UNIVERSITÀ DEGLI STUDI DI MILANO
FACOLTÀ DI SCIENZE E TECNOLOGIE

DIPARTIMENTO DI INFORMATICA
GIOVANNI DEGLI ANTONI

Corso di Laurea Magistrale in
Informatica per la Comunicazione

ANALISI E MODELLAZIONE DI INFORMAZIONE MUSICALE SIMBOLICA

Relatore: Prof. Luca Andrea Ludovico
Correlatore: Prof. Adriano Baratè

Tesi di Laurea di:
Paolo Agnelli
Matr. Nr. 804331

ANNO ACCADEMICO 2018-2019

A Katia, splendida compagna di vita,
ed ai nostri figli Alessandro, Sara, Andrea.

Ringraziamenti

Ringrazio i professori Luca Andrea Ludovico e Adriano Baratè che con la loro disponibilità, anche al di fuori dell'orario canonico di servizio, mi hanno aiutato a completare la tesi senza allontanarmi dalla mia famiglia e dal luogo di lavoro. Un sentito ringraziamento anche a tutti gli altri professori che mi hanno accompagnato in questo percorso di crescita professionale.

Grazie ai miei nonni, principali sponsor e fan, e ai miei genitori che mi hanno sempre appoggiato e che sono stati disponibili, così come i suoceri, nei momenti in cui ho avuto bisogno di aiuto per affrontare gli imprevisti logistici.

Grazie agli amici del *Majons Club* che mi attendono in Valmalenco per festeggiare e ad Alessandro per la nostra bella amicizia!

Un abbraccio ai miei figli che in questi anni hanno lasciato tempo al papà di studiare per gli esami, dopo il lavoro, e di preparare la tesi durante le festività.

Grazie Katia: senza la tua pazienza non ce l'avrei mai fatta!

Indice

Capitolo 1

Introduzione

Nei primi anni 2000, presso il Laboratorio di Informatica Musicale dell'Università degli Studi di Milano, è stato ideato un formato aperto, basato su XML, per la descrizione dell'informazione musicale. Questo formato, inizialmente denominato MX (Musical application using XML), è stato oggetto di standardizzazione IEEE nell'anno 2008: il gruppo di lavoro *WG_1599 - Working Group for XML Musical Application*, che ha visto la partecipazione di decine di esperti internazionali in materia, ha lavorato con l'obbiettivo di definire un linguaggio comune, ora noto come IEEE 1599, per la rappresentazione multilivello dell'informazione musicale.

Tra le nuove sfide che si presentano a distanza di un decennio, c'è quella di permettere ad IEEE 1599, e alle applicazioni che lo utilizzano, di svincolarsi dalla struttura di rappresentazione dati proprietaria: generalizzando la struttura dei layer, in modo da renderne possibile la personalizzazione, lo standard potrà continuare la sua evoluzione diventando estendibile a piacere e consentendo, ad esempio, di accogliere informazioni provenienti da formati alternativi di codifica dell'informazione musicale (come quelli descritti in [1]).

Nell'ottica di affrontare questa sfida, il presente elaborato di tesi tratta l'implementazione di un modello generico, liberamente estendibile, per la rappresentazione dell'informazione musicale. Nel Capitolo 2 viene fornita una panoramica dei principali formati disponibili per la codifica multilivello dell'informazione musicale e viene illustrato un esempio di applicativo (player multimediale IEEE 1599) che potrebbe beneficiare del modello proposto per ampliare la portata dei contenuti fruibili. Il successivo Capitolo 3 descrive gli aspetti relativi alla progettazione delle strutture dati di base e all'implementazione dei meccanismi di controllo per verificare che ne venga fatto un uso consistente. Il Capitolo 4 dettaglia l'implementazione delle strutture dati e degli algoritmi specifici per trattare i formati file analizzati nel Capitolo 2. Infine vengono illustrati alcuni esempi di utilizzo pratico delle strutture dati (Capitolo 5) e tratte le conclusioni del caso (Capitolo 6).

Capitolo 2

Stato dell'arte

2.1 Informazione musicale

La musica è un'arte che ben si presta ad essere trattata dal punto di vista informatico: il processo di digitalizzazione[1] di beni culturali di tipo musicale permette di liberare il contenuto informativo dalle catene dell'involucro fisico. Campagne di digitalizzazione e informatizzazione[2] hanno interessato, in tempi recenti, archivi storici di primaria rilevanza: basti citare, a titolo d'esempio, i processi che hanno riguardato gli archivi storici del teatro alla Scala di Milano [2] o l'archivio musicale Ricordi. I contenuti vengono valorizzati e resi fruibili con modalità alternative [3] che spesso comportano benefici per la conservazione delle opere (in certi casi l'obbiettivo primario del processo è proprio quello di garantire una vita più duratura ai beni).

