UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione
Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

PROTOTIPI SOFTWARE PER LA CONVERSIONE DA KERN A IEEE 1599

Tesi di laurea di
Giorgio RAPETTI
Matricola **722007**

Relatore: Prof. **Luca Andrea LUDOVICO**

Correlatore: Dott. **Adriano BARATÈ**

Anno Accademico 2009-2010

UNIVERSITÀ DEGLI STUDI DI MILANO

Facoltà di Scienze Matematiche, Fisiche e Naturali
Dipartimento di Informatica e Comunicazione
Corso di Laurea in Scienze e Tecnologie della Comunicazione Musicale

PROTOTIPI SOFTWARE PER LA CONVERSIONE DA KERN A IEEE 1599

Tesi di laurea di
Giorgio RAPETTI
Matricola **722007**

Relatore: Prof. **Luca Andrea LUDOVICO** _____

Correlatore: Dott. **Adriano BARATÈ** _____

Anno Accademico 2009-2010

INDICE

Introduzione.

Scopo del presente elaborato è di fornire uno strumento software che consenta la traduzione dell'informazione musicale contenuta da un file in formato **kern in documento XML conforme allo standard IEEE 1599, sviluppato presso il Laboratorio di Informatica Musicale (LIM), presso la sede universitaria di via Complico 39.
Prima di procedere alla presentazione dei quattro prototipi software realizzati, occorre fornire alcune informazioni di base sul linguaggio **kern e fare alcuni brevi cenni circa lo standard IEEE 1599.

Nel capitolo I troverà spazio la trattazione di **kern e del software Humdrum che consente la manipolazione dei file in formato **kern.

Nel capitolo II, invece, verranno elencati i tratti salienti del formato IEEE 1599, con particolare attenzione al livello logic ed ai suoi sublayer spine e los, che costituiscono il cuore di un file in formato IEEE 1599.

Il capitolo III è dedicato alla trattazione di un paio di approcci che si possono utilizzare per tradurre da **kern a IEEE 1599 utilizzando strumenti già esistenti: l'Online Humdrum Editor presente sul sito KernScores, Finale ed il plug-in per la conversione in IEEE 1599 creato presso il LIM.

Nel capitolo IV troverà spazio la presentazione dei prototipi software da me realizzati. Tale presentazione sarà corredata dalle parti più significative del codice scritto nel linguaggio di programmazione C# e da esempi dell'output generato dall'esecuzione dei software.

A conclusione dell'elaborato è posta un'appendice nella quale si troveranno i file **kern di origine degli esempi trattati nel capitolo IV ed altri esempi di output generato dall'esecuzione dei software.

CAPITOLO I

**kern

Humdrum

Humdrum[1] è un software general-purpose creato con lo scopo di fornire un valido aiuto ai ricercatori in campo musicale. Le capacità di Humdrum sono piuttosto ampie, quindi risulta difficile descrivere in maniera breve ma esauriente tutto ciò che Humdrum è in grado di fare. In questo paragrafo ci occuperemo di spiegare brevemente la sintassi di Humdrum e di fare alcuni accenni alle sue principali funzioni.
Prima di addentrarci nella trattazione di Humdrum, è utile sottolineare che esso consiste di due parti distinte: Humdrum Syntax e Humdrum Toolkit[2].

Humdrum Syntax

I dati Humdrum sono organizzati in modo molto simile alle tabelle di un foglio di calcolo; come in un normale foglio di calcolo è possibile definire ed etichettare i vari tipi di dati ed impostare le colonne di modo che ciascuna rappresenti un particolare tipo di dati. Le righe, invece, hanno un significato fisso: rappresentano la successione temporale degli eventi. Il tempo, quindi, avanza man mano che ci muoviamo verso il basso della tabella.

	Pitch	Duration	Valve Combination
1st note	C4	quarter	0
2nd note	B3	eighth	2
3rd note	G4	eighth	0
4th note	F4	eighth	1
5th note	G4	eighth	0
6th note	A4	quarter	1-2
7th note	G4	eighth	0
8th note	Ab4	quarter	2-3

**Note	**Pitch	**Duration	**Valve Combination
1st note	C4	quarter	0
2nd note	B3	eighth	2
3rd note	G4	eighth	0
4th note	F4	eighth	1
5th note	G4	eighth	0
6th note	A4	quarter	1-2
7th note	G4	eighth	0
8th note	Ab4	quarter	2-3
*-	*-	*-	*-

Entrambe le immagini qui a fianco mostrano due tabelle che rappresentano le stesse informazioni: in particolare, l'ultima colonna, contiene le combinazioni delle valvole di una tromba.

L'immagine più a sinistra mostra una semplice tabella, mentre l'immagine più a destra mostra (appena alla nostra sinistra) una tabella in cui i dati sono codificati secondo la sintassi di Humdrum.

Non ci sono particolari restrizioni su quali caratteri possano apparire nelle tabelle, se non per quanto riguarda l'utilizzo di spazi e tabulazioni. Non esiste una dimensione massima delle tabelle sia per il numero di colonne, sia per quello delle righe (ossia l'estensione temporale).
Bisogna, tuttavia, fare molta attenzione al comportamento delle colonne: a differenza di quanto succede con un normale foglio di calcolo, le colonne di Humdrum possono unirsi tra loro, dividersi, cambiare posizione, terminare in qualsiasi punto della tabella oppure iniziare successivamente alle altre. Dato che queste colonne possono "spostarsi" all'interno della tabella in modo semi-flessibile, esse sono conosciute anche con il nome "spines". Questo termine non è assolutamente da confondere con il sublayer "spine" di IEEE 1599, di cui parleremo in seguito.

Humdrum Toolkit

Il software Humdrum non assomiglia né ad un editor di testo né ad un software dedicato per la notazione musicale. Non è nemmeno un programma di grandi dimensioni: Humdrum, infatti, fornisce una serie di piccole e leggere utilities a cui è possibile accedere in qualsiasi momento e da qualsiasi parte del sistema.
Tutti i dati conformi alla sintassi Humdrum possono essere manipolati attraverso i tools che ci vengono messi a disposizione dal software. Esistono innumerevoli modalità di accesso e di manipolazione dei dati, poiché tali tools possono essere liberamente interconnessi tra loro ed il loro utilizzo è possibile anche congiuntamente ad altri strumenti non facenti parti del toolkit Humdrum.

Di seguito verranno elencati alcuni dei principali comandi[2][3] di Humdrum:

Extract. Questo comando consente all'utente di selezionare uno o più spines da un ingresso Humdrum. Generalmente viene utilizzato per estrarre parti da una partitura con più strumenti, tuttavia a volte può servire anche per isolare indicazioni dinamiche, testo di una canzone o altre informazioni che sono state codificate in uno spine. Il comando extract ha quattro diverse modalità di funzionamento: field mode (1), interpretation mode (2); spine-path mode (3) e field-trace mode (4), di seguito elencate nell'ordine.

```
1 - extract -f field1,field2,...,$-1,$ [inputfile ...]
2 - extract -i interp1,interp2,...,interpn [inputfile ...]
3 - extract -p spine#n [inputfile ...]
4 - extract -t field_trace_file.ftf [inputfile ...]
```

Nel primo caso vengono specificati i campi che devono essere estratti attraverso il numero di colonna; nel secondo caso, invece, verranno estratti tutte le colonne contenenti una particolare espressione (come può essere ad esempio "**MIDI"). Nel terzo caso si ha invece l'estrazione di uno spine dall'inizio alla fine, compresi eventuali cambi di path. Nell'ultimo caso, invece, viene accettato in input un file contenente i riferimenti ai campi da estrarre codificati come numeri di riga e di colonna.

Yank. Come extract, serve per estrarre delle informazioni da un input Humdrum. In questo caso, però, le informazioni estratte non corrispondono agli spines di cui è composto il file Humdrum, ma a specifici segmenti del file (sezioni, frasi, battute), individuati tramite i numeri di riga o i marcatori corrispondenti.

```
yank -s Trio -r 1 filename
yank -n = -r 114-183 filename
yank -o { -e } -r '$-1' filename
```

Nel primo esempio, si ha l'estrazione di una sezione tramite l'indicazione della sua etichetta ("trio"). Nel secondo esempio il materiale da selezionare viene individuato tramite l'indicazione dei numeri di misura. L'ultimo esempio mostra, invece, l'estrazione della penultima frase del file **kern preso in esame.

Mint. Tramite questo comando Humdrum ci dà la possibilità di generare informazioni circa gli intervalli melodici di un brano: esso, infatti, determina la distanza (in termini di pitch) esistente tra un suono ed il successivo. L'output generato dall'esecuzione di questo comando consiste nell'intervallo diatonico preceduto dal segno "+" (se l'intervallo è ascendente) o dal segno "–" se l'intervallo è discendente. Mint determina gli intervalli esclusivamente tra i suoni appartenenti allo stesso spine.

**kern	**mint
=1	=1
8c	[c]
8g	+P5
4.b-	+m3
.	.
8e	-d5
=2	=2
4f	+m2
8r	-P11
8C	-P5
4FF	=3
=3	*-
*-	

L'immagine a fianco mostra il funzionamento del comando mint. Nella colonna a sinistra abbiamo l'input, costituito da due battute di un file **kern. Nella colonna di destra, invece, abbiamo l'output generato in seguito all'esecuzione del comando mint. Si noti che la prima nota viene indicata tra parentesi graffe ed indica il suono di partenza. Ogni nota successiva viene calcolata come intervallo ascendente o discendente rispetto alla nota precedente. I null token, le stanghette (con gli eventuali numeri di battuta) e le pause non vengono computate dal comando mint e la loro presenza non influisce in alcun modo sull'output. In particolare null token e stanghette/numeri di battuta vengono mantenuti anche nell'output **mint, mentre le pause vengono ovviamente eliminate dall'output dato che non hanno alcun valore al fine del computo degli intervalli.

Cosa è possibile fare con Humdrum?

Il toolkit Humdrum contiene oltre 70 software interconnessi tra loro. Le informazioni musicali sono contenute in dati testuali codificati in forma ASCII[4], manipolabili attraverso i tools che Humdrum ci mette a disposizione: essi ci consentono di estrarre dal testo codificato in ASCII sia la musica sia le informazioni ad essa correlate.
Ma quali tipi di manipolazioni è possibile fare attraverso Humdrum? Al momento possiamo raggruppare i diversi tools in 16 gruppi di operazioni:

- **Visual display.** Permette di visualizzare una particolare istanza della partitura: ad esempio scegliere visualizzare una parte a partire da una determinate misura ed eseguirne l'analisi armonica;

- **Aural display.** Consente di scegliere una parte da far suonare ed impostare quante misure devono essere riprodotte, oppure da quale misura deve iniziare la riproduzione; è anche possibile decidere di far suonare esclusivamente un determinato numero di misure facenti parte dei brani selezionati;

- **Searching.** Uno strumento che di ricerca da utilizzare ad esempio per cercare tutte le volte che in un singolo brano o in un gruppo compare un motivo: consente di localizzare le cosiddette "cadenze d'inganno"[5], oppure di trovare tutti i brani in cui viene suonato un particolare strumento;

- **Counting.** La ricerca che viene fatta attraverso i tools preposti a svolgere questo tipo di operazioni è molto semplice: contare tutte le volte che si verifica un determinato fenomeno;

- **Editing.** Permette di modificare da 1 uno a n elementi o eventi del file **kern in esame;

- **Editorializing.** Consente di aggiungere note di spiegazione o di commento: (ad esempio indicare che la notazione del brano differisce da quella del manoscritto autografo del compositore);

- **Transforming or translating between representations.** Gruppo di tools che permettono di effettuare una trasposizione da una chiave ad un'altra e calcolare gli intervalli armonici tra due parti;

- **Arithmetic transformations of representations.** Tools che consentono di effettuare un'analisi aritmetica della partitura: calcolare i semitoni tra note successive o determinare i punti in cui le parti si incrociano relativamente all'altezza;

- **Extracting or selecting information.** Come di evince dal nome, questi tools permettono di selezionare uno o più eventi per estrarne le informazioni;

- **Linking or joining information.** Gruppo di tools che consentono l'assemblaggio di più parti in un'unica partitura completa, marcare le note aventi funzioni armoniche oppure sincronizzare il file **kern con un file audio;

- **Generating inventories.** Creare inventari circa le informazioni presenti nel file **kern in cui elencare quali siano quelle più comuni e quali quelle meno comuni. Determinare la presenza o l'assenza di determinare "chord functions";

- **Classifying.** Sostanzialmente i tools facenti parte di questo gruppo eseguono operazioni di classificazione degli elementi che compongono il file **kern: classificare accordi ed intervalli; identificare le dominanti secondarie; classificare le diteggiature per pianoforte in base alla loro difficoltà.

- **Labelling.** I tools facenti parte di questo gruppo svolgono prevalentemente funzioni di marcatura (es. per le sezioni di strumenti) e di identificazione (es. identificare il tipo di accordi utilizzati)

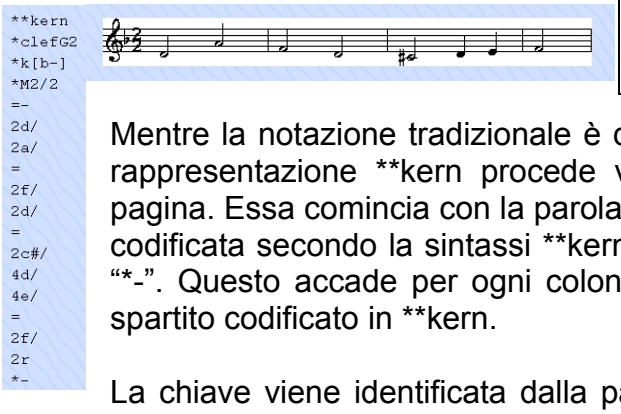
- **Comparison.** Tools che hanno lo scopo di comparare partiture dello stesso autore o di autori diversi ed indicarne similitudini e differenze.

- **Capturing Data.** Importare dati da altri formati (ad esempio una traccia MIDI[?] o uno spartito scritto con un programma per la notazione tipo Finale).

- **Trouble-shooting.** Identificare "trasgressioni" alla notazione convenzionale oppure ottenere un aiuto in particolari situazioni di difficile soluzione.

Rappresentare la musica utilizzando **kern

Il formato **kern[1][2][3][6] può essere utilizzato per rappresentare le informazioni musicali di base dei brani scritti nella notazione occidentale. Con **kern, infatti, possiamo codificare gran parte delle informazioni relative ad un brano musicale.
Generalmente **kern è destinato a rappresentare la base funzionale dell'informazione trasmessa da una partitura piuttosto che le sue peculiarità visive ed ortografiche: **kern è stato infatti progettato principalmente per facilitare le applicazioni analitiche, non già per la stampa dello spartito o la riproduzione di un brano. Tuttavia sia l'output visivo sia quello sonoro possono essere generati a partire dalla rappresentazione **kern.



L'immagine a sinistra mostra le prime quattro battute del brano "Die Kunst der Fuge" di Johann Sebastian Bach e la loro corrispondente traduzione in **kern.

Mentre la notazione tradizionale è disposta orizzontalmente lungo la pagina, la rappresentazione **kern procede verticalmente dall'alto verso il basso della pagina. Essa comincia con la parola chiave "**kern" che indica l'inizio della parte codificata secondo la sintassi **kern. La fine della partitura è indicata dal token "*-". Questo accade per ogni colonna, ovvero per ogni parte che compone lo spartito codificato in **kern.

La chiave viene identificata dalla parola chiave "*clef" seguita dalla sigla della nota corrispondente al tipo di chiave (G per la chiave di violino, F per quella di basso..etc…) e dal numero che indica la posizione della chiave sul pentagramma (ad esempio G2 è la chiave di violino, F4 quella di basso, etc...). Nella riga sottostante la chiave troviamo l'indicazione di tempo, che ha la seguente sintassi: la lettera "M" seguita dalla frazione che indica il tempo, scritta in maniera analoga a quanto avviene su un normale pentagramma (2/2, 4/4, 3/4 etc…).

Le linee di battuta sono solitamente indicate attraverso il carattere "=" che può essere seguito o meno dal numero di battuta ("=1", "=2" e così via). Inoltre in alcune partiture l'inizio della prima battuta è codificato con "=-" oppure "=1-": tali codifiche hanno solo valore funzionale e non trovano alcuna corrispondenza nella partitura stampata. In particolare, il simbolo "-" in questo contesto sta ad indicare che la linea di battuta è invisibile: non sarà quindi presente nel corrispondente spartito visualizzato o stampato.

La durata delle note è codificata attraverso numeri reciproci (vedi tabella 1), mentre la posizione delle stanghette sulla partitura si indica attraverso il carattere "/" quando esse sono rivolte verso l'alto e "\" quando invece sono rivolte in basso.

Nome nota	Valore nota	Codifica **kern
Semibreve	4/4	1
Minima	2/4	2
Semiminima	1/4	4
Croma	1/8	8
Semicroma	1/16	16
Biscroma	1/32	32
Semibiscroma	1/64	64
Fusa*	1/128	128

Tabella 1: Valore delle note e loro corrispondente nella codifica **kern. Il valore codificato in **kern corrisponde al reciproco del valore della durata della nota: ¼ = 4.

*Fusa: detta anche quintupla, è usata raramente: compare, ad esempio in alcuni movimenti lenti contenuti nelle composizioni di Ludwig van Beethoven.

Le pause vengono identificate tramite la lettera minuscola "r" preceduta dal reciproco della durata della pausa in notazione tradizionale, come avviene per la durata delle note (tabella 1).

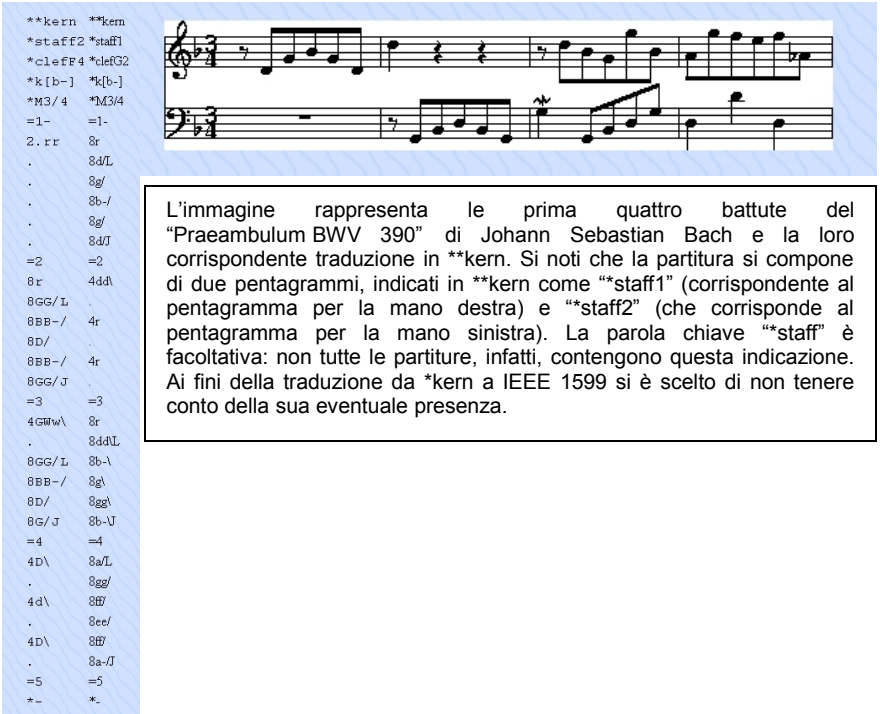
Di particolare interesse, invece, è il cosiddetto “null token”, rappresentato dal carattere “.” (punto): esso, infatti, non ha alcun valore ai fini della codifica dell’informazione musicale. La sua unica funzione è quella di mantenere ordinate le parti.
Il pitch è codificato attraverso l’utilizzo di sequenze di caratteri maiuscoli e minuscoli (tabella 2).

Ottava	Codifica **kern
C1	“CCC”
C2	“CC”
C3	“C”
C4	“c”
C5	“cc”
C6	“ccc”
C7	“cccc”

Tabella 2: Pitch codificato in **kern (colonna di destra). Ad ogni ottava corrisponde una sequenza di caratteri maiuscoli o minuscoli corrispondente al nome della nota in notazione anglosassone.

Le alterazioni sono codificate con i seguenti caratteri: “#” per i diesis, “-” per i bemolli e “n” quando la nota non presenta alterazioni e segue una nota alterata. Vengono poste successivamente alla lettera che codifica il nome della nota e la sua altezza. Le eventuali doppie alterazioni vengono rappresentate utilizzando 2 caratteri: “##” per il doppio diesis e “--” per il doppio bemolle. Eventuali triple alterazioni o alterazioni ancora maggiori vengono codificare attraverso la ripetizione dei caratteri “#” e “-” tante volte quante sono le alterazioni. È bene ricordare che anche in **kern le alterazioni sono mutuamente esclusive, quindi non ha senso scrivere note del tipo “4CC#n” oppure “2eee-#”.
Infine è importante fare alcuni brevi cenni circa l’utilizzo delle parentesi:

- La parentesi graffa aperta “{” indica l’inizio di una frase, mentre la graffa chiusa “}” indica la sua fine;
- Parentesi quadre aperte “[” e chiuse “]” denotano l’inizio e la fine delle cosiddette ties. Per l’indicazione delle eventuali note centrali di una tie, invece, viene utilizzato il carattere “_” (underscore);
- Le parentesi tonde “(” e “)” indicano rispettivamente l’inizio e la fine di una legatura di valore.



**kern	**kern
*staff2	*staff1
*clefF4	*clefG2
*k[b-]	*k[b-]
*M3/4	*M3/4
=1-	=1-
2.rr	8r
.	8d/L
.	8g/
.	8b-/
.	8g/
.	8d/J
=2	=2
8r	4dd\
8GG/L	.
8BB-/	4r
8D/	.
8BB-/	4r
8GG/J	.
=3	=3
4GWw\	8r
.	8dd\L
8GG/L	8b-\
8BB-/	8g\
8D/	8gg\
8G/J	8b-\J
=4	=4
4D\	8a/L
.	8gg/
4d\	8ff/
.	8ee/
4D\	8ff/
.	8a-/J
=5	=5
*-	*-

L’immagine rappresenta le prima quattro battute del “Praeambulum BWV 390” di Johann Sebastian Bach e la loro corrispondente traduzione in **kern. Si noti che la partitura si compone di due pentagrammi, indicati in **kern come “*staff1” (corrispondente al pentagramma per la mano destra) e “*staff2” (che corrisponde al pentagramma per la mano sinistra). La parola chiave “*staff” è facoltativa: non tutte le partiture, infatti, contengono questa indicazione. Ai fini della traduzione da *kern a IEEE 1599 si è scelto di non tenere conto della sua eventuale presenza.

CAPITOLO II

IEEE 1599

Introduzione a IEEE 1599

Dopo aver brevemente descritto le principali caratteristiche di **kern, il formato di partenza da cui il nostro software dovrà estrarre le informazioni da tradurre in IEEE 1599, spendiamo qualche parola su quest'ultimo formato.

IEEE 1599[7] è un formato XML[8] sviluppato presso il Laboratorio di Informatica Musicale (LIM)[9] del Dipartimento di Informatica e Comunicazione dell'Università degli Studi di Milano dal professor Goffredo Haus, coadiuvato da Luca Andrea Ludovico, Adriano Baratè, Davide A. Mauro, Maurizio Longari, Alberto Pinto e altri. IEEE 1599 è stato creato con lo scopo di fornire una descrizione completa dell'informazione musicale. Dal settembre 2008 IEEE 1599 è diventato uno standard internazionale[9][10]. È anche conosciuto come "MX". Pertanto, nelle pagine seguenti, verranno utilizzati indistintamente i termini "IEEE 1599" e "MX".
Tutti siamo a conoscenza del fatto che l'informazione musicale sia un qualcosa di molto complesso. Per questo motivo non è facile riuscire a descriverla in tutta la sua completezza. I creatori di MX hanno pensato che per una descrizione minuziosa di tutta l'informazione musicale contenuta in un brano fosse necessario individuare un certo numero di layers, ognuno dei quali rappresenta un grado di astrazione diverso di tale informazione.
MX, per raggiungere lo scopo di descrivere completamente l'informazione musicale, si basa su sei livelli (o meglio layers): general, music logic, structural, notational, performance e audio.

Lo scheletro di un file MX è il seguente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mx SYSTEM "http://standards.ieee.org/downloads/1599/1599-2008/ieee1599.dtd">
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>…</general>
  <logic>
      <spine>…</spine>
      <los>…</los>
      <layout>…</layout>
  </logic>
  <structural>…</structural>
  <notational>…</notational>
  <performance>…</performance>
  <audio>…</audio>
</ieee1599>
```

Le prime due righe rappresentano l'header del file MX: la prima definisce la versione XML adottata e la codifica dei caratteri Unicode[11], mentre la seconda contiene il riferimento al DTD[8][12] ufficiale di MX che si trova sul sito standards.ieee.org. Attualmente esistono innumerevoli formati per la rappresentazione della musica: formati audio come .mp3 e .wav; formati immagine come .tiff, jpeg o png che ci consentono di rappresentare notazione; formati per la codifica della performance (il più importante è sicuramente il MIDI[1]); formati proprietari, come Finale, per la scrittura degli spartiti.
A tutto ciò, però, manca una struttura che permetta di rappresentare la relazione spazio-temporale, fondamentale elemento intrinseco della musica. MX individua questa struttura nel proprio sublayer spine che, come vedremo tra breve, rappresenta proprio il cuore di questo formato. Esso non è altro che una mappatura bidimensionale tra i domini dello spazio e del tempo, grazie alla quale è possibile porre in relazione tra loro diversi formati file ed ottenere così una descrizione completa dell'informazione musicale.

I Layers

```
<ieee1599>
        <general>…</general>
        <logic>
                <spine>…</spine>
                <los>…</los>
                <layout>…</layout>
        </logic>
        <structural>…</structural>
        <notational>…</notational>
        <performance>…</performance>
        <audio>…</audio>
</ieee1599>
```

In questo paragrafo descriveremo brevemente i layers[7][13] general, structural, notational, performance e audio. Il layer logic che, come precedentemente detto, è il più importante di MX, verrà trattato approfonditamente nel prossimo paragrafo. Prima di procedere alla descrizione dei layers, però, è opportuno far notare che l'elemento radice <ieee1599> è fondamentale per la well-formedness del documento XML.

Layer general
È il primo dei layer di MX ed è obbligatorio. Prende in considerazione il brano nella sua interezza e contiene le informazioni generali sul brano in questione, elencate nel sublayer description. Esso è l'unico sublayer obbligatorio di general e le informazioni che vengono annotate al suo interno, sotto forma di elementi, riguardano il titolo del brano, l'autore (o gli autori), il tipo di autore (compositore, strumentista, cantante…), il numero dell'opera ed il titolo originale (o completo) del brano, il genere ed altri dati di una certa rilevanza ai fini della descrizione e della contestualizzazione del brano preso in esame.

Di seguito le element type definition di general e description contenute nel DTD di MX[14]:

```
<!ELEMENT general (description, casting?, related_files?, analog_media?, notes?, rights?)>
```

```
<!ELEMENT description (work_title?, work_number?, movement_title, movement_number?, genre?,
  author+)>
```

Qui sotto, invece, forniamo un esempio di layer general:

```
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0" >
      <general>
            <description>
                  <main_title>Don't Say a Word</main_title>
                  <author type="performers">Sonata Arctica</author>
                  <author type="writer">Tony Kakko</author>
                  < number>5</number>
                  <work_title>Reckoning Night</work_title>
                  <genre>speed power melodic metal</genre>
            </description>
      </general>
      …
</ieee1599>
```

Layer structural
Questo layer si occupa di descrivere le relazioni esistenti all'interno del brano musicale esaminato. In poche parole nello structural troveremo quelle informazioni relative a temi musicali, soggetti, sequenze di note o battute che si ripetono, oppure altre strutture caratteristiche o di rilevante interesse. Solitamente il contenuto dello structural deriva da un lavoro di segmentazione del brano che consenta l'analisi attraverso reti di Petri[9].

Layer notational
Come il precedente layer structural, anche il layer notational è facoltativo. Esso si riferisce alle istanze "visive" di un brano musicale.
Per la scrittura e lettura della musica esistono due diverse modalità: quella notazionale e quella grafica. La prima modalità utilizza file in formato binario come NIFF o Enigma e rappresenta informazione simbolica. La seconda modalità, invece, si serve di file immagine che rappresentano la partitura: i formati più utilizzati sono JPEG, TIFF o PDF, che sono tutti e tre formati binari).
L'informazione di questo layer fa riferimento alla parte "spaziale" dello spine ed in questo modo consente la corretta localizzazione delle istanze ivi riportate.
Sebbene le istanze grafiche e notazionali siano riportate in elementi diversi del file MX, esse hanno gli stessi attributi e gli stessi elementi figli, come possiamo vedere nella parte del DTD di MX che riporta le element type definition e le attribute list declaration relative ai "componenti" del layer notational:

```
<!ELEMENT notational (notation_instance | graphic_instance)+>
<!ELEMENT notation_instance (desc?, part_ref+, rights)>
<!ATTLIST notation_instance
   file_name CDATA #REQUIRED
   format CDATA #REQUIRED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
>
<!ELEMENT graphic_instance (desc?, part_ref+, rights)>
<!ATTLIST graphic_instance
   file_name CDATA #REQUIRED
   format CDATA #REQUIRED
   spine_start_ref IDREF #REQUIRED
   spine_end_ref IDREF #REQUIRED
>
```

La presenza degli attribute spine_start_ref e spine_end_ref è dovuta al fatto che ogni file di notazione potrebbe fare riferimento ad una singola porzione dello spine e non all'intera partitura. Anche l'elemento part_ref, figlio di entrambe le istanze, è deputato a questo scopo: esso, infatti, permette di lineare all'MX singole parti di strumenti e contiene gli attributi spine_start_ref e spine_end_ref:

```
<!ELEMENT part_ref EMPTY>
<!ATTLIST part_ref
   part_id IDREF #REQUIRED
   voice IDREF #IMPLIED
   spine_start_ref IDREF #IMPLIED
   spine_end_ref IDREF #IMPLIED
>
```

Layer performance
Altro layer facoltativo che supporta formati file che codificano i parametri delle note da eseguire (come avviene nel layer notational) ed i parametri dei suoni da creare (come accade nel layer audio che vedremo successivamente), ma sempre al fine di una produzione musicale sintetica. Il layer performance supporta diversi formati come MIDI (il più importante), Csound e MPEG4.
Unica "pecca" di questo layer è il fatto che non consenta il riferimento a singoli eventi, ma solo a blocchi di essi: un' istanza ivi contenuta può riferirsi all'intero brano, ad un solo segmento oppure ad un singolo strumento e non è possibile avere una granularità più fine di questa.

```
<!ELEMENT performance (performance_instance+)>
<!-- The performance Layer is composed by zero or more representation in a subsymbolic form -->
<!ELEMENT performance_instance (desc?, (MIDI | CSOUND | MPEG4), rights)>
<!ATTLIST performance_instance
  file_name CDATA #REQUIRED
  spine_start_ref IDREF #REQUIRED
  spine_end_ref IDREF #REQUIRED
>
```

In dettaglio la parte del DTD di MX riguardante il formato MIDI:

```
<!ELEMENT MIDI (MIDI_part_ref+)>
<!ATTLIST MIDI
  format (0 | 1 | 2) #REQUIRED
>
<!ELEMENT MIDI_part_ref EMPTY>
<!-- track and channel attributes are for the identification of parts within the MIDI file,
others attributes are similar to that of part_ref ELEMENT -->
<!ATTLIST MIDI_part_ref
  part_id IDREF #REQUIRED
  voice IDREF #IMPLIED
  spine_start_ref IDREF #IMPLIED
  spine_end_ref IDREF #IMPLIED
  track CDATA #REQUIRED
  channel CDATA #REQUIRED
>
```

Layer audio
Anche il layer audio è facoltativo, come i precedenti structural, notational e performance. Esso descrive le proprietà del materiale musicale codificato sia in formati audio compressi loseless e lossy sia in formati non compressi: PCM/WAV, AIFF, MuLaw, MPEG, DOLBY, ADPCM e SHN.
Per poter collegare un qualsiasi file audio allo spine è necessario estrarre le caratteristiche degli eventi musicali relative alla loro localizzazione temporale, ma questo non dipende né dal tipo di codifica adottata né dall'eventuale compressione.

Il cuore di MX: il Layer Logic

In questo paragrafo ci occuperemo del layer logic e dei suoi tre sottolivelli: spine, los e layout.

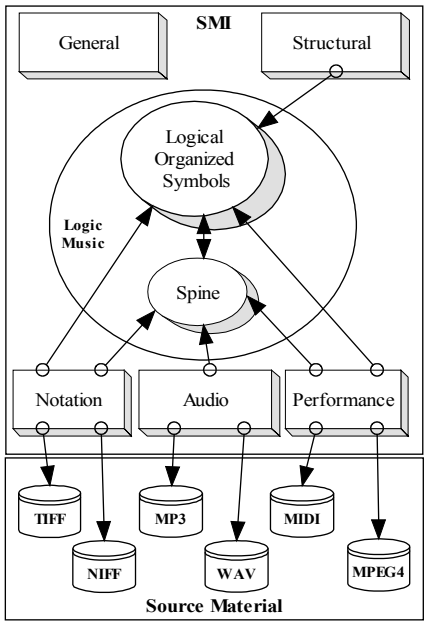
Il logic, come il general, è un layer obbligatorio. Come già detto, è il livello più importante dell'MX poiché al suo interno contiene i due sublayer fondamentali di tale formato: spine e los. Oltre ad essi, troviamo anche un terzo sublayer che, a differenza dei primi due, è facoltativo: il layout.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "mx.dtd">
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>…</general>
  <logic>
        <spine>…</spine>
        <los>…</los>
        <layout>…</layout>
  </logic>
  …
</ieee1599>
```

Passiamo ora ad esaminare nel dettaglio i sublayers spine e los.

Il sublayer spine

Lo spine rappresenta la struttura logica che implementa l'integrazione dei layers performance, notational e audio con il logic. Scopo dello spine è la costruzione di una struttura a cui facciano riferimento tutti gli strati che descrivono l'informazione musicale. Come già accennato precedentemente, grazie allo spine è possibile mettere in relazione tra loro i diversi formati in cui troviamo l'informazione musicale. Più eventi distinti (note sulla partitura o istanze di una registrazione audio), possono fare riferimento allo stesso evento logico marcato nello spine. In poche parole lo spine rappresenta il cuore dell'MX, come si evince dall'immagine qui sotto.



Nell'immagine a sinistra è rappresentata la struttura di un file MX. In essa appare evidentissima la centralità dello spine rispetto a tutti gli altri livelli che compongono il documento MX. Tutto (eccetto il layer general) fa capo allo spine. I layers notational, performance e audio, oltre ad avere riferimenti nello spine, ne hanno anche nel los, l'altro sublayer fondamentale di MX. Il layer structural non è collegato direttamente allo spine, ma fa comunque riferimento ad esso tramite il los, al quale è direttamente connesso.

Nello spine vengono etichettati gli eventi significativi che compaiono sulla partitura. Essi consistono in tutti i segni di cui è necessario tenere traccia nell'MX, che solitamente corrispondono alle note ed alle pause, gli elementi base del linguaggio musicale. Tuttavia è possibile operare altre scelte ed inserire nello spine anche elementi diversi, come le segnature e le armature di chiave, le indicazioni di tempo e le agogiche.
In pratica tutto ciò che si considera importante ai fini della descrizione dell'informazione musicale tramite MX dev'essere incluso nello spine.

```
<ieee1599>
  <general>
    …
  </general>
  <logic>
    <spine>
      <event id="evento0" timing="NULL" hpos="NULL"/>
      <event id="evento1" timing="0" hpos="6"/>
      <event id="evento2" timing="1000" hpos="0"/>
      <event id="evento3" timing="0" hpos="0"/>
      <event id="evento4" timing="2000" hpos="6"/>
      …
    </spine>
    …
  </logic>
  …
</ieee1599>
```

Lo spine definisce un ordinamento stretto tra eventi, perciò gli eventi possono essere elencati anche i maniera disordinata (es. la nota 34 di una voce può comparire in elenco prima della prima nota della stessa voce) dato che lo spine ricrea comunque l'ordinamento. Ovviamente non è consigliabile una descrizione disordinata degli eventi poiché genererebbe molta confusione nel malcapitato che deve andare a leggersi il file MX scritto in questa maniera deprecabile.
Come si evince dall'immagine precedente, l'elemento spine contiene al suo interno degli elementi figli: gli "event". Event è un elemento vuoto a cardinalità multipla e presenta tre attributi: "id", "timing" e "hpos".

Il primo attributo è "id" che assegna all'evento un identificativo univoco: durante il parsing gli eventuali "id" duplicati vengono riconosciuti come errore e perciò il file MX non risulta valido. Il nome dell'attributo "id" può essere scelto in maniera arbitraria e può contenere sia lettere sia numeri sia caratteri come "_" oppure "-": possiamo chiamare i nostri events "Duff", "Leo" e "Kelly" oppure "evento1", "evento2" ed "evento3" dato che il parser MX accetta sia gli uni sia gli altri. È comunque sempre consigliabile denominare gli eventi elencati nello spine operando scelte che portino alla massima comprensibilità, ovvero utilizzando una numerazione progressiva e, ove necessario, adottare anche un sistema di nomi che renda possibile distinguere eventi appartenenti a voci diverse.

L'attributo "timing" costituisce una temporizzazione virtuale in VTU (virtual timing units). I VTU non hanno un'unità di misura prefissata: sta all'utente scegliere volta per volta quale significato attribuire ai VTU. Non esistono regole specifiche che indichino caso per caso quali siano le scelte migliori da operare, tuttavia, sul Manuale di MX di Luca A. Ludovico, vengono dati alcuni consigli per la scelta dei VTU.
È consigliabile utilizzare valori grandi, divisibili per molti divisori e potenze di due (se questi valori sono molto grandi). Multipli di due e potenze di due ci consentono di rappresentare minime, semiminime, crome etc, mentre valori base divisibili per 3, 5 e 7 permettono di rappresentare con numeri interi gruppi di note come terzine, quintine e settimine. La scelta è comunque legata anche al tipo di brano con cui abbiamo a che

fare: se, ad esempio, ci troviamo di fronte ad un a sonata di Bach contenente solamente note da 1/4 1/8 e 1/16, possiamo adottare i seguenti valori per i VTU:
semicroma = 1;
croma = 2;
semiminima = 4;

L'attributo "hpos", invece, tiene conto della spazializzazione orizzontale virtuale, ovvero della distanza tra un elemento della partitura ed il successivo. La partitura va pensata come nella rappresentazione Score view di Finale, tutta di seguito e senza ritorni a capo. Sulla scelta dell'unità di misura per "hpos" valgono le stesse considerazioni fatte per i VTU dell'attributo timing. Risulta comodo adottare una spaziatura in base al tempo: valori doppi occupano il doppio dello spazio. In questo caso è possibile copiare i valori della colonna timing nella colonna "hpos", anche se il significato della temporizzazione risulta diverso da quello della spazializzazione orizzontale.
Oltre ai valori numerici interi positivi o nulli, sia "timing" sia "hpos" supportano il valore speciale "NULL", utilizzabile quando non ha senso parlare di spazializzazione o temporizzazione. Un caso in cui l'attributo "timing" e l'attributo "hpos" vanno impostati a "null" si ha quando etichettiamo nello spine l'armatura di chiave, il tempo o altri elementi con temporizzazione non determinabile o comunque priva di significato.

Il sublayer los
Il los è il secondo sublayer contenuto nel layer logic di un documento MX. La sigla "los" è l'acronimo di "Logical Organized Symbols". Qui troviamo le informazioni simboliche di partitura: note, pause, segni di articolazione, dinamiche, etc.

Se vogliamo inserire una partitura nel sublayer los dobbiamo stabilire una gerarchia delle informazioni in essa contenute. È facile individuare quali siano il vertice e la base di questa gerarchia: il primo è rappresentato dall'intera partitura, mentre la seconda corrisponde alle singole note ed alle singole pause: gli atomi del linguaggio musicale.

Ma come possiamo strutturare l'informazione che sta tra il vertice e la base?
MX ci ha dato questa risposta:

1. una partitura è costituita da più accollature;
2. un'accollatura è formata da più pentagrammi;
3. un pentagramma contiene una o più parti;
4. una parte può essere suddivisa in più voci;
5. una voce è costituita da un certo numero di battute;
6. una battuta contiene accordi e pause;
7. un accordo contiene una o più note.

Questa struttura, tuttavia, non è in grado di supportare tutti i casi che si presentano in letteratura: esistono, infatti, brani in cui si ha la migrazione di una voce da un pentagramma ad un altro, oppure parti che si compongono di più pentagrammi (musica per tastiera o per arpa, ad esempio). Tutti questi casi particolari andranno di volta in volta gestiti attraverso strutture create ad hoc.

Le battute sono descritte parte per parte: prima tutte le misure della prima voce della prima parte, poi quelle delle voci successive della stessa parte. Esaurita la descrizione di una parte si passa alla successiva e si procede allo stesso modo.

Nella descrizione gerarchica della partitura in MX, con il termine accordo si intende esclusivamente la sovrapposizione di note all'interno di una parte e di una voce specifica.
La singola nota, ovvero quella che, non facendo parte di un accordo, non si sovrappone verticalmente ad alcun suono appartenente alla voce stessa, viene codificare come un accordo.

Seguendo la gerarchia sopra descritta, lo scheletro del sublayer los è simile al seguente:

```
<los>
  <staff_list>
    <staff id="staff_1">  … </staff>
    <staff id="staff_2">  … </staff>
  </staff_list>
  <part id="part_1" dfstaff_ref="staff_1">
    <voice_list>
      <voice_item id="voce_1"/>
    </voice_list>
    <measure number="1"> … </measure>
    <measure number="2"> … </measure>
    …
  </part>
  <part id="part_2_destra" dfstaff_ref="staff_2">
    <voice_list>
      <voice_item id="voce_dx_sup"/>
      <voice_item id="voce_dx_inf"/>
    </voice_list>
    <measure number="1"> … </measure>
    <measure number="2"> … </measure>
    …
  </part>
  <part id="part_2_sinistra" dfstaff_ref="staff_2">
    <voice_list>
      <voice_item id="voce_sx_sup"/>
      <voice_item id="voce_sx_inf"/>
    </voice_list>
    <measure number="1"> … </measure>
    <measure number="2"> … </measure>
    …
  </part>
</los>
```

Il tag “staff_list” indica l’accollatura, ovvero l’elenco dei pentagrammi (“staff”); “part”, ovviamente, indica la parte, mentre “voice_list” contiene l’elenco delle voci in cui è suddivisa la parte; “voice_item”, invece, è la singola voce. Da notare che il livello di indentazione delle parti è uguale a quello dell’accollatura: pur essendo associate ai pentagrammi e quindi trovandosi ad un livello gerarchico più basso, non vengono associate ad essi tramite indentazione come elementi figli di “staff”. La corrispondenza tra pentagrammi e parti, viene comunque riconosciuta immediatamente tramite l’utilizzo dell’attributo “dstaff_ref” che ha come valore l’identificatore definito per una delle parti.

```
<!ELEMENT staff_list (staff+)>
<!ELEMENT staff (clef | key_signature | time_signature | barline | staff_hiding)*>
<!ATTLIST staff
  id ID #IMPLIED
  ossia (yes | no) "no"
  line_number CDATA "5">
```

Passiamo ora ad esaminare l'elemento "clef", figlio di "staff": si tratta di un elemento vuoto, il cui contenuto informativo è legato ai suoi attributi, come si evince dalle righe del DTD riportate qui sotto:

```
<!ELEMENT clef EMPTY>
<!ATTLIST clef
  type (G | F | C | percussion | doubleG | tabguitar) #REQUIRED
  staff_step CDATA #REQUIRED
  octave_num (0 | 8 | -8 | 15 | -15) "0"
  %spine_attributes; >
```

L'attributo "type", richiesto, indica il tipo di chiave: chiave di Sol, di Fa, di Do e chiavi per percussioni ed intavolature (TAB). La posizione della chiave sul pentagramma è indicata dall'attributo "staff_step", anch'esso richiesto: al momento segue lo standard NIFF per cui, partendo dal basso del pentagramma, il numero "0" rappresenta la prima riga, "1" il primo spazio, "2" la seconda riga e così via…

Nella seguente tabella sono riportate le chiavi più comuni e la loro rappresentazione in MX:

Canto		<clef type="G" staff_step="2" event_ref="clef_1"/>
Soprano		<clef type="C" staff_step="0" event_ref="clef_1"/>
Mezzosoprano		<clef type="C" staff_step="2" event_ref="clef_1"/>
Contralto		<clef type="C" staff_step="4" event_ref="clef_1"/>
Tenore		<clef type="C" staff_step="6" event_ref="clef_1"/>
Baritono		<clef type="F" staff_step="4" event_ref="clef_1"/>
Basso		<clef type="F" staff_step="6" event_ref="clef_1"/>
Intavolatura		<clef type="tabguitar" staff_step="?" event_ref="clef_1"/>
Percussioni		<clef type="percussion" staff_step="0" event_ref="clef_1"/>

Altri due attributi di "clef" sono "octave_num" e "event-ref". Il primo, che non dev'essere necessariamente indicato in MX, denota i numeri posti solitamente sopra o sotto la chiave per definire l'innalzamento o l'abbassamento di una o più ottave delle note del rigo: i valori ammessi per questo attributo sono "0" (valore di default); "+8" e "-8" (che indicano rispettivamente l'innalzamento e l'abbassamento di un'ottava) e "+15" e "-15" (due ottave sopra e due ottave sotto). Il secondo attributo, invece, costituisce un riferimento allo spine ed è perciò indispensabile.

Di seguito proponiamo la definizione dell'elemento "time_signature" nel DTD di MX:

```
<!ELEMENT time_signature EMPTY>
<!ATTLIST time_signature
  num CDATA #REQUIRED
  den CDATA #REQUIRED
  char_type (yes | no) "no"
  cut (yes | no) "no"
  single_number (yes | no) "no"
  v_placement (above | center | below) "center"
  size (normal | large | small) "normal"
  visible (yes | no) "yes"
  vtu_amount CDATA #IMPLIED
  %spine_attributes; >
```

Per questo elemento non c'è molto da dire, a parte alcuni brevissimi cenni sull'attributo "vtu_amount". Si tratta di un attributo strettamente collegato allo spine, dato che indica la somma dei VTU che compongono una battuta così come l'utente li ha scelti nello spine: se il tempo è di 4/4 e abbiamo assegnato il valore 1024 ai quarti, vtu_amount corrisponderà a 4096 (1024x4).

L'ultimo elemento di cui ci occupiamo in questa breve trattazione dell'MX è l'elemento "measure", figlio di "part". Per questo motivo la descrizione della partitura avviene parte per parte: all'interno della misura vengono descritte tutte le voci che compongono la parte.

```
<los>
  <staff_list> … </staff_list>
  <part id="part_organo" dfstaff_ref="staff_organo">
    <voice_list>
      <voice_item id="mano_dx"/>
      <voice_item id="mano_sx"/>
      <voice_item id="pedale"/>
    </voice_list>
    <measure number="1">
      <voice ref="mano_dx"> … </voice>
      <voice ref=”mano_sx”> … </voice>
      <voice ref=”pedale”> … </voice>
    </measure>
    <measure number="2">
      <voice ref="mano_dx"> … </voice>
      <voice ref=”mano_sx”> … </voice>
      <voice ref=”pedale”> … </voice>
    </measure>
    …
  </part>
</los>
```

Measure ha un attributo fondamentale: "number", che indica il numero di battuta. È inutile dire che questo attributo sia obbligatoriamente richiesto.
Un altro attributo di measure è "id": quest'ultimo, però, è facoltativo e può essere utile nel caso ci fossero riferimenti alla battuta da parte di altri elementi dell'MX.
Ricordiamo che nell'elemento "measure" non compaiono indicazioni sulla segnatura di tempo, né sul tipo di chiave né sulla sua armatura: le battute, infatti, si mappano nei pentagrammi tramite l'attributo "staff_ref" che caratterizza ogni elemento "notehead".

Di seguito riportiamo un esempio di come vengono codificati gli accordi e le note all'interno di una measure. Dato che la sintassi è molto chiara, non sono necessarie molte spiegazioni. Le uniche annotazioni da fare riguardano il fatto che la singola nota

viene codificata al pari di un accordo e l'obbligatorietà degli attributi "number" per measure; "voice_item ref" per voice; "event_ref" per chord; "num" e "den" per duration; "octave" e "step" per pitch. L'attributo "actual_accidental" di pitch e l'elemento printed_accidentals sono invece facoltativi: non tutte le note, infatti, hanno alterazioni. I valori possibili dell'attributo "actual_accidental" sono "natural", "sharp" (diesis) e "flat" (bemolle). L'eventuale elemento printed_accidentals deve contenere uno dei seguenti elementi vuoti: natural, sharp o flat.

```
<measure number="4">
 <voice voice_item_ref="rh_voice">
  <chord event_ref="rh_01">
    <duration num="1" den="4" />
    <notehead>
     <pitch octave="6" step="E" actual_accidental="flat" />
    </notehead>
    <notehead>
     <pitch octave="5" step="A" actual_accidental="natural" />
      <printed_accidentals>
        <natural/>
      </printed_accidentals>
    </notehead>
    <notehead>
     <pitch octave="5" step="F" actual_accidental="sharp" />
     <printed_accidentals>
       <sharp/>
     </printed_accidentals>
    </notehead>
  </chord>
```

CAPITOLO III

COME TRADURRE DA **KERN A IEEE 1599

In questo capitolo ci occuperemo di individuare quali soluzioni possiamo adottare per effettuare la conversione dal formato **kern al formato IEEE 1599.

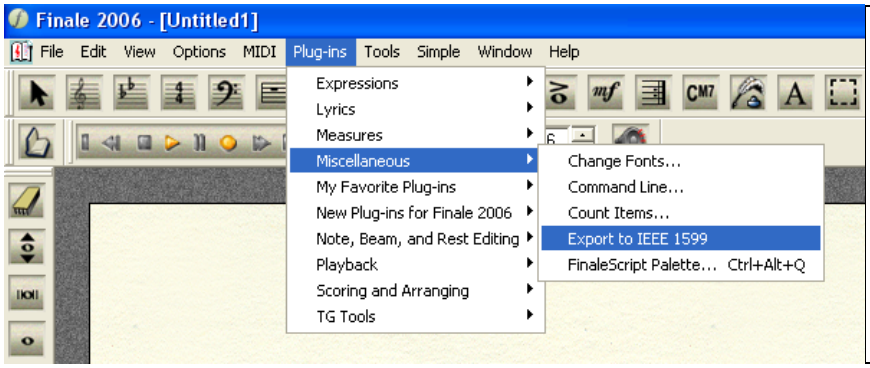
Esistono diversi metodi che ci consentono la traduzione da **kern a MX, tuttavia nessuno di quelli conosciuti finora è in grado di effettuare la conversione diretta da un formato all'altro: pertanto si rende necessario passare attraverso uno o più formati "intermedi".

Il primo metodo che prendiamo in considerazione permette di passare da **kern a IEEE 1599 attraverso l'utilizzo del software per la notazione Finale e del plug-in per la conversione in IEEE 1599 sviluppato presso il LIM. Elenchiamo di seguito i passi da effettuare per la conversione:

1. Selezionare il file **kern da tradurre tra quelli presenti nel database del sito http://kern.ccarh.org/;
2. Copiare l'url del file selezionato;
3. Dalla homepage del sito http://kern.ccarh.org/, aprire la pagina Online Humdrum Editor[6][15] ed incollare l'url del file selezionato nella barra "input url";
4. Selezionare il pulsante "notation" e scegliere uno dei formati immagine in cui è possibile ottenere la partitura del file **kern. I due formati possibili sono "png" e "gif".
5. Fare click sul pulsante "generate" per aprire l'immagine della partitura di origine;
6. Salvare l'immagine ottenuta in formato "tiff" ed importarla in Finale, preferibilmente la versione 2006 od una più recente;
7. Utilizzare il plug-in "Finale Conversion Plug-in" realizzato presso il LIM per convertire il file in IEEE 1599. Se non si disponesse del suddetto plug-in è possibile scaricarlo dal seguente link: http://www.mx.dico.unimi.it/download.php.

Con sette semplici passaggi siamo in grado di tradurre da **kern a IEEE 1599. Tuttavia basta non avere la possibilità o non essere in grado di svolgere uno di essi e la traduzione non è possibile.

Immagine a sinistra: una parte della schermata principale del sito KernScores (http://kern.ccarh.org/). Sotto la barra di ricerca, si trova il link alla pagina Online Humdrum Editor (è il primo sulla destra. Nella parte centrale, invece, trova spazio l'elenco dei compositori le cui opere sono presenti nel database **kern.
Immagine a destra: la schermata principale dell'Online Humdrum Editor. Nel riquadro input (a sinistra) va inserita la partitura da convertire o analizzare, tramite la barra "input url" che si trova sotto al riquadro. Al di sotto dei riquadri sono elencate alcune delle operazioni che è possibile svolgere: visualizzare la partitura in formato pdf o come immagine; scaricare la traccia midi, effettuare trasposizioni; estrarre uno o più spines dal file **kern... Nel riquadro a destra viene generato l'output, dopo che viene dato l'avvio all'estrazione/analisi tramite il comando "generate" che si trova sotto al suddetto riquadro.
Attenzione: le operazioni Notation e MIDI non mettono il loro outuput nel riquadro: nel primo caso lo spartito viene direttamente visualizzato a schermo intero, mentre nel secondo caso la traccia midi viene automaticamente scaricata ed è subito disponibile per l'ascolto.



L'immagine a sinistra mostra, sulla schermata principale di Finale 2006, il percorso per trovare il plug-in MXPlugin per l'esportazione in IEEE 1599.
L'esecuzione di tale plug-in creerà un documento MX contenente spine e los, completi di tutti i riferimenti ai simboli in partitura (indicazioni di tempo, chiave, note e pause).

Un secondo metodo di traduzione, fortemente sconsigliato, consiste nella traduzione manuale da **kern a IEEE 1599. Questo metodo di traduzione è praticabile se abbiamo a che fare con brani di poche battute e con al massimo due parti, ma se ci troviamo di fronte a pezzi molto lunghi e complessi, la conversione diventa lunga e praticamente impossibile.

Per questo motivo si ha la necessità di creare uno strumento software che consenta un'immediata traduzione da **kern a IEEE 1599 e che non comporti molti passi per l'utente.

Nel seguente capitolo verranno presentati alcuni prototipi software, di complessità crescente, deputati allo scopo di convertire da **kern a IEEE 1599.

CAPITOLO IV

PROTOTIPI SOFTWARE PER LA CONVERSIONE DA **KERN A IEEE 1599

In questi mesi di lavoro ho tentato di sviluppare uno strumento software in grado di ricevere in input un file codificato secondo la sintassi **kern e di convertirlo in un documento XML conforme allo standard IEEE 1599.

Si è trattato di un lavoro lungo e complesso che ha portato alla realizzazione di alcuni prototipi software di complessità via via crescente, in grado di creare i sublayer spine e los del layer logic di un documento MX ricevendo in input un file **kern, o comunque un file di testo in cui è codificata una partitura **kern. Quindi, oltre ai file .krn, il software può leggere anche files in formato .doc o .txt, ad esempio.

La scrittura del codice per la traduzione dello spine è stata un'operazione relativamente semplice e l'unica lieve criticità è stata riscontrata nella ricerca di un metodo per l'assegnazione dei vtu alle note puntate, poi risolta attraverso l'uso dei cicli for e delle istruzioni if che vedremo più avanti nella trattazione del primo prototipo software.
Altro problema di più complessa soluzione è stato riscontrato nella traduzione del los, in particolare nella scrittura del codice riguardante la creazione degli elementi corrispondenti alle misure e nell'incremento del numero di misura al comparire del simbolo che in **kern indica le linee di battuta. Anche questa criticità è stata risolta, come vedremo quando si tratterà di parlare del secondo prototipo software.

L'unica problematica non risolta, invece, riguarda la conversione automatica di partiture **kern complesse, contenenti due o più spines. Se vogliamo tradurre partiture complesse utilizzando i software realizzati per questo elaborato, infatti, dobbiamo scomporre la partitura **kern nei suoi spines e tradurli uno per uno servendoci di una variante che, invece di creare ex-novo il file XML su cui andare a scrivere lo spine ed il los, consentirà all'utente di caricare un file XML contenente già la traduzione della prima parte. Il codice dell'eventuale variante, perciò, differirà leggermente dal codice che vedremo tra breve per il Software 1 ed il Software 2.
In particolare, al posto delle istruzioni per la creazione di un nuovo file XML, ci sarebbero delle righe di codice simili alle seguenti:

```
// caricamento del file xml
Console.WriteLine("DIGITARE IL NOME DEL FILE XML DA CARICARE:");
string strFilename = Console.ReadLine();
XmlDocument xmlDoc = new XmlDocument();
xmlDoc.Load(strFilename);

XmlElement elmRoot = xmlDoc.DocumentElement;
```

grazie a cui l'utente potrà aprire il file desiderato in cui introdurre gli eventi relativi alle parti successive alla prima.

Inoltre l'identificatore della parte, "kernCount", invece che essere impostato a 1, consisterebbe in una variabile definita di volta in volta dall'utente in base al numero della parte che deve tradurre:

```
Console.WriteLine("DIGITARE IL NUMERO DELLA PARTE: ");
string kernCount = Console.ReadLine();
```

Se la parte da tradurre è la seconda verrà inserito il numero "2", se è la terza il "3" e così via.

Altre differenze consisterebbero nelle istruzioni per la creazione e l'annidamento degli elementi che, invece di andare a compilare un file nuovo, devono essere introdotti correttamente all'interno della struttura di un file XML già salvato.

Un esempio: `elmRoot.LastChild.LastChild.AppendChild(N_event);` in una variante al Software 1 sarebbe l'istruzione che consente di annidare l'elemento event di seguito agli altri events già contenuti nello spine.

Di seguito viene proposto un esempio di file **kern. Il brano in questione è la Sonata I di Johann Sebastian Bach. In particolare, il file **kern contiene la parte per pianoforte suonata dalla mano sinistra del pianista.

File Sonata_I.krn

```
**kern
*clefF4
*M4/4
=1
4C
4G
4D
4A
=2
4F
4G
4C
4c
=3
4A
4F
4B-
4G
=4
4A
4D
4A
4AA
=5
4D
4G
4C
4F
*-
```

Lo stesso file può anche avere una sintassi leggermente diversa, con le battute indicate semplicemente dal carattere "=", senza numero. La sintassi **kern, infatti, in questo come in altri casi, lascia molta libertà.

File Sonata_1(variante).krn

```
**kern
*clefF4
*M4/4
=
4C
4G
4D
4A
=
4F
4G
4C
```

```
4c
=
4A
4F
4B-
4G
=
4A
4D
4A
4AA
=
4D
4G
4C
4F
```

Dopo aver fatto un breve accenno alle criticità riscontrate durante la realizzazione dei software ed aver proposto questi due esempi di partiture **kern, è arrivato il momento di presentare i prototipi software da me realizzati.

Si tratta di semplici applicazioni console in linguaggio C#[16][17] realizzati con Microsoft Visual Studio 2008[17] con le seguenti caratteristiche:

- Software 1: consente solo la traduzione dello spine di una partitura costituita da un'unica parte con un unico pentagramma;
- Software 2: traduce spine e los di una partitura composta da un unico pentagramma.

Di seguito verrà proposta l'intestazione comune a tutti i software:

```
// Titolo:                  Prototipi Software per la Conversione da Kern a IEEE 1599
// Tipo di pubblicazione:   Elaborato finale (Laurea triennale)
// Corso di laurea:         Scienze e Tecnologie della Comunicazione Musicale
// Ateneo:                  Università degli Studi di Milano
// Laureando/Programmatore: Giorgio Rapetti <giorgio.rapetti@studenti.unimi.it>
// Relatore:                Luca Andrea Ludovico
// Correlatore:             Adriano Baratè
// Filename:                PrototipiSoftwareConversioneKern_IEEE1599
// Sintassi:                C#
// Descrizione:             Software per la conversione da Kern a IEEE 1599

using System;
using System.IO;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Linq;
using System.Xml;
using System.Xml.Serialization;
using System.Net;
```

Da notare la presenza fondamentale delle librerie per la gestione di XML[18] [19] tramite il linguaggio di programmazione C#: esse sono richiamate nel programma a mezzo delle direttive "using System.Xml" e "using System.Xml.Serialization", sottolineate in giallo.

Nei prossimi paragrafi, invece, analizzeremo i prototipi software, riportandone le caratteristiche principali, le parti del codice più significative ed il relativo output in formato XML.

Software 1

Il primo software proposto è molto semplice: esso, infatti, è in grado di creare un documento XML contenente il layer general e il sublayer spine del layer logic a partire da un file **kern che rappresenta una partitura composta da un'unica voce.
La prima istruzione del software consente all'utente di immettere il nome del file **kern da tradurre e di stamparne a video il contenuto:

```
//apertura del file KernFileToTranslate nel quale abbiamo copiato la
partitura kern da tradurre
Console.WriteLine("DIGITARE IL NOME DEL FILE **KERN DA TRADURRE:");
string fileKern = Console.ReadLine();
StreamReader HumdrumFile = new StreamReader(fileKern);
string sLine = "";
ArrayList arrText = new ArrayList();
while (sLine != null)
{
      sLine = HumdrumFile.ReadLine();
      if (sLine != null)
      {
            arrText.Add(sLine);
      }
}
HumdrumFile.Close();
foreach (string sOutput in arrText)
{
Console.WriteLine(sOutput);
}
```

Dopo aver aperto il file **kern che si desidera convertire in IEEE1599, il software crea il documento XML in cui scrivere il general e lo spine:

```
// creazione del file xml
XmlDocument xmlDoc = new XmlDocument();
XmlNode xmlDocNode = xmlDoc.CreateXmlDeclaration("1.0", "UTF-8", null);
xmlDoc.AppendChild(xmlDocNode);
XmlElement elmRoot = xmlDoc.CreateElement("ieee1599");
xmlDoc.AppendChild(elmRoot);
XmlAttribute creator = xmlDoc.CreateAttribute("creator");
elmRoot.SetAttribute("creator", "Laboratorio di Informatica Musicale");
XmlAttribute version = xmlDoc.CreateAttribute("version");
elmRoot.SetAttribute("version", "1.0");
```

Con le istruzioni riportate qui sopra, il software crea l'header del file XML ed il root element <ieee1599> con i relativi attributi. Le istruzioni che vedremo qui sotto, invece, permettono di creare il layer general con le informazioni riguardanti titolo ed autore del brano:

```
// creazione layer general
XmlElement N_general = xmlDoc.CreateElement("general");
elmRoot.AppendChild(N_general);
XmlElement N_description = xmlDoc.CreateElement("description");
N_general.AppendChild(N_description);
XmlElement MainTitle = xmlDoc.CreateElement("main_title");
N_description.AppendChild(MainTitle);
Console.WriteLine("Inserire titolo del brano:");
string title = Console.ReadLine(); // il titolo del brano viene inserito
dall'utente
XmlText MainTitleText = xmlDoc.CreateTextNode(title);
MainTitle.AppendChild(MainTitleText);
```

```
XmlElement N_author = xmlDoc.CreateElement("author");
Console.WriteLine("Inserire nome dell'autore del brano:");
string author = Console.ReadLine();
XmlText Author_name = xmlDoc.CreateTextNode(author); //l'utente inserisce
l'autore
N_author.AppendChild(Author_name);
XmlAttribute A_type = xmlDoc.CreateAttribute("type");
Console.WriteLine("Inserire il tipo di autore:");
string typeOfAuthor = Console.ReadLine();
N_author.SetAttribute("type", typeOfAuthor);
N_description.AppendChild(N_author);
```

L'utente può inserire il contenuto degli elementi <main_title> e <author>, ed il valore dell'attributo "type" di author. In questo modo l'utente ha la possibilità di compilare il layer general come meglio crede.

Istruzione fondamentale è quella che permette di creare il layer logic ed il suo sublayer spine:

```
// creazione layer logic con sublayer spine
XmlElement N_logic = xmlDoc.CreateElement("logic");
XmlElement N_spine = xmlDoc.CreateElement("spine");
elmRoot.AppendChild(N_logic);
N_logic.AppendChild(N_spine);
```

Di seguito, invece, riportiamo la parte di codice contenente le istruzioni che consentono di creare gli eventi dello spine relativi alle note ed alle pause:

```
// creazione di tutti gli eventi corrispondenti alle note da marcare nello
spine
int vtu = 0; // vtu per timing e hpos degli events dello spine
int contaCifre = 0; // contatore per le cifre che indicano la durata della
nota
int contaPunti = 0; // contatore per i punti di valore
int durata = 0; // durata della nota
string punti = "";
foreach (string sOutput in arrText)
{
      contaCifre = 0;
      contaPunti = 0;
      i = 0;
      durata = 0;
      while (sOutput != "*clef*" && sOutput != "*M*" && sOutput == "**kern")
      {
            if (sOutput != null)
            {
            evCount++;
            }
      }
      XmlElement N_event = xmlDoc.CreateElement("event");
      XmlAttribute id = xmlDoc.CreateAttribute("id"); // attributo id OK
      N_event.SetAttribute("id", "parte" + kernCount + idx + evCount++);
      // timing e hpos:
      // ciclo for x individuare numeri presenti nella stringa: i numeri
      indicano la durata della nota
      for (; i < sOutput.Length; i++)
      {
            if (sOutput[i] >= '0' && sOutput[i] <= '9')
                  contaCifre++;
            else
                        break;
```

```
        }
        if (sOutput != null)
        {
              durata = Convert.ToInt32(sOutput.Substring(0, contaCifre));
              vtu = 4096 / durata;
        }
        // ciclo per trovare eventuali punti di valore
        for (; i < sOutput.Length; i++)
        {
              if (sOutput[i] == '.')
                    contaPunti++;
              else
                     break;
        }
punti = sOutput.Substring(0, contaPunti);
Console.WriteLine("punti: " + punti);
// previsti casi fino ad un massimo di tre punti di valore poichè in musica
non vi sono note con più di 3 punti!!!
if (contaPunti == 1)
      vtu = vtu + vtu / 2;
if (contaPunti == 2)
      vtu = vtu + vtu/2 + vtu /4;
if (contaPunti == 3)
      vtu = vtu + vtu / 2 + vtu / 4 + vtu / 8;
XmlAttribute timing = xmlDoc.CreateAttribute("timing");
XmlAttribute hpos = xmlDoc.CreateAttribute("hpos");
N_event.SetAttribute("timing", vtu.ToString());
N_event.SetAttribute("hpos", vtu.ToString());
N_spine.AppendChild(N_event);
}
```

Il software, dopo aver contato le righe corrispondenti alle note (o pause) da tradurre negli events dello spine, conta le cifre che stanno ad indicare la durata della nota e le converte in un intero, chiamato “durata”. Questo intero è fondamentale per il calcolo dei vtu da inserire come valore degli attributi “timing” e “hpos” di event.
Il valore dei vtu corrispondente alla nota intera è 4096. I vtu delle note di durata inferiore si calcolano dividendo 4096 per la durata della nota corrispondente.

```
if (sOutput != null)
{
      durata = Convert.ToInt32(sOutput.Substring(0, contaCifre));
      vtu = 4096 / durata;
}
```

Per quanto riguarda il calcolo dei vtu delle note puntate, invece, le operazioni da svolgere sono un pochino più complesse. In musica, infatti, possiamo trovare note che presentano fino a un massimo di tre punti di valore: il punto di valore singolo aumenta la nota di metà del suo valore;il doppio punto di valore aumenta la nota di ½ + ¼ del suo valore; mentre il triplo punto di valore aumenta la durata della nota di ½ + ¼ + 1/8 del suo valore iniziale. Perciò bisogna calcolare i vtu in base al numero di punti di valore che seguono la nota: il software svolge questa operazione grazie al contatore contaPunti che viene incrementato di un’unità per ogni punto di valore che trova in ogni riga. In questo modo le operazioni per il calcolo dei vtu possono essere implementate con le seguenti istruzioni:

```
if (contaPunti == 1)
      vtu = vtu + vtu / 2;
if (contaPunti == 2)
      vtu = vtu + vtu/2 + vtu /4;
```

```
if (contaPunti == 3)
      vtu = vtu + vtu / 2 + vtu / 4 + vtu / 8;
```

Ricapitolando, quindi, i valori dei vtu per le note non puntate saranno i seguenti (tra parentesi la codifica in **kern):
4096 per la semibreve (1);
2048 per la minima (2);
1024 per la semiminima (4);
512 per la croma (8);
256 per la semicroma (16);
128 per la biscroma (32);
64 per la semibiscroma (64);
32 per la fusa (128);

Il software si chiude con l'istruzione:

```
Console.WriteLine("Inserire il nome del file nella riga sottostante");
string Name = Console.ReadLine();
xmlDoc.Save(Name);
```

che consente all'utente di inserire il nome o il percorso del file XML appena creato e di salvarlo.

Di seguito proponiamo un esempio di output generato dall'esecuzione del software. Come esempio abbiamo scelto l'introduzione della canzone "San Sebastian" del gruppo power metal finlandese Sonata Arctica.
Il file **kern di origine è riportato nell'Appendice I.

```
<?xml version="1.0" encoding="UTF-8"?>
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>
    <description>
      <main_title>San Sebastian</main_title>
      <author type="composer">Sonata Arctica</author>
    </description>
  </general>
  <logic>
    <spine>
      <event id="parte1ev_clef" timing="NULL" hpos="NULL" />
      <event id="parte1ev_time" timing="NULL" hpos="NULL" />
      <event id="parte1_evento1" timing="1536" hpos="1536" />
      <event id="parte1_evento2" timing="1536" hpos="1536" />
      <event id="parte1_evento3" timing="1024" hpos="1024" />
      <event id="parte1_evento4" timing="1024" hpos="1024" />
      <event id="parte1_evento5" timing="1024" hpos="1024" />
      <event id="parte1_evento6" timing="1024" hpos="1024" />
      <event id="parte1_evento7" timing="1024" hpos="1024" />
      <event id="parte1_evento8" timing="1024" hpos="1024" />
      <event id="parte1_evento9" timing="1024" hpos="1024" />
      <event id="parte1_evento10" timing="1024" hpos="1024" />
      <event id="parte1_evento11" timing="1024" hpos="1024" />
      <event id="parte1_evento12" timing="2048" hpos="2048" />
      <event id="parte1_evento13" timing="1024" hpos="1024" />
      <event id="parte1_evento14" timing="1024" hpos="1024" />
      <event id="parte1_evento15" timing="1024" hpos="1024" />
      <event id="parte1_evento16" timing="1024" hpos="1024" />
      <event id="parte1_evento17" timing="1024" hpos="1024" />
      <event id="parte1_evento18" timing="1024" hpos="1024" />
      <event id="parte1_evento19" timing="1024" hpos="1024" />
      <event id="parte1_evento20" timing="1024" hpos="1024" />
      <event id="parte1_evento21" timing="1024" hpos="1024" />
```

```
      <event id="parte1_evento22" timing="1024" hpos="1024" />
      <event id="parte1_evento23" timing="1024" hpos="1024" />
      <event id="parte1_evento24" timing="1024" hpos="1024" />
      <event id="parte1_evento25" timing="1024" hpos="1024" />
      <event id="parte1_evento26" timing="1024" hpos="1024" />
      <event id="parte1_evento27" timing="2048" hpos="2048" />
      <event id="parte1_evento28" timing="1024" hpos="1024" />
      <event id="parte1_evento29" timing="1024" hpos="1024" />
      <event id="parte1_evento30" timing="1536" hpos="1536" />
      <event id="parte1_evento31" timing="1536" hpos="1536" />
      <event id="parte1_evento32" timing="1024" hpos="1024" />
      <event id="parte1_evento33" timing="1024" hpos="1024" />
      <event id="parte1_evento34" timing="1024" hpos="1024" />
      <event id="parte1_evento35" timing="1024" hpos="1024" />
      <event id="parte1_evento36" timing="1024" hpos="1024" />
      <event id="parte1_evento37" timing="1024" hpos="1024" />
      <event id="parte1_evento38" timing="1024" hpos="1024" />
      <event id="parte1_evento39" timing="1024" hpos="1024" />
      <event id="parte1_evento40" timing="1024" hpos="1024" />
      <event id="parte1_evento41" timing="2048" hpos="2048" />
      <event id="parte1_evento42" timing="1024" hpos="1024" />
      <event id="parte1_evento43" timing="1024" hpos="1024" />
      <event id="parte1_evento44" timing="1024" hpos="1024" />
      <event id="parte1_evento45" timing="1024" hpos="1024" />
      <event id="parte1_evento46" timing="1536" hpos="1536" />
      <event id="parte1_evento47" timing="512" hpos="512" />
      <event id="parte1_evento48" timing="1024" hpos="1024" />
      <event id="parte1_evento49" timing="1024" hpos="1024" />
      <event id="parte1_evento50" timing="1536" hpos="1536" />
      <event id="parte1_evento51" timing="512" hpos="512" />
      <event id="parte1_evento52" timing="1024" hpos="1024" />
      <event id="parte1_evento53" timing="1024" hpos="1024" />
      <event id="parte1_evento54" timing="1536" hpos="1536" />
      <event id="parte1_evento55" timing="512" hpos="512" />
      <event id="parte1_evento56" timing="4096" hpos="4096" />
    </spine>
  </logic>
</ieee1599>
```

Software 2

Il secondo prototipo software costituisce un passo avanti rispetto al prototipo appena visto. Esso, infatti, aggiunge una nuova funzionalità rispetto al precedente: la traduzione completa anche del sublayer los. Anche questo software consente di convertire una partitura costituita da un'unica voce.
Le parti del codice relativa all'apertura del file **kern da tradurre, alla creazione del file XML ed alla compilazione del layer general e dello spine, sono identiche a quelle appena viste per il software 1, pertanto non verranno esposte nuovamente anche per questo secondo prototipo software.

La prima istruzione presentata è quella relativa alla creazione del sublayer los ed al suo annidamento come elemento figlio del layer logic:

```
elmRoot = xmlDoc.DocumentElement;
XmlElement N_los = xmlDoc.CreateElement("los");
N_logic.AppendChild(N_los);
```

Il seguente blocco di codice gestisce la creazione degli elementi <measure> e di tutti i suoi elementi figli: voice, chord, duration, notehead e pitch, con i relativi attributi. Le istruzioni riportate qui sotto si riferiscono sia all'esatto annidamento degli elementi nel file XML, sia all'esatta collocazione degli attributi, sia all'esatta traduzione delle informazioni da inserire negli attributi:

```
foreach (string sOutput in arrText)
{
	XmlElement voice = xmlDoc.CreateElement("voice");
	if (sOutput == "=")
	{
		number_measure++;
		XmlElement misura = xmlDoc.CreateElement("measure");
		N_los.LastChild.AppendChild(misura);
		XmlAttribute number = xmlDoc.CreateAttribute("number");
		misura.SetAttribute("number", number_measure.ToString());
		voice = xmlDoc.CreateElement("voice");
		misura.AppendChild(voice);
		XmlAttribute item_ref = xmlDoc.CreateAttribute("voice_item_ref");
		voice.SetAttribute("voice_item_ref", "voice" + voice_num);
		continue;
	}

	i = 0;
	contaCifre = 0;
	while (sOutput == "**kern" && sOutput != null)
	{
		if (sOutput != "**kern" && sOutput != "*M*" && sOutput != "=" &&
		sOutput != "clef*" && sOutput != "*-" )
		{
			spineEventCount++;
		}
	}
	XmlElement chord = xmlDoc.CreateElement("chord");
	N_los.LastChild.LastChild.LastChild.AppendChild(chord);
	XmlAttribute event_ref = xmlDoc.CreateAttribute("event_ref");
	chord.SetAttribute("event_ref", "parte" + kernCount + idx + spineEvent-
	Count++);
	// inserimento element duration con relativi attributi
	XmlElement duration = xmlDoc.CreateElement("duration");
	chord.AppendChild(duration);
```

```
int c_num = 0, c_den = 0;
for (; i < sOutput.Length; i++)
{
      if (sOutput[i] >= '0' && sOutput[i] <= '9')
            contaCifre++;
}
if (sOutput != null && sOutput != "*clef*" && sOutput != "*M*" && sOutput !=
"=" && sOutput != "*-" && sOutput != "**kern")
{
      durata = Convert.ToInt32(sOutput.Substring(0, contaCifre));
      c_num = 1;
      c_den = durata;
}
for (; i < sOutput.Length; i++)
{
      if (sOutput[i] == '.')
            contaPunti++;
      else
          break;
}
// previsti casi fino ad un massimo di tre punti di valore poichè in musica
non vi sono note con più di 3 punti!!!
if (contaPunti == 1)
      c_den = durata + durata / 2;
if (contaPunti == 2)
      c_den = durata + durata / 2 + durata / 4;
if (contaPunti == 3)
      c_den = durata + durata / 2 + durata / 4 + durata / 8;

XmlAttribute chord_num = xmlDoc.CreateAttribute("num");
duration.SetAttribute("num", c_num.ToString());
XmlAttribute chord_den = xmlDoc.CreateAttribute("den");
duration.SetAttribute("den", c_den.ToString());

// inserimento notehead, pitch e relativi attributi
XmlElement n_head = xmlDoc.CreateElement("notehead");
chord.AppendChild(n_head);
XmlElement pitch = xmlDoc.CreateElement("pitch");
n_head.AppendChild(pitch);
i = 0;
contaLettereMaiuscole = 0;
contaLettereMinuscole = 0;
sharpCount = 0;
flatCount = 0;
octaveA = 0;
octaveB = 0;
step_pitch = "";
alterazione = "natural";

for (; i < sOutput.Length; i++)
{
      if (sOutput[i] >= 'a' && sOutput[i] <= 'g')
            contaLettereMinuscole++;

      if (sOutput[i] >= 'A' && sOutput[i] <= 'G')
            contaLettereMaiuscole++;
}
octaveA = 3 + contaLettereMinuscole; //non prende l'ottava
octaveB = 4 - contaLettereMaiuscole;
XmlNode ottava = xmlDoc.CreateAttribute("octave");
if (contaLettereMaiuscole == 0)
      pitch.SetAttribute("octave", octaveA.ToString());
```

```
else
      pitch.SetAttribute("octave", octaveB.ToString());

XmlAttribute step = xmlDoc.CreateAttribute("step");
for (; i < sOutput.Length; i++)
{
      if (sOutput[i] >= 'a' && sOutput[i] <= 'g')
            contaLettereMinuscole++;
}
altezza47 = sOutput.Substring(contaCifre + contaPunti, 1);
for (; i < sOutput.Length; i++)
{
      if (sOutput[i] >= 'A' && sOutput[i] <= 'G')
            contaLettereMaiuscole++;
}
altezza13 = sOutput.Substring(contaCifre + contaPunti, 1); // come sopra

if (contaLettereMaiuscole == 0)
{
      step_pitch = altezza47.ToUpper();
      pitch.SetAttribute("step", step_pitch);
}
else
{
      step_pitch = altezza13;
      pitch.SetAttribute("step", step_pitch);
}
i = 0;
for (; i < sOutput.Length; i++)
{
      if (sOutput[i] == '#')
            sharpCount++;
      if (sOutput[i] == '-')
            flatCount++;
}
if (sharpCount == 1)
      alterazione = "sharp";
if (sharpCount == 2)
      alterazione = "double_sharp";
if (flatCount == 1)
      alterazione = "flat";
if (flatCount == 2)
      alterazione = "double_flat";

XmlAttribute accidental = xmlDoc.CreateAttribute("actual_accidental");
pitch.SetAttribute("actual_accidental", alterazione);
}
```

L'istruzione if iniziale permette di creare gli elementi <measure> quando il software legge una riga di testo corrispondente al simbolo "=" e di incrementare il valore dell'attributo "number" ogni volta che compare un "=" nel file **kern. La parola chiave "continue" consente di ripetere questa operazione ogni volta che ci troviamo davanti al simbolo "=".
All'interno dell'elemento voice, figlio di measure, creato congiuntamente ad essa, vanno inseriti gli elementi <chord> che rappresentano le note, sia se facenti parte di accordi, sia singole. Ad ogni riga del file **kern corrispondente ad una nota marcata nello spine, deve corrispondere un elemento <chord>, con il relativo riferimento allo spine (dato dall'attributo "event_ref". La durata della nota, il nome della nota, l'altezza e le eventuali alterazioni vengono codifcate come elementi figli di <chord>: l'elemento <duration> con gli attributi "num" e "den" indica la durata della nota, mentre <pitch>, contenuto

nell'elemento <notehead>, indica l'altezza della nota tramite gli attributi "octave", "step" e "actual_accidental".
Mentre il valore di "num" è sempre impostato a 1, per il calcolo del valore da attribuire a "den", bisogna tenere conto della presenza o meno dei punti di valore. Se la nota non è puntata, il valore dell'attributo "den" sarà identico alla variabile "durata" che contiene il valore della durata della nota per ogni riga del file **kern corrispondente ad una nota.

```
int c_num = 0, c_den = 0;
for (; i < sOutput.Length; i++)
{
	if (sOutput[i] >= '0' && sOutput[i] <= '9')
		contaCifre++;
}
if (sOutput != null && sOutput != "*clef*" && sOutput != "*M*" && sOutput != "=" &&
sOutput != "*-" && sOutput != "**kern")
{
	durata = Convert.ToInt32(sOutput.Substring(0, contaCifre));
	c_num = 1;
	c_den = durata;
}
```

Se, invece, la nota da convertire è puntata, bisogna fare un calcolo molto simile a quello svolto per impostare i vtu nel valore corretto. Anche in questa operazione ci è molto utile il contatore contaPunti che viene incrementato di un'unità per ogni punto di valore che trova. In questo modo, analogamente a quanto avviene per i vtu, le operazioni per il calcolo del denominatore possono essere implementate con le seguenti istruzioni. Questa volta, però, il valore da dividere non è vtu, ma durata:

```
if (contaPunti == 1)
	c_den = durata + durata / 2;
if (contaPunti == 2)
	c_den = durata + durata / 2 + durata / 4;
if (contaPunti == 3)
	c_den = durata + durata / 2 + durata / 4 + durata / 8;
```

Per quanto riguarda il contenuto dell'attributo "octave", invece, bisogna fare in modo che il software conti quante sono le lettere maiuscole o minuscole che in **kern codificano l'ottava di appartenenza della nota. Ricordiamo che in **kern le lettere che codificano le note sono le stesse utilizzate per i nomi delle note nella notazione anglosassone (da "a" a "g") e che:

3 lettere maiuscole codificano una nota della prima ottava (4CCC);
2 lettere maiuscole codificano una nota della seconda ottava (4CC);
1 sola maiuscola codifica una nota della terza ottava (4C);
1 sola minuscola codifica una nota della quarta ottava (4c);
2 minuscole una nota della quinta ottava (4cc);
3 minuscole una nota della sesta ottava (4ccc);
4 minuscole una nota della settima ottava(4cccc);

È necessario quindi contare le lettere che codificano l'ottava e, in base al fatto che siano minuscole o maiuscole, assegnarle all'attributo "octave". Questo conteggio e la successiva assegnazione vanno fatte per mezzo di un ciclo for che per ogni riga conta le occorrenze delle lettere (contaLettereMaiuscole & contaLettereMinuscole) e poi le assegna all'intero octaveA se i caratteri sono lettere minuscole oppure all'intero octaveB se si tratta di lettere maiuscole.

```
for (; i < sOutput.Length; i++)
{
	if (sOutput[i] >= 'a' && sOutput[i] <= 'g')
		contaLettereMinuscole++;

	if (sOutput[i] >= 'A' && sOutput[i] <= 'G')
		contaLettereMaiuscole++;
}
octaveA = 3 + contaLettereMinuscole; //non prende l'ottava
octaveB = 4 - contaLettereMaiuscole;
XmlNode ottava = xmlDoc.CreateAttribute("octave");
```

L'assegnazione del valore all'attributo "octave" di pitch va fatta anche in base al fatto che le lettere contate siano maiuscole o minuscole:

```
if (contaLettereMaiuscole == 0)
	pitch.SetAttribute("octave", octaveA.ToString());
else
	pitch.SetAttribute("octave", octaveB.ToString());
```

Il valore dell'attributo "step", invece, dev'essere costituito da un carattere maiuscolo da "A" a "G" che indica appunto il nome della nota in questione. Per attribuire il valore corretto a questo attributo il software analizza ogni riga e stabilisce a quale carattere corrisponda la prima occorrenza delle lettere che indicano il nome della nota e l'ottava ed assegna questo valore all'attributo "step". Dato che esso accetta solo caratteri maiuscoli, il carattere corrispondente al nome della nota per le ottave dalla quarta in su (indicate in **kern con lettere minuscole), dev'essere convertito in maiuscolo:

```
XmlAttribute step = xmlDoc.CreateAttribute("step");
for (; i < sOutput.Length; i++)
{
	if (sOutput[i] >= 'a' && sOutput[i] <= 'g')
		contaLettereMinuscole++;
}
altezza47 = sOutput.Substring(contaCifre + contaPunti, 1);
for (; i < sOutput.Length; i++)
{
	if (sOutput[i] >= 'A' && sOutput[i] <= 'G')
		contaLettereMaiuscole++;
}
altezza13 = sOutput.Substring(contaCifre + contaPunti, 1);
if (contaLettereMaiuscole == 0)
{
	step_pitch = altezza47.ToUpper();
	pitch.SetAttribute("step", step_pitch);
}
else
{
	step_pitch = altezza13;
	pitch.SetAttribute("step", step_pitch);
}
```

Per le alterazioni, invece, viene utilizzato un ciclo che conta le occorrenze dei diesis (indicati con # anche in **kern) e dei bemolli, che in **kern sono indicati con il carattere "-" invece che con la lettera "b", già usata per la codifica della nota si. Il contatore sharpCount viene incrementato di un'unità per ogni diesis che incontra, mentre flatCount viene incrementato per ogni bemolle che incontra. In base al valore di sharpCount e flatCount viene modificato il contenuto dell'attributo "actual_accidental" che di default è posto a "natural".

```
i = 0;
for (; i < sOutput.Length; i++)
{
      if (sOutput[i] == '#')
            sharpCount++;
      if (sOutput[i] == '-')
            flatCount++;
}
if (sharpCount == 1)
      alterazione = "sharp";
if (sharpCount == 2)
      alterazione = "double_sharp";
if (flatCount == 1)
      alterazione = "flat";
if (flatCount == 2)
      alterazione = "double_flat";
```

L'ultima istruzione del software, analoga a quella vista per il Software 1, consente all'utente di inserire il nome o il percorso del file XML e di salvarlo.

```
Console.WriteLine("Inserire il nome del file nella riga sottostante");
string Name = Console.ReadLine();
xmlDoc.Save(Name);
```

Di seguito l'output generato dall'esecuzione del Software 2. Il file .krn preso come esempio contiene le prime due battute del brano "San Sebastian" dei Sonata Arctica, utilizzato come file di prova anche per il precedente software.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mx SYSTEM "http://standards.ieee.org/downloads/1599/1599-2008/ieee1599.dtd">
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>
    <description>
      <main_title>San Sebastian</main_title>
      <author type="composer">Sonata Arctica</author>
    </description>
  </general>
  <logic>
    <spine>
      <event id="parte1ev_clef" timing="NULL" hpos="NULL" />
      <event id="parte1ev_time" timing="NULL" hpos="NULL" />
      <event id="parte1_evento1" timing="1536" hpos="1536" />
      <event id="parte1_evento2" timing="1536" hpos="1536" />
      <event id="parte1_evento3" timing="1024" hpos="1024" />
      <event id="parte1_evento4" timing="1024" hpos="1024" />
      <event id="parte1_evento5" timing="1024" hpos="1024" />
      <event id="parte1_evento6" timing="1024" hpos="1024" />
      <event id="parte1_evento7" timing="1024" hpos="1024" />
    </spine>
    <los>
      <staff_list>
        <staff id="staff_1">
          <clef shape="G" staff_step="2" octave_num="0" event_ref="parte1ev_clef" />
          <time_signature visible="yes" event_ref="parte1ev_time">
            <time_indication num="4" den="4" />
          </time_signature>
        </staff>
      </staff_list>
      <part id="parte_1">
        <voice_list>
          <voice_item id="voice1" staff_ref="staff_1" />
        </voice_list>
        <measure number="1">
        <voice voice_item_ref="voice1">
          <chord event_ref="parte1_evento1">
```

```
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="F" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento2">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="C" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento3">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="F" actual_accidental="natural" />
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="2">
        <voice voice_item_ref="voice1">
          <chord event_ref="parte1_evento4">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="D" actual_accidental="sharp" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento5">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="D" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento6">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="C" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento7">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="4" step="A" actual_accidental="sharp" />
            </notehead>
          </chord>
        </voice>
      </measure>
      </part>
    </los>
  </logic>
</ieee1599>
```

Conclusioni.

Questo elaborato costituisce un primo tentativo di ricerca di un metodo che consenta di convertire velocemente e correttamente un file **kern in un documento IEEE 1599.

La ricerca di un metodo per la traduzione dello spine e la scrittura del codice a ciò deputato sono state sicuramente le operazioni più brevi e semplici. Per quanto riguarda la stesura del codice per le operazioni di conversione del sublayer los, invece, si è trattato di un'operazione ben più lunga e complessa: tutto ciò nonostante, a prima vista, la converisone del los possa sembrare molto più semplice ed intuitiva di quella dello spine.

Sebbene la sintassi **kern sia concettualmente più vicina al sublayer los che allo spine, scrivere del codice per convertire correttamente da **kern al los di IEEE 1599 è molto complesso poiché los presenta una grande quantità di elementi e attributi che devono essere annidati e compilati correttamente per poter dar vita ad un file XML valido, oltre che ben formato. Molti infruttuosi tentativi sono stati fatti prima di giungere alla stesura del codice che consente la corretta costruzione del los e la corretta compilazione automatica degli attributi presenti negli elementi interni al sublayer los.

I prototipi software da me realizzati consentono solamente di creare i sublayers spine e los del layer logic di un documento MX a partire da un file **kern costituito da un unico pentagramma. L'augurio per il futuro è che, dopo un'analisi completa ed accurata di un nutrito campione delle partiture **kern al momento disponibili, si possa riuscire a creare uno strumento software in grado di effettuare la conversione automatica in IEEE 1599 anche per partiture più complesse, composte da due o più spines.

Appendice I

Software 1

SanSebastian.krn (file di esempio per la generazione dell'output del Software 1). La partitura contenuta nel file rappresenta in codifica **kern le prime 16 battute dell'introduzione del brano "San Sebastian" dei Sonata Arctica:

```
**kern
*clefG2
*M4/4
=1
4.ff
4.cc
4ff
=2
4dd#
4dd
4cc
4a#
=3
4g#
4cc
4dd#
4gg
=4
2gg
4ff
4gg
=5
4gg#
4gg
4ff
4gg
=6
4gg#
4gg
4ff
4gg
=7
4gg#
4gg
4ff
4dd#
=8
2dd#
4ff
4gg
=9
4.ff
4.cc
4ff
=10
4dd#
4dd
4cc
4a#
=11
4g#
4cc
4dd#
4ccc
=12
2aa#
4gg#
4aa#
=13
4ccc
4aa#
4.gg#
8aa#
=14
4ccc
4aa#
4.gg#
8aa#
=15
4ccc
4aa#
4.gg#
8aa#
=16
1aa
*-
```

File SonataI.xml, generato a partire dal file Sonata_I.krn, contenente l'omonima sonata di Bach, visto all'inizio del Capitolo IV:

```
<?xml version="1.0" encoding="UTF-8"?>
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>
    <description>
      <main_title>Sonata I</main_title>
      <author type="composer">J. S. Bach</author>
    </description>
  </general>
  <logic>
    <spine>
```

```
      <event id="parte1ev_clef" timing="NULL" hpos="NULL" />
      <event id="parte1ev_time" timing="NULL" hpos="NULL" />
      <event id="parte1_evento1" timing="1024" hpos="1024" />
      <event id="parte1_evento2" timing="1024" hpos="1024" />
      <event id="parte1_evento3" timing="1024" hpos="1024" />
      <event id="parte1_evento4" timing="1024" hpos="1024" />
      <event id="parte1_evento5" timing="1024" hpos="1024" />
      <event id="parte1_evento6" timing="1024" hpos="1024" />
      <event id="parte1_evento7" timing="1024" hpos="1024" />
      <event id="parte1_evento8" timing="1024" hpos="1024" />
      <event id="parte1_evento9" timing="1024" hpos="1024" />
      <event id="parte1_evento10" timing="1024" hpos="1024" />
      <event id="parte1_evento11" timing="1024" hpos="1024" />
      <event id="parte1_evento12" timing="1024" hpos="1024" />
      <event id="parte1_evento13" timing="1024" hpos="1024" />
      <event id="parte1_evento14" timing="1024" hpos="1024" />
      <event id="parte1_evento15" timing="1024" hpos="1024" />
      <event id="parte1_evento16" timing="1024" hpos="1024" />
      <event id="parte1_evento17" timing="1024" hpos="1024" />
      <event id="parte1_evento18" timing="1024" hpos="1024" />
      <event id="parte1_evento19" timing="1024" hpos="1024" />
      <event id="parte1_evento20" timing="1024" hpos="1024" />
    </spine>
  </logic>
</ieee1599>
```

File StarSpangledBanner.krn, contenente le prime cinque battute dell'inno nazionale degli Stati Uniti d'America e relativo output XML:

```
**kern
*clefG2
*M3/4
=1
8.g
16e
=2
4c
4e
4g
=
2cc
8.ee
16dd
=3
4cc
4e
4f#
=4
2g
8g
8g
*-
```

File StarSpangledBanner.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>
    <description>
      <main_title>The Star Spangled Banner</main_title>
      <author type="composer">J. S. Smith</author>
    </description>
```

```
  </general>
  <logic>
    <spine>
      <event id="parte1ev_clef" timing="NULL" hpos="NULL" />
      <event id="parte1ev_time" timing="NULL" hpos="NULL" />
      <event id="parte1_evento1" timing="768" hpos="768" />
      <event id="parte1_evento2" timing="256" hpos="256" />
      <event id="parte1_evento3" timing="1024" hpos="1024" />
      <event id="parte1_evento4" timing="1024" hpos="1024" />
      <event id="parte1_evento5" timing="1024" hpos="1024" />
      <event id="parte1_evento6" timing="2048" hpos="2048" />
      <event id="parte1_evento7" timing="768" hpos="768" />
      <event id="parte1_evento8" timing="256" hpos="256" />
      <event id="parte1_evento9" timing="1024" hpos="1024" />
      <event id="parte1_evento10" timing="1024" hpos="1024" />
      <event id="parte1_evento11" timing="1024" hpos="1024" />
      <event id="parte1_evento12" timing="2048" hpos="2048" />
      <event id="parte1_evento13" timing="512" hpos="512" />
      <event id="parte1_evento14" timing="512" hpos="512" />
    </spine>
  </logic>
</ieee1599>
```

Software 2.

SanSebastian(2measures).krn (file di esempio per il Software 2). Contiene le prime 2 battute del brano “SanSebastian” dei Sonata Arctica:

```
**kern
*clefG2
*M4/4
=
4.ff
4.cc
4ff
=
4dd#
4dd
4cc
4a#
```

File SonataI(spine+los).xml, generato a partire dal file Sonata_I.krn, visto all’inizio del Capitolo IV:

```
<?xml version="1.0" encoding="UTF-8"?>
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>
      <description>
        <main_title>Sonata I</main_title>
        <author type="composer">j. S. Bach</author>
      </description>
  </general>
  <logic>
    <spine>
      <event id="parte1ev_clef" timing="NULL" hpos="NULL" />
      <event id="parte1ev_time" timing="NULL" hpos="NULL" />
      <event id="parte1_evento1" timing="1024" hpos="1024" />
      <event id="parte1_evento2" timing="1024" hpos="1024" />
      <event id="parte1_evento3" timing="1024" hpos="1024" />
      <event id="parte1_evento4" timing="1024" hpos="1024" />
      <event id="parte1_evento5" timing="1024" hpos="1024" />
      <event id="parte1_evento6" timing="1024" hpos="1024" />
      <event id="parte1_evento7" timing="1024" hpos="1024" />
      <event id="parte1_evento8" timing="1024" hpos="1024" />
      <event id="parte1_evento9" timing="1024" hpos="1024" />
      <event id="parte1_evento10" timing="1024" hpos="1024" />
      <event id="parte1_evento11" timing="1024" hpos="1024" />
      <event id="parte1_evento12" timing="1024" hpos="1024" />
      <event id="parte1_evento13" timing="1024" hpos="1024" />
      <event id="parte1_evento14" timing="1024" hpos="1024" />
      <event id="parte1_evento15" timing="1024" hpos="1024" />
      <event id="parte1_evento16" timing="1024" hpos="1024" />
      <event id="parte1_evento17" timing="1024" hpos="1024" />
      <event id="parte1_evento18" timing="1024" hpos="1024" />
      <event id="parte1_evento19" timing="1024" hpos="1024" />
      <event id="parte1_evento20" timing="1024" hpos="1024" />
    </spine>
    <los>
      <staff_list>
        <staff id="staff_1">
          <clef shape="F" staff_step="4" octave_num="0" event_ref="parte1ev_clef" />
          <time_signature visible="yes" event_ref="parte1ev_time">
            <time_indication num="4" den="4" />
          </time_signature>
        </staff>
```

```
</staff_list>
<part id="parte_1">
  <voice_list>
    <voice_item id="voice1" staff_ref="staff_1" />
  </voice_list>
  <measure number="1">
    <voice voice_item_ref="voice1">
      <chord event_ref="parte1_evento1">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="C" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento2">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="G" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento3">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="D" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento4">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="A" actual_accidental="natural" />
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="2">
    <voice voice_item_ref="voice1">
      <chord event_ref="parte1_evento5">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="F" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento6">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="G" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento7">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="3" step="C" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento8">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="4" step="C" actual_accidental="natural" />
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="3">
```

```
  <voice voice_item_ref="voice1">
    <chord event_ref="parte1_evento9">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="A" actual_accidental="natural" />
      </notehead>
    </chord>
    <chord event_ref="parte1_evento10">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="F" actual_accidental="natural" />
      </notehead>
    </chord>
    <chord event_ref="parte1_evento11">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="B" actual_accidental="flat" />
      </notehead>
    </chord>
    <chord event_ref="parte1_evento12">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="G" actual_accidental="natural" />
      </notehead>
    </chord>
  </voice>
</measure>
<measure number="4">
  <voice voice_item_ref="voice1">
    <chord event_ref="parte1_evento13">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="A" actual_accidental="natural" />
      </notehead>
    </chord>
    <chord event_ref="parte1_evento14">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="D" actual_accidental="natural" />
      </notehead>
    </chord>
    <chord event_ref="parte1_evento15">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="A" actual_accidental="natural" />
      </notehead>
    </chord>
    <chord event_ref="parte1_evento16">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="2" step="A" actual_accidental="natural" />
      </notehead>
    </chord>
  </voice>
</measure>
<measure number="5">
  <voice voice_item_ref="voice1">
    <chord event_ref="parte1_evento17">
      <duration num="1" den="4" />
      <notehead>
        <pitch octave="3" step="D" actual_accidental="natural" />
      </notehead>
```

```
            </chord>
            <chord event_ref="parte1_evento18">
              <duration num="1" den="4" />
              <notehead>
                <pitch octave="3" step="G" actual_accidental="natural" />
              </notehead>
            </chord>
            <chord event_ref="parte1_evento19">
              <duration num="1" den="4" />
              <notehead>
                <pitch octave="3" step="C" actual_accidental="natural" />
              </notehead>
            </chord>
            <chord event_ref="parte1_evento20">
              <duration num="1" den="4" />
              <notehead>
                <pitch octave="3" step="F" actual_accidental="natural" />
              </notehead>
            </chord>
          </voice>
        </measure>
      </part>
    </los>
  </logic>
</ieee1599>
```

File StarSpangledBanner(los).xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<ieee1599 creator="Laboratorio di Informatica Musicale" version="1.0">
  <general>
    <description>
      <main_title>The Star Spangled Banner</main_title>
      <author type="composer">J. S. Smith</author>
    </description>
  </general>
  <logic>
    <spine>
      <event id="parte1ev_clef" timing="NULL" hpos="NULL" />
      <event id="parte1ev_time" timing="NULL" hpos="NULL" />
      <event id="parte1_evento1" timing="768" hpos="768" />
      <event id="parte1_evento2" timing="256" hpos="256" />
      <event id="parte1_evento3" timing="1024" hpos="1024" />
      <event id="parte1_evento4" timing="1024" hpos="1024" />
      <event id="parte1_evento5" timing="1024" hpos="1024" />
      <event id="parte1_evento6" timing="2048" hpos="2048" />
      <event id="parte1_evento7" timing="768" hpos="768" />
      <event id="parte1_evento8" timing="256" hpos="256" />
      <event id="parte1_evento9" timing="1024" hpos="1024" />
      <event id="parte1_evento10" timing="1024" hpos="1024" />
      <event id="parte1_evento11" timing="1024" hpos="1024" />
      <event id="parte1_evento12" timing="2048" hpos="2048" />
      <event id="parte1_evento13" timing="512" hpos="512" />
      <event id="parte1_evento14" timing="512" hpos="512" />
    </spine>
    <los>
      <staff_list>
        <staff id="staff_1">
          <clef shape="G" staff_step="2" octave_num="0" event_ref="parte1ev_clef" />
          <time_signature visible="yes" event_ref="parte1ev_time">
            <time_indication num="3" den="4" />
          </time_signature>
```

```
  </staff>
</staff_list>
<part id="parte_1">
  <voice_list>
    <voice_item id="voice1" staff_ref="staff_1" />
  </voice_list>
  <measure number="1">
    <voice voice_item_ref="voice1">
      <chord event_ref="parte1_evento1">
        <duration num="1" den="8" />
        <notehead>
          <pitch octave="4" step="G" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento2">
        <duration num="1" den="16" />
        <notehead>
          <pitch octave="4" step="E" actual_accidental="natural" />
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="2">
    <voice voice_item_ref="voice1">
      <chord event_ref="parte1_evento3">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="4" step="C" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento4">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="4" step="E" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento5">
        <duration num="1" den="4" />
        <notehead>
          <pitch octave="4" step="G" actual_accidental="natural" />
        </notehead>
      </chord>
    </voice>
  </measure>
  <measure number="3">
    <voice voice_item_ref="voice1">
      <chord event_ref="parte1_evento6">
        <duration num="1" den="2" />
        <notehead>
          <pitch octave="5" step="C" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento7">
        <duration num="1" den="8" />
        <notehead>
          <pitch octave="5" step="E" actual_accidental="natural" />
        </notehead>
      </chord>
      <chord event_ref="parte1_evento8">
        <duration num="1" den="16" />
        <notehead>
          <pitch octave="5" step="D" actual_accidental="natural" />
```

```
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="4">
        <voice voice_item_ref="voice1">
          <chord event_ref="parte1_evento9">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="5" step="C" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento10">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="4" step="E" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento11">
            <duration num="1" den="4" />
            <notehead>
              <pitch octave="4" step="F" actual_accidental="sharp" />
            </notehead>
          </chord>
        </voice>
      </measure>
      <measure number="5">
        <voice voice_item_ref="voice1">
          <chord event_ref="parte1_evento12">
            <duration num="1" den="2" />
            <notehead>
              <pitch octave="4" step="G" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento13">
            <duration num="1" den="8" />
            <notehead>
              <pitch octave="4" step="G" actual_accidental="natural" />
            </notehead>
          </chord>
          <chord event_ref="parte1_evento14">
            <duration num="1" den="8" />
            <notehead>
              <pitch octave="4" step="G" actual_accidental="natural" />
            </notehead>
          </chord>
        </voice>
      </measure>
    </part>
  </los>
 </logic>
</ieee1599>
```

Bibliografia

1. Selfridge-Field, E., ed. *Beyond MIDI: The Handbook of Musical Codes*. MIT Press, (1997).
2. CCARH Humdrum Portal, http://humdrum.ccarh.org/.
3. Humdrum Command Reference, http://humdrum.org/Humdrum/commands/.
4. Glenn Brookshear, J.: *Informatica – una panoramica generale*. Pearson Addison Wesley, (2006).
5. Piston, W.: Armonia. EDT, Torino, (1989).
6. KernScores, http://kern.ccarh.org/.
7. Ludovico, L.A.: *Key concepts of the IEEE 1599 Standard*. In: Baggi, D., Haus, G. (eds.) Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects. pp. 15–26. IEEE CS, Lugano (2008).
8. Moller, A., Schwartzbach, M.I.: *Introduzione a XML*. Addison Wesley Longman Italia, (2007).
9. LIM - Laboratorio di Informatica Musicale, http://www.lim.dico.unimi.it/.
10. IEEE Xplore, http://ieeexplore.ieee.org/.
11. *Il campione di Unicode, versione 5.0, quinta edizione*, il consorzio di Unicode, professionista del Addison-Wesley, (2006).
12. Skonnard, D. B.: Essential XML – Beyond Markup. Addison Wesley (2000).
13. Ludovico, L.A.: An XML Multi-layer Framework For Music Information Description. Ph.D. diss., Università degli Studi di Milano, Milan (2006).
14. IEEE 1599.dtd, http://standards.ieee.org/downloads/1599/1599-2008/ieee1599.dtd.
15. Online Humdrum Editor: http://kern.humdrum.org/cgi-bin/kern/kseditor.
16. Nagel, C.,Evejn, B., Glynn, J., Watson K., Skinner, M.: *Professional C# 2008*, Wiley & Sons Ltd., (2008).
17. Sharp J.: *Microsoft Visual C# 2008 – Passo per Passo*. Mondadori Informatica, (2008).
18. C# Key – Introduction to XML, http://www.csharpkey.com/csharp/xml/Lesson01.htm.
19. Albahari, J., Albahari, B.: *C# 4.0 in a Nutshell, Fourth Editino - The definitive Reference.* O'Reilly Media, (2010).