break;
          case 88:
               for (c=2;c<=msg[4];c++)
                    d*=2;
               tbatt=new tempo(msg[3],d);
               break;
          case 89:
               tonalitÃ = msg[3];
               modo = msg[4];
               break;
          case 3:
               for (c=0;c<msg[2]-1;c++)
               {
                    tracce[i].nome+= (char) bytemsg[c+3];
               }
               break;
     }
}
```

Parte di codice in cui il prototipo riconosce un metaevento e il suo tipo e decide come deve analizzar-

Se il codice è 88 allora il messaggio indica la durata in quarti di una battuta, quindi il programma calcola il denominatore e lo salva in una variabile di tipo tempo.

Nel caso in cui il messaggio abbia codice 89, vengono salvati la tonalità e il modo della scala. Mentre se il codice è uguale a 3 viene salvato il nome della traccia.

Se l'evento è un messaggio di note on, quindi il suo status byte è compreso tra 144 e 160, crea una nuova nota, imposta il valore e l'ottava e calcola la distanza in tempo dall'evento prece-

dente. Se questa ha un valore diverso da zero, il programma verifica se c sono note ancora attive, in caso negativo crea una pausa mentre in caso positivo crea un nuovo accordo con queste e la nota contenuta nel messaggio e setta il valore dell'attributo tie di quelle vecchie a true per indicare la legatura di valore. Inoltre, se le note attive si trovano in un accordo distante, le copia in tutti gli accordi intermedi.

Se la distanza è nulla, il programma aggiunge la nota all'ultimo accordo creato o, nel caso non ve ne siano, ne crea uno.

```
accordo.tick_inizio = tick;
accordo.note.add(notaccordo);
tracce[i].accordi.add(accordo);
vecttick[nota][0] = tracce[i].accordi.indexOf(accordo);
vecttick[nota][1] = accordo.note.indexOf(notaccordo);
```

Parte di codice in cui viene aggiunta una nota a un accordo.

Se l'evento è un messaggio di note off, cioè ha lo status byte compreso tra 128 e 144, il programma controlla se l'accordo ha già settato il valore tick_fine, che indica il momento in cui l'accordo finisce. In caso positivo controlla se questo è uguale al valore in tick del messaggio che si sta analizzando e, se sono diversi, crea un nuovo accordo con la nota e setta a true il campo tie di quella contenuta nell'accordo precedente. Nel caso in cui la nota sia stata accesa a distanza di più accordi, il programma l'aggiungerà in ogni accordo intermedio con il valore di tie settato a true.

Se il campo tick_fine dell'accordo è uguale a tick o non è stato ancora impostato, controlla che il campo tie sia falso e, nel secondo caso , setta il campo tick_fine uguale a tick.

segmentazione()

La funzione segmentazione ha in ingresso un array di tracce. Serve a dividere in battute il brano e a rifinire le durate delle note, aggiungendo i punti di valore.

Per ogni accordo di ogni traccia controlla se è maggiore della durata libera rimanente della battuta e, in caso negativo, lo aggiunge a quest'ultima, altrimenti lo spezza, lo aggiunge e crea un'altra battuta contenente la parte di accordo rimanente.

In musica la nota può essere rappresentata con un numero limitato di simboli che indicano la sua durata. Tendenzialmente quest'ultima è rappresentata sotto forma di frazione con numeratore uguale a 1. Quando il prototipo riceve e salva le note non controlla il numeratore, quindi,

in questa funzione, si controlla per ogni accordo il numeratore. Se è maggiore di 1, vengono calcolati i valori delle nuove note, sottratti a quella vecchia e aggiunti a una lista di tempi. Per ogni durata che vale la metà di quella successiva, si aggiunge un punto di valore, in caso contrario, o nel caso delle pause, si creano nuovi accordi.

Prima di passare all'accordo successivo la funzione controlla se l'elemento analizzato può essere parte di una terzina.

```
while (tacc.numeratore()>1)
{
     if (!tmp.isEmpty())
          tmp.remove(tacc);
     taccn = new tempo(1,accordo.duratabpm[1]);
     tacc = new tempo(accordo.duratabpm[0]-1,accordo.duratabpm[1]);
     accordo.duratabpm[0]= (byte) tacc.numeratore();
     accordo.duratabpm[1]= (byte) tacc.denominatore();
     tmp.add(taccn);
     tmp.add(tacc);
}
```

Parte di codice il prototipo riconosce una nota composta e si prepara a dividerla.

creaxml()

La funzione creaxml serve per creare il file XML contenente il brano convertito allo standard IEEE1599. Pende in ingresso un parametro MidiFileFormat che verrà utilizzato per ricavare le informazioni sul brano, un array contenente le tracce analizzate e una stringa contenente il nome del nuovo file.

Il file viene creato un layer alla volta. La prima parte è l'intestazione, nella quale viene inserito anche il riferimento al file dtd contenente le specifiche del formato. Poi viene creato il general, nel quale vengono inseriti il nome del brano e l'autore ricavati dalla fase di analisi.

Il terzo elemento è il logic. In questo viene creato prima lo spine, nel quale si inseriscono tutti gli eventi di ogni traccia ordinati per battuta ai quali viene assegnato un identificativo e impostato il valore di timing e hpos.

In seguito viene creato il los cominciando dalla staff_list, in cui vengono specificati gli eventi key_signature, time_signature e clef di ogni traccia, seguita dalla voice_list, in cui le voci vengono abbinare alla riga dello spartito a cui corrispondono. All'interno di questo elemento c'è anche la definizione delle battute e degli accordi che contengono. Per farlo vengono utilizzati dei for nidificati che scorrono le tracce, le battute, gli accordi e le note che poi la funzione converte nei relativi tag e parametri insieme a tutte le loro informazioni.

A questo punto il programma crea l'elemento audio in cui è presente la mappatura audio del brano. Per farlo crea il tag track, completa file_name con il percorso del file MIDI e imposta i suoi parametri file_format e encoding_format a audio_x_midi.

In seguito crea il track_indexing impostando il timing_type in secondi e inserendo al suo interno la lista di track_events per i quali calcola il tempo in cui l'evento comincia tramite la formula:

$$t = \frac{60000 \div bpm}{durata}$$

La divisione per 60000 serve a convertire il risultato da millisecondi a secondi mentre tpb indica la durata in tick della nota da un quarto e durata indica la distanza dell'evento dall'inizio e viene calcolata usando la formula:

$$durata = tick\_inizio \div ppq$$

dove tick_inizio rappresenta la distanza in tick dall'inizio.

```
for (m=0;m<tracks[i].battute.size();m++)
{
     batt=tracks[i].battute.get(m);
     s1=batt.accordi.size();
     for(j=0;j<s1;j++)
     {
          accordo=batt.accordi.get(j);
          event= new Element("track_event");
          trki.addContent(event);
          event.setAttribute("event_ref", "track_"+i+"_measure_"+m+"_ev_"+j);
          numquarti=(float)(accordo.tick_inizio)/(float)ppq;
          initime= (float) Math.floor((((float)(60000)/bpm))*numquarti)/1000;
          event.setAttribute("start_time", ""+initime);
     }
}
```

Parte di codice in cui vengono sincronizzati gli eventi con il file MIDI all'interno del layer audio.

Altre funzioni

Nel programma ci sono altre quattro funzioni: due utilizzate per la stampa a video, una che serve per riconoscere il nome della nota e l'ultima che serve per capire quale chiave è meglio utilizzare.

errore()

La prima funzione per la stampa a video è errore() che non ha parametri in ingresso e in uscita e viene utilizzata per stampare un messaggio di errore quando l'istruzione o il nome del file non sono corretti.

```
private static void errore()
{
     out("Errore nell'istruzione:");
     out("java tesi <midifile>");
     System.exit(1);
}
```

Codice della funzione errore().

out()

La seconda funzione per la stampa è out che ha in ingresso la stringa da stampare e serve a richiamare più rapidamente la funzione di stampa a video System.out.println().

```
private static void out(String strMessage)
{
     System.out.println(strMessage);
}
```

Codice della funzione out().

FIGURA 3.2 Messaggio di errore che viene visualizzato quando l'istruzione non è corretta.

chiave()

La funzione chiave() serve per decidere che chiave assegnare al rigo dello spartito. Consiste in sette cicli for che contano le note all'interno degli intervalli delle varie scale e in degli if che verificano quale strumento di sta analizzando e decide quali sono i contatori da prendere in considerazione. A seconda dei casi il contatore che ha in valere più alto decide quale chiave mandare in uscita.

In ingresso, la funzione ha due parametri: un vettore di byte e il codice dello strumento. Ogni cella dell'array rappresenta una nota e contiene il valore del numero di presenze di questa nel brano.

```
private static String chiave(byte[] note, int strumento)
{
...
     for (j=58; j<=77; j++)
     {
          chiavi[2]+= note[i];
     }
     for (j=53; j<=86; j++)
     {
          chiavi[3]+= note[i];
     }
     for (j=50; j<=71; j++)
     {
          chiavi[4]+= note[i];
     }
     for (j=47; j<=79; j++)
     {
          chiavi[5]+= note[i];
     }
     for (j=0; j<=65; j++)
     {
          chiavi[6]+= note[i];
     }
     if (strumento >=52 && strumento <=53)
     {
          j=1;
          for(i=2;i<6;i++)
          {
               if(chiavi[j]>chiavi[i])
               {
                    i=j;
               }
          }
          switch(i)
          {
          case 3:return"C0";
          case 4:return"C2";
          case 5:return"C4";
          case 6:return"C6";
          case 7:return"F4";
          case 2:return"F6";
          }
     }
```

Una parte del codice della funzione chiave().

nomenota()

nomenota() è una funzione che serve per ottenere il nome della nota in caratteri. Ha in ingresso un intero e in uscita una stringa. Contiene due switch, uno che contiene le note con i diesis e uno che ha quelle con i bemolle. In base alla tonalità del brano si decide a quale dei due accedere.

Si è dovuto aggiungere l'istruzione return "errore"; perché in caso alternativo non sarebbe stato possibile compilare ed eseguire il programma.

```
private static String nomenota(int nota)
{
     if(tonalitÃ  < 8 && tonalitÃ  >=0)
     {
          switch (nota)
          {
               case 0: return "C";
               case 1: return "C#";
               case 2: return "D";
               case 3: return "D#";
               case 4: return "E";
               case 5: return "F";
               case 6: return "F#";
               case 7: return "G";
               case 8: return "G#";
               case 9: return "A";
               case 10: return "A#";
               case 11: return "B";
          }
     }
     else
     {
          switch (nota)
          {
               case 0: return "C";
               case 1: return "Db";
               case 2: return "D";
               case 3: return "Eb";
               case 4: return "E";
               case 5: return "F";
               case 6: return "Gb";
               case 7: return "G";
               case 8: return "Ab";
               case 9: return "A";
               case 10: return "Bb";
               case 11: return "B";
          }
     }
     return "errore";
}
```

Codice della funzione nomenota().

Classi utilizzate nel prototipo

Non potendo usufruire di strutture, come struct in C, si è dovuto comporre il prototipo da sei classi.

La prima è quella principale che contiene le funzioni del programma utilizzate per convertire i file.

La seconda è chiamata nota. Questa ha al suo interno tre attributi: nota, ottava e tie. Il primo indica la nota che è stata attivata, il secondo indica l'ottava in cui si trova e il terzo indica se è presente la legatura di valore.

```
public class nota {
     String nome;
     int nota;
     int ottava;
     boolean tie = false;
}
```

Dichiarazione della classe nota.

La terza è chord, che contiene cinque attributi e un metodo costruttore. Le proprietà che possono essere salvate in questa classe sono: note, una lista di note; idevento e idtrk, il primo è un intero che indicizza gli eventi mentre il secondo indica a quale traccia appartiene; duratabpm, un array di byte che contiene i due valori della durata in forma di frazione; tick_inizio e tick_fine che memorizzano i tick di inizio e di fine dell'accordo e sono inizializzati a -1; dots che conta i punti di valore.; due variabili di tipo tempo triplet e intriplet che servono per gestire le terzine: la prima contiene il valore totale delle note che compongono la terzina, mentre la seconda contiene il valore della durata totale della terzina.

Il metodo costruttore serve per inizializzare la lista e il vettore.

```
public class chord {
     ArrayList<nota> note;
     int idevento, idtrk;
     byte[] duratabpm;
     long tick_inizio =-1;
     long tick_fine =-1;
     int dots;
     tempo triplet, intriplet;

     public chord(){
          note = new ArrayList<>();
          duratabpm = new byte[2];
          triplet=intriplet= new tempo(0,1);
     }
}
```

Dichiarazione della classe chord.

La quarta classe è battuta che contiene una lista di accordi e il metodo costruttore che la inizializza.

```
public class battuta {
     ArrayList<chord> accordi;
     public battuta()
     {
          accordi = new ArrayList<>();
     }
}
```

Dichiarazione della classe battuta.

La quinta si chiama traccia. Contiene cinque attributi: nome, che indica il nome della traccia; tonalità, un intero che indica in termini numerici la tonalità della traccia; battute e accordi, due liste, una di battute e una di accordi; contnote, un vettore di contatori che conta le note della traccia.

In questa classe è presente un metodo costruttore che inizializza le liste, il nome e i contatori.

```
public class traccia {
     int tonalità ;
     String nome;
     ArrayList<battuta> battute;
     ArrayList<chord> accordi;
     byte[] contnote;

     public traccia(){
          accordi = new ArrayList<>();
          battute= new ArrayList<>();
          contnote = new byte[12];
          int i;
          for (i=0; i<12; i++){
               contnote[i]=0;
          }
          nome = "";
     }
}
```

Dichiarazione della classe traccia.

Infine vi è tempo che serve per gestire la durata delle note sotto forma di frazione. Contiene al suo interno due attributi, il numeratore num e il denominatore den, e diversi metodi: il primo è il costruttore, il secondo e il terzo servono per sommare e sottrarre altre frazioni, poi ci sono due funzioni che restituiscono rispettivamente il numeratore e il denominatore e una che calcola il massimo comun divisore, che viene utilizzato per la semplificazione. Vi è anche una funzione che compara la frazione con un'altra indicando se è maggiore.

Conclusioni

Nell'elaborato sono state analizzate le tecnologie utilizzate e si è visto il loro funzionamento all'interno del prototipo sviluppato. Si è spiegato anche il funzionamento del programma nei suoi metodi principali.

In questo paragrafo vengono analizzati i principali utilizzi e le funzionalità che potranno essere implementate in futuro.

Lo standard IEEE 1599 può essere utilizzato nell'ambito dell'educazione musicale per semplificare l'apprendimento e il perfezionamento della tecnica di lettura degli spartiti sia per chi è all'inizio del percorso musicale sia per chi è a un livello avanzato.

Inoltre può semplificare l'apprendimento anche per persone portatrici di disabilità fisiche e psicologiche.

Il prototipo può semplificare la creazione dei nuovi file che vengono utilizzati per questi scopi.

Il prototipo, in futuro, può essere migliorato facendo in modo che riconosca anche gli elementi dinamici del brano analizzato e che distingua anche le note che vengono suonate piano da quelle che invece vengono suonate forte.

Un altro possibile sviluppo può essere l'implementazione di una modalità in cui si possa suonare seguendo la base e lo spartito. Questo può essere molto utile nell'ambito dell'educazione musicale perché consentirebbe di annotare gli errori di lettura e di tempo portando ad un conseguente miglioramento della tecnica dello studente.

Bibliografia

Baggi Dennis, Haus Goffredo – The new standard IEEE 1599, Introduction and Examples – Journal of multimedia volume 4 numero 1 – febbraio 2009

Baratè Adriano, Haus Goffredo, Ludovico Luca Andrea – IEEE1599: a new standard for music education – Innovation in Communication Paradigms and Technologies, pp. 29-45 – Edizioni Nuova Cultura, Milano - 2009

De Sio Cesari Claudio – Manuale di Java 7 – Hoepli – 2011

Ludovico Luca Andrea –Encoding music information – Wiley – IEEE computer society press – 2013

Selfridge-Field Eleanor – Beyond MIDI: the handbook of musical code – The MIT Press – 1990

Sitografia

http://www.mx.lim.di.unimi.it/reference_manual/index.php

http://it.wikipedia.org/wiki/Musical_Instrument_Digital_Interface

http://it.wikipedia.org/wiki/Java_(linguaggio_di_programmazione)

http://it.wikipedia.org/wiki/XML

http://www.agentgroup.unimo.it/didattica/curriculum/marco/MAIN/didattica/TECN_APP_WEB/PDF/javaxml.pdf

https://www.academia.edu/3401899/Elementi_di_Programmazione_MIDI

http://www.html.it/pag/15097/introduzione-alla-programmazione-java/