L'avvento dell'era digitale ha permesso la nascita di strumenti che hanno impattato in maniera notevole sulle attività concernenti l'informazione musicale. A titolo di esempio si possono citare alcuni dei processi interessati:

- produzione - software di sintesi musicale, supporto alla composizione, editing, registrazione, effetti digitale, DAW etc.;
- riproduzione - dispositivi hardware per la fruizione mobile (dai primi lettori portatili agli smartphones e ai dispositivi indossabili), piattaforme per condivisione di contenuto musicale e di streaming, etc.;
- rappresentazione - applicativi e formati file per codificare e rendere intellegibile il contenuto informativo musicale.

[1] trasformazione dell'informazione da forma analogica a forma digitale

[2] organizzazione dei beni digitalizzati arricchiti di informazioni (metadati)

In particolare, per quanto riguarda il processo di rappresentazione dell'informazione musicale, nel corso degli anni abbiamo assistito, e continuiamo ad assistere, all'evoluzione di formati file sempre più sofisticati che, basandosi su algoritmi avanzati, permettono di codificare l'informazione del dominio di interesse (audio, immagine, video) riducendo progressivamente la discrepanza tra quello che i nostri sensi sono in grado di percepire e la quantità di risorse necessarie per immagazzinare/trasmettere l'informazione. Riuscire ad integrare in modo coerente e strutturato le varie modalità di rappresentazione dell'informazione musicale (struttura simbolica, immagine di uno spartito, registrazione audio, ripresa video, etc.) rappresenta una delle sfide più significative sulle quali si sta lavorando da inizio secolo.

2.2 Organizzazione a layer

Per descrivere le sfaccettature con le quali si può codificare un oggetto musicale, si può ricorrere ad una rappresentazione a strati: ogni livello offre una particolare vista del contenuto informativo. Le ricerche condotte negli ultimi decenni sul tema, come quelle descritte in [4], hanno portato all'individuazione di una struttura composta dai seguenti layer:

- *General* - metadati di una composizione musicale (autore, titolo, opera, etc.);
- *Logic* - descrive una composizione musicale come insieme organizzato di eventi musicali o sonori, nel rispetto di quanto ideato dal compositore;
- *Notational* - descrive la rappresentazione in simboli musicali su partitura degli eventi musicali o sonori (in certi casi l'aspetto notazionale è parte viva e fondamentale della composizione);
- *Structural* - descrive l'aggregazione logica di eventi o di simboli musicali (temi, regioni notali, motivi ed altri aspetti frutto di analisi musicologica);
- *Performance* - descrive i parametri per la sintesi / generazione di eventi musicali o sonori (MIDI, CSound, etc.);
- *Audio* - descrive le proprietà audio del materiale musicale (file audio/video in formato compresso / non compresso).

Come già analizzato in [1], le informazioni rappresentate in un layer possono essere messe in relazione con altre rappresentazioni dello stesso tipo (ad esempio differenti performance audio di un brano) o con quelle codificate in altri livelli (ad esempio associare un file audio alla sua rappresentazione grafica in partitura). Le modalità con le quali avviene il linking tra i vari layer è una caratteristica peculiare del

formato utilizzato: ad esempio nello standard IEEE 1599 (descritto nella sezione 2.5) il collegamento avviene con una struttura a stella il cui centro è rappresentato dagli eventi contenuti nel layer Logic (spine). Ci sono anche casi nei quali la rappresentazione multilayer può essere inadatta o non esaustiva (ad esempio nell'improvvisazione jazz): la tendenza verso cui si stanno orientando alcuni gruppi di lavoro è quella di generalizzare il concetto di layer, fornendo la possibilità all'utente di personalizzarne la struttura.

I formati Sul finire del secolo scorso, sono stati proposti differenti formati di codifica dell'informazione musicale: quelli che hanno avuto maggiore diffusione si sono spesso focalizzati sulla rappresentazione delle informazioni relative ad un insieme limitato di layer (tipicamente Logic, General e Notational). Le soluzioni adottate per la codifica delle informazioni erano tipicamente basate su formati di tipo testuale: a titolo d'esempio si possono citare *abcnotation* [3] e *Humdrum **kern*[4] (descritto nella sezione 2.3) che sono stati impiegati con successo per la codifica di centinaia di migliaia di brani, pubblicati in archivi online liberamente accessibili[5].

I formati più recenti hanno potuto beneficiare dell'introduzione dell'eXtensible Markup Language (XML) per gestire strutture complesse che, con differenti livelli di completezza, permettono una rappresentazione multilayer della musica. Come trattato in dettaglio in [5] e in [6], XML presenta peculiari caratteristiche di modularità, robustezza e leggibilità, oltre a garantire la possibilità di separare le strutture descrittive dai dati: questi fattori chiave lo rendono particolarmente adatto allo scopo di descrivere materiale musicale. Tra i principali formati che adottano questo tipo di codifica si possono citare:

- IEEE 1599;
- Music Encoding Initiative (MEI);
- Music XML.

Oltre ad essere accomunati dall'uso di XML, questi sistemi di codifica offrono la possibilità di fare riferimento a file esterni: si tratta di un notevole beneficio, in quanto permette di riutilizzare informazioni già disponibili in archivi preesistenti (immagini, file audio, video, teche multimediali etc.). Ognuno di questi sistemi presenta peculiarità distintive: in [1] e in [7] i formati vengono messi a confronto per far emergere i rispettivi punti di forza/debolezza e prospettare scenari evolutivi futuri.

[3] `http://abcnotation.com/wiki/abc:standard`
[4] `https://www.humdrum.org/guide/ch02/`
[5] `http://kern.humdrum.org/`

Nelle prossime sezioni vengono approfonditi i dettagli relativi ai formati che sono stati oggetto di analisi per la realizzazione del progetto di tesi.

2.3 Humdrum **kern

Humdrum, creato da David Huron negli anni '80, è un insieme di strumenti software destinati ad aiutare musicologi e ricercatori nel compito di codificare, manipolare e produrre un'ampia varietà di rappresentazioni musicali. Esempi pratici di utilizzo di Humdrum sono descritti in [8].

**kern è una rappresentazione di tipo testuale ottenibile con gli strumenti Humdrum: i file relativi hanno estensione .krn e codificano informazioni di base relative a brani di musica occidentale. Dato che Humdrum nasce per scopi analitici, la funzione principale di **kern è quella di rappresentare le informazioni simboliche afferenti al layer Logic (altezza, durata delle note e delle pause, etc.). Questo non esclude che possano essere presenti, come spesso accade, metadati (layer General) e informazioni riconducibili ad altri layer (Notational e Structural) che, con alcune limitazioni, possono essere utilizzati per la generazione di partiture. Da un brano rappresentato in formato **kern, ad esempio, è possibile generare un file in formato MEI da utilizzare per produrre partiture SVG con Verovio [9]: questo procedimento viene sfruttato per rendere disponibili rappresentazioni grafiche di tutti quei brani **kern della libreria online Humdrum per i quali non sia già presente la scansione di una partitura (per approfondimenti si può fare riferimento a [10]).

I file **kern, in maniera analoga ad altre rappresentazioni prodotte con il toolkit Humdrum, sono composti da linee (record) contenenti testo e sono strutturati per essere letti in maniera bidimensionale: gli elementi sono tipicamente disposti in una struttura a griglia con token orizzontali, che rappresentano informazioni simultanee, separati dal carattere di tabulazione per ottenere un allineamento verticale; le colonne (denominate *spine*) rappresentano informazioni che si susseguono in sequenza nel tempo. Sono ammessi i seguenti tipi di record mutualmente esclusivi:

- comment records;
- interpretation records;
- data records.

Non sono ammessi record vuoti.

Comment records I *comment records* servono per inserire commenti ed iniziano sempre con il carattere '!': possono essere locali (riferiti ad una colonna) oppure globali; nel secondo caso iniziano con due punti esclamativi ed occupano un'intera riga. I reference records sono un caso particolare di comment record globale: servono a fornire i metadati riguardanti il brano musicale (layer General: titolo, compositori, riferimenti all'opera di appartenenza, etc.). I reference record iniziano con 3 punti esclamativi seguiti da una sigla di 3 caratteri che identifica il tipo di metadato e dal carattere ':' utilizzato come separatore.

A titolo d'esempio si riporta il seguente frammento di file in rappresentazione **kern che descrive i metadati del brano *Mazurka in C-sharp Minor* di F. Chopin:

Listing 2.1: reference records in **kern

```
!!!COM: Chopin, Frederic
!!!CDT: 1810///-1849///
!!!OTL: Mazurka in C-sharp Minor, Op. 41, No. 1
!!!OPS: Op. 41
!!!ONM: No. 1
!!!OMD: Maestoso
!!!ODT: 1838/11/28/
!!!PDT: 1840///
!!!PPP: Leipzig, Paris and London (1840)
!!!ODE: Stefan Witwicki
!!!AIN: piano
```

Interpretation records Gli *interpretation record*s iniziano con il carattere '*' e forniscono indicazioni sul tipo di rappresentazione: si distinguono tra *tandem records* ed *exclusive records*; nel secondo caso iniziano con il doppio asterisco. Ogni file deve contenere almeno un *exclusive record* che fornisca indicazioni su come trattare le informazioni codificate nello spine di interesse: i file .krn devono contenere almeno un *exclusive record* con un token ***kern*. Oltre al token (o ai token) ***kern*, in un *exclusive record* spesso si trova anche un token ***silbe* (o ***text*) che descrive uno spine che conterrà informazioni sul testo del brano. I *tandem records* forniscono informazioni supplementari, come quelle sulle alterazioni in chiave (**k[]*), sulla metrica (**M*) e sul tempo(**MM*).

Listing 2.2: interpretation records in **kern

```
**kern    **kern
*staff2   *staff1
*clefF4   *clefG2
*k[f#c#g#d#]      *k[f#c#g#d#]
*c#:      *c#:
```

∗M3/4 ∗M3/4
∗MM128 ∗MM128

Humdrum prevede inoltre la presenza di particolari *tandem record* contenenti token definiti *spine paths indicators*: questi elementi servono a gestire in maniera dinamica il numero di colonne/spine presenti in una rappresentazione **kern. Esistono cinque *spine path indicators* più un simbolo placeholder:

- *+ - nuovo spine a destra dello spine corrente (*new*);
- *- terminatore spine (*end*);
- *^ divisione di uno spine in due spine (*split*);
- *v unione di due o più spine in uno spine (*join*);
- *x scambio sul posto delle informazioni di due spine (*exchange*);
- * indica che non devono essere eseguite azioni sullo spine indicato (*placeholder*).

Gli *spine paths*, nel contesto **kern, vengono sovente utilizzati per rappresentare sequenze di note eseguite estemporaneamente da voci aggiuntive in determinate misure: nel momento in cui la voce entra in gioco, si può aggiungere uno spine in cui si descrivono gli eventi musicali relativi; la colonna verrà poi riunita a quella principale nel momento in cui la voce terminerà la propria esecuzione. Una soluzione di questo tipo, ad esempio, viene adottata per descrivere l'andamento delle voci alla misura 9 della già citata *Mazurka in C-sharp Minor* di F. Chopin rappresentata in Figura 1.

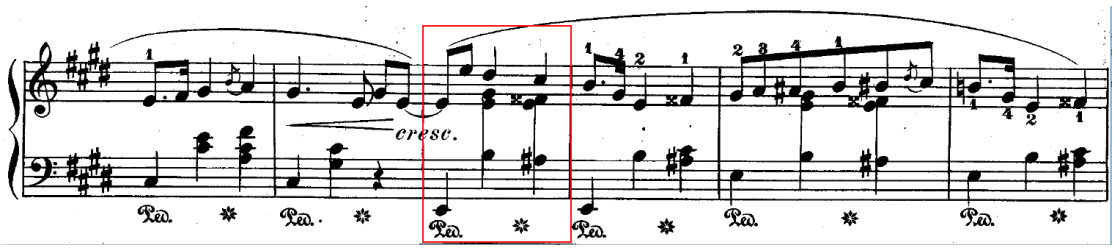


Figura 1: misure 7-12 Mazurka in C-sharp Minor (F. Chopin)

Il seguente estratto del file originale documenta l'utilizzo degli *spine paths*.

Listing 2.3: spine split + spine join nella misura 9

```
=9          =9
*           *^
4EE/        (8e/L]    4r
.           8ee/J     .
4B\         4dd#/     4e\  4g#\
4A#\        4cc#/     4e\  4f##\
*           *v        *v
=10         =10
```

Ovviamente questo utilizzo non è l'unico possibile: si possono presentare situazioni che richiedono accorgimenti particolari per essere gestite correttamente (ad esempio scambio di parti, voci multiple, etc.): il manuale d'uso di Humdrum regolamenta l'utilizzo di questi tipi di record fornendo casi pratici di utilizzo.

Data records I *data records* contengono i token con le informazioni relative al layer Logic (note, pause, sillabe) e alle misure. Le misure vengono codificate con il carattere '=', che corrisponde alla barra singola e che può essere ripetuto per rappresentare la barra doppia, seguito da un identificativo numerico. Un evento musicale (nota/pausa/testo) codificato in una determinata riga di uno spine può contenere più elementi separati da uno spazio (denominati *stop*): questa soluzione viene tipicamente impiegata per rappresentare gli accordi. I token delle pause sono denotati dall'utilizzo del carattere 'r' abbinato ad un valore numerico che ne identifica la durata: gli attributi afferenti alla rappresentazione grafica del simbolo vengono codificati accostando caratteri speciali ad hoc. Il carattere 'r' può essere ripetuto più volte, nello stesso token, per identificare una pausa che occupa l'intera misura. La convenzione con la quale viene descritta una nota è la seguente:

- il token contiene il nome della nota in notazione anglosassone (c=do, d=re, e=mi, f=fa, g=sol, a=la, b=si);
- l'altezza della nota viene indicata con la ripetizione del carattere che ne identifica il nome: la singola lettera minuscola corrisponde all'ottava centrale del pianoforte (ad esempio *c* corrisponde a do4), successive ripetizioni del carattere minuscolo indicano ottave superiori (*cc* corrisponde a do5, *ccc* a do6 etc.). Le ottave inferiori a quella centrale vengono indicate con caratteri maiuscoli e loro ripetizioni (*C* corrisponde a do3, *CC* a do2 etc.);
- le alterazioni vengono esplicitate per ogni nota accostando al nome il simbolo relativo (eventualmente ripetuto). Esempi di valori ammissibili:

- - (doppio bemolle);

- (bemolle);

n (bequadro);

(diesis);

(doppio diesis);

- la durata viene espressa con un valore numerico accostato al nome della nota (tipicamente lo precede);
- possono essere presenti caratteri speciali che descrivono attributi notazionali o tipologie di note particolari[6].

Il valore numerico che esprime la durata, comune ai token di note e pause, segue la convenzione descritta in Tabella 1.

Tabella 1: Valori di durata di note e pause in **kern

Significante	**Significato**
0	breve
1	semibreve
2	minima
4	semiminima
8	croma
16	semicroma
32	biscroma
64	semibiscroma
128	fusa

Con l'eccezione del valore associato alla nota breve, impostato di prassi a 0, il significante corrisponde al reciproco della durata relativa della nota (es: 16 indica la nota da $\frac{1}{16}$). Alla durata della nota può essere accostato il punto di valore, rappresentato con il carattere '.' (eventualmente ripetuto per codificare punti doppi o tripli).

I gruppi irregolari vengono gestiti secondo la stessa logica: ogni nota ha durata corrispondente al reciproco del valore relativo; ad esempio il valore 6 rappresenta la durata di ogni nota in una terzina di quarti (3 note da $\frac{1}{4}$ nel tempo di due semiminime / una minima), il valore 10 rappresenta la durata relativa di 1/10, cioè quella corrispondente ad una divisione in 5 parti della durata di due semiminime (una minima), etc.

[6] `http://www.humdrum.org/rep/kern/#signifiers`

Un token che contiene unicamente il carattere '.' svolge la funzione di placeholder (le specifiche del formato non ammettono la presenza di elementi vuoti).

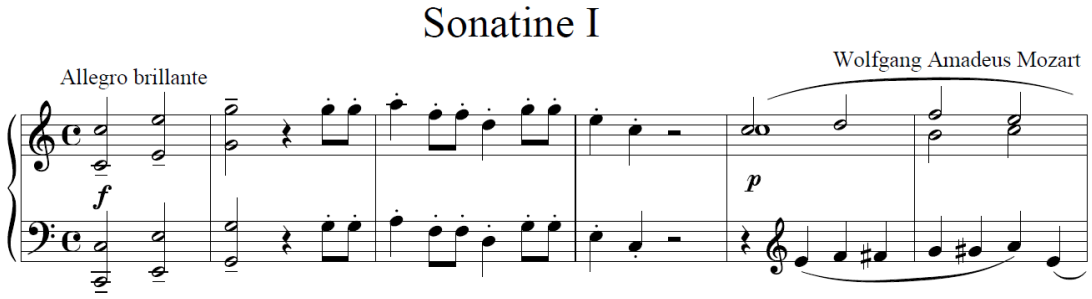


Figura 2: Le prime 6 battute della Sonatina in C-Major di Mozart

Il seguente estratto di codice mostra un utilizzo pratico degli elementi della rappresentazione **kern, finora descritti, per codificare le informazioni musicali relative alle prime misure del brano *Sonatina in C-major* di *W.A.Mozart* illustrate in Figura 2.

Listing 2.4: rappresentazione Humdrum **kern

```
!!!COM: Mozart, Wolfgang Amadeus
!!!CDT: 1756///-1791///
!!!OTP: Viennese Sonatina No. 1 in C-major
!!!OTL: Sonatina in C-major
!!!OMV: No. 1
!!!GNM: No. 1
!!!OMD: Allegro
!!!ENC: Craig Stuart Sapp
!!!EEV: 19 Feb 2001
!!!YEC: Copyright 2001 by Craig Stuart Sapp
!!!YEN: United States of America
**kern  **kern
*k[]    *k[]
*C:     *C:
*M4/4   *M4/4
*MM152  *MM152
*>[A,A,B,B]     *>[A,A,B,B]
*>A     *>A
2CC 2C  2c 2cc
2EE 2E  2e 2ee
```

```
=2          =2
2GG  2G     2g  2gg
4r          4r
8G'         8gg'
8G'         8gg'
=3          =3
4A'         4aa'
8F'         8ff'
8F'         8ff'
4D'         4dd'
8G'         8gg'
8G'         8gg'
=4          =4
4E'         4ee'
4C'         4cc'
2r          2r
=5          =5
*           *^
4r          1cc         2cc
4e          .           .
4f          .           2dd
4f#         .           .
=6          =6          =6
4g          2b          2ff
4g#         .           .
4a          2cc         2ee
4e          .           .
*           *v          *v
```

2.4 MusicXML

MusicXML nasce originariamente come rivisitazione in chiave XML di MuseData [11], con alcune caratteristiche ereditate da Humdrum. Sviluppato da *Recordare LLC*, cresce nel tempo con l'obbiettivo di fornire un formato per l'interscambio di informazioni tra applicativi musicali, ambendo a diventare uno standard universale per la codifica della notazione di musica occidentale in modo da consentire la conservazione delle partiture per usi futuri. Supportato da centinaia di applicazioni commerciali[7], rappresenta uno dei casi di maggior successo e diffusione per un

[7] `https://www.musicxml.com/software/`

formato di descrizione musicale. L'ultima versione dello standard (3.1 del dicembre 2017), analizzata nel presente lavoro, è stata sviluppata dal gruppo di lavoro *W3C Music Notation Community Group* ed è rilasciata con licenza W3C che ne consente il libero utilizzo. Per la sua natura di file di interscambio, con elementi derivati dalla codifica MIDI (ad esempio per la rappresentazione della durata delle note), un file musicxml tipicamente contiene informazioni afferenti ai layer General, Logic, Notational e Performance. Lo standard non supporta nativamente tipologie di ripresentazione riconducibili ad altri layer: si ritiene utile riportare, per completezza, che risulta possibile codificare informazioni aggiuntive mediante l'adozione di un ulteriore standard sviluppato dal medesimo gruppo W3C (MNX, che non viene trattato nel presente lavoro). MusicXML riesce con successo a soddisfare gli scopi per cui è stato ideato (codifica della notazione occidentale), di converso può mancare del livello di astrazione necessario per rappresentare altri tipi di stili notazionali (in [1] viene definito come complementare dello standard MEI).

Un file MusicXml ha estensione .musicxml e il contenuto può essere organizzato con due strutture gerarchiche alternative:

- rappresentazione *score-partwise* (misure nella parti);
- rappresentazione *score-timewise* (parti nelle misure).

Utilizzando i fogli di stile XSL forniti a corredo del toolkit di MusicXML, risulta possibile passare agevolmente da un tipo di rappresentazione all'altro con una semplice operazione di trasformazione. Ai fini del presente lavoro si fa riferimento alla rappresentazione in formato *score-partwise*.

Gli elementi ammessi in un file MusicXML sono descritti nei file DTD (Document Type Definition)[8] e XSD (XML Schema)[9] liberamente scaricabili dal sito internet ufficiale. Nel caso della rappresentazione *score-partwise*, l'omonimo elemento radice contiene l'header del documento e la descrizione di una o più parti. L'header del documento permette di specificare, mediante l'uso di differenti tipologie di tag, i metadati facoltativi del documento (layer Logic) e l'elenco, obbligatorio, delle parti presenti: in questo elenco (*part-list*), ogni parte viene identificata con un nome e un id univoco; opzionalmente si possono specificare proprietà aggiuntive utili per la rappresentazione grafica (layer Notational) o per la riproduzione MIDI (layer Performance).

[8] `https://www.musicxml.com/for-developers/musicxml-dtd/`
[9] `https://www.musicxml.com/for-developers/musicxml-xsd/`

Il seguente frammento XML fornisce un esempio degli elementi finora descritti:

Listing 2.5: header documento + part-list

```
<work>
  <work-number>D. 839</work-number>
  <work-title>Ave Maria (Ellen's Gesang III) - Page 1</work-title>
</work>
<identification>
  <creator type="composer">Franz Schubert</creator>
  <creator type="poet">Walter Scott</creator>
  <creator type="translator">D. Adam Storck</creator>
  <rights>Copyright 2002 MakeMusic, Inc.</rights>
</identification>
<part-list>
  <score-part id="P1">
    <part-name>Voice</part-name>
    <score-instrument id="P1-I14">
      <instrument-name>Choir Aahs</instrument-name>
      <instrument-sound>voice.vocals</instrument-sound>
      <solo/>
    </score-instrument>
    <midi-instrument id="P1-I14">
    <midi-channel>1</midi-channel>
    <midi-program>53</midi-program>
    <volume>80</volume>
    <pan>0</pan>
    </midi-instrument>
  </score-part>
  <score-part id="P2">
    <part-name>Piano</part-name>
    <score-instrument id="P2-I1">
      <instrument-name>Grand Piano</instrument-name>
      <instrument-sound>keyboard.piano</instrument-sound>
    </score-instrument>
    <midi-instrument id="P2-I1">
      <midi-channel>2</midi-channel>
      <midi-program>1</midi-program>
      <volume>80</volume>
      <pan>0</pan>
    </midi-instrument>
  </score-part>
</part-list>
```

Le parti vere e proprie (*part*), che compaiono allo stesso livello degli header tag, sono composte da misure (*measure*) descritte con appositi sotto elementi (*divisions*, *key*, *time*, *clef*, *transpose*, etc.).

Listing 2.6: descrizione delle caratteristiche di una misura

```
<part id="P1">
  <measure number="1">
    <attributes>
      <divisions>48</divisions>
      <key>
        <fifths>-2</fifths>
        <mode>major</mode>
      </key>
      <time symbol="common">
        <beats>4</beats>
        <beat-type>4</beat-type>
      </time>
      <clef>
        <sign>G</sign>
        <line>2</line>
      </clef>
    </attributes>
  </measure>
  [...]
</part>
```

Quando sono presenti delle note, queste vengono codificate con l'elemento *note* contenuto nelle misure. Le caratteristiche delle note afferenti al layer Logic vengono descritte con i seguenti elementi:

- *chord*: indica se la nota forma un accordo con la precedente;
- *pitch*: rappresenta l'altezza ed è descritto con gli elementi *step* (nome nota, in notazione anglosassone), *alter* (alterazione) e *octave* (ottava). L'alterazione, quando presente, viene espressa in formato numerico: -1 = bemolle, 0 = bequadro, 1 = diesis etc. (frazioni di intero rappresentano micro-alterazioni);
- *duration*: valore numerico che descrive la durata della nota in riferimento alle divisioni della nota da $\frac{1}{4}$ (*divisions* della misura di appartenenza);
- *tie*: indica la presenta di una legatura di valore;
- *lyric*: descrive il testo associato ad una nota, con elementi che rappresentano il testo e la suddivisione in sillabe;
- *dot*: descrive la presenza di uno o più punti di durata (se ripetuto);
- *voice*: descrive la voce a cui appartiene la nota;

- *staff*: descrive il pentagramma a cui appartiene la nota.

Le pause vengono rappresentate come *note* degeneri che contengono l'elemento *rest* in luogo di *pitch*.

La presenza di elementi appartenenti a gruppi irregolari viene espressa con il tag *time-modification* che codifica il rapporto tra la quantità di elementi del gruppo irregolare (*actual-notes*) e quella degli elementi che normalmente occuperebbero lo stesso tempo all'interno della misura (*normal-notes*).

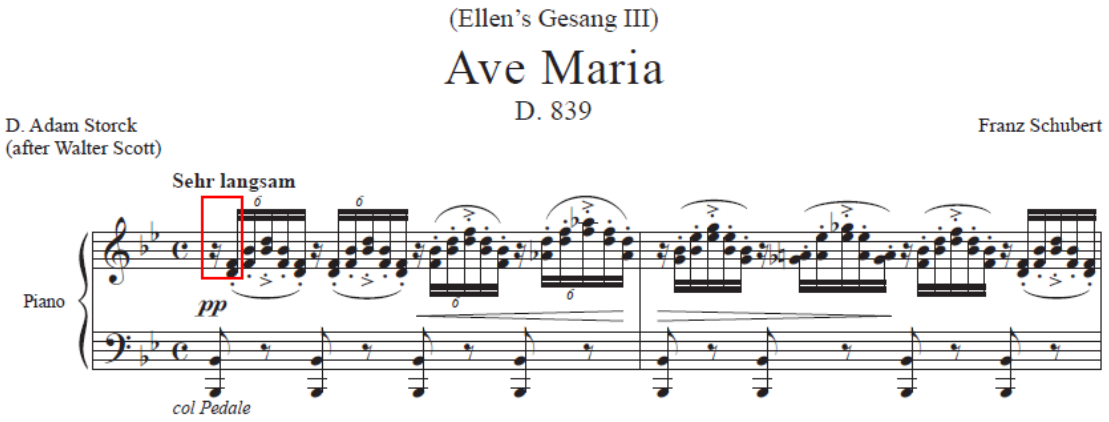


Figura 3: Prime due misure del brano "Ave Maria" (F. Schubert)

Il seguente esempio mostra l'utilizzo della notazione finora descritta per rappresentare i primi due elementi della sestina presente nella prima misura della parte di piano (mano destra) dell'*Ave Maria di F. Shubert* (evidenziati in rosso in Figura 3).

Listing 2.7: primi due elementi di una sestina (pausa e accordo)

```
<part id="P2">
  <measure number="1" width="600">
    <note default-x="128">
      <rest/>
      <duration>8</duration>
      <voice>1</voice>
      <type>16th</type>
      <time-modification>
        <actual-notes>6</actual-notes>
        <normal-notes>4</normal-notes>
      </time-modification>
    </note>
    <note default-x="147">
      <pitch>
        <step>D</step>
```

```
        <octave>4</octave>
      </pitch>
      <duration>8</duration>
      <voice>1</voice>
      <type>16th</type>
      <time-modification>
        <actual-notes>6</actual-notes>
        <normal-notes>4</normal-notes>
      </time-modification>
    </note>
    <note default-x="147">
      <chord/>
      <pitch>
        <step>F</step>
        <octave>4</octave>
      </pitch>
      <duration>8</duration>
      <voice>1</voice>
      <type>16th</type>
      <time-modification>
        <actual-notes>6</actual-notes>
        <normal-notes>4</normal-notes>
      </time-modification>
    </note>
  </measure>
</part>
```

Lo standard prevede appositi tag per lo spostamento lungo l'asse temporale:

- *backup* - serve per "riavvolgere il nastro" del numero di unità temporali specificate con l'elemento *duration*;
- *forward* - consente di spostarsi ad un istante di esecuzione successivo per una determinata voce.

Questi elementi vengono tipicamente impiegati per gestire l'andamento di voci secondarie e/o spostamenti tra pentagrammi. A titolo d'esempio si può citare la parte di pianoforte del brano *Wie Melodien zieht es mir* di Johannes Brahms rappresentata in Figura 4 - il frammento evidenziato in rosso viene codificato in MusicXML seguendo i seguenti passaggi:

- vengono descritti gli eventi della prima voce: le ultime due note subiscono un cambio di pentagramma;
- si utilizza l'elemento backup per tornare ad inizio misura;
- vengono descritte le note della seconda voce;

- con l'elemento forward ci si sposta all'inizio della seconda misura.

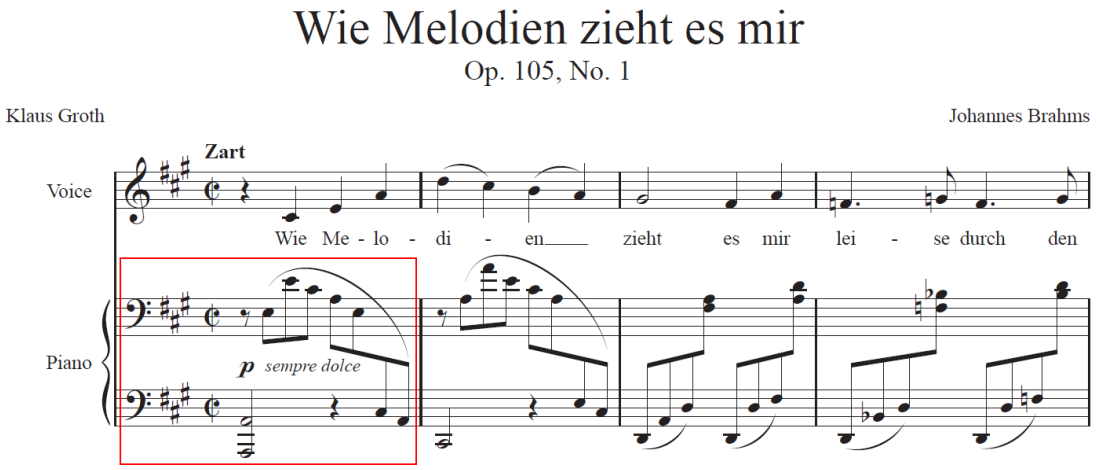


Figura 4: Prime battute del brano "Wie Melodien zieht es mir" (J.Brahms)

Di seguito viene riportata la codifica XML risultante:

Listing 2.8: utilizzo di backup e forward

```
<part id="P2">
  <measure number="1" width="340">
    [...]
    <divisions>8</divisions>
    <key>
      <fifths>3</fifths>
      <mode>major</mode>
    </key>
    <time symbol="cut">
      <beats>2</beats>
      <beat-type>2</beat-type>
    </time>
    <staves>2</staves>
    <clef number="1">
      <sign>F</sign>
      <line>4</line>
    </clef>
    <clef number="2">
      <sign>F</sign>
    <line>4</line>
    </clef>
    <note default-x="139">
      <rest/>
```

```
      <duration>4</duration>
      <voice>1</voice>
      <staff>1</staff>
    </note>
    <note default-x="164">
      <pitch>
        <step>E</step>
        <octave>3</octave>
      </pitch>
      <duration>4</duration>
      <voice>1</voice>
      <staff>1</staff>
    </note>
  [...]
    <note default-x="289">
      <pitch>
        <step>C</step>
        <alter>1</alter>
        <octave>3</octave>
      </pitch>
      <duration>4</duration>
      <voice>1</voice>
      <staff>2</staff>
    </note>
    <note default-x="313">
      <pitch>
        <step>A</step>
        <octave>2</octave>
      </pitch>
      <duration>4</duration>
      <voice>1</voice>
      <type>eighth</type>
    </note>
    <backup>
      <duration>32</duration>
    </backup>
    <note default-x="139">
      <pitch>
        <step>A</step>
        <octave>1</octave>
      </pitch>
      <duration>16</duration>
      <voice>3</voice>
      <staff>2</staff>
    </note>
    <note default-x="139">
      <chord/>
      <pitch>
        <step>A</step>
```

```
        <octave>2</octave>
      </pitch>
      <duration>16</duration>
      <voice>3</voice>
      <staff>2</staff>
    </note>
    <note default-x="239">
      <rest/>
      <duration>8</duration>
      <voice>3</voice>
      <staff>2</staff>
    </note>
    <forward>
      <duration>8</duration>
      <voice>3</voice>
      <staff>2</staff>
    </forward>
  </measure>
</part>
```

Lo standard MusicXML prevede la presenza di una serie di elementi/attributi aggiuntivi, non mostrati per brevità negli esempi citati finora, che premettono di descrivere in maniera completa le caratteristiche necessarie per una corretta rappresentazione grafica della partitura (direzione gambo note, composizione di tuple innestate, legature di portamento etc.).

2.5 IEEE 1599

IEEE 1599 nasce in risposta all'esigenza di avere un framework in grado di rappresentare l'informazione musicale in maniera coerente e completa [4]. Il formato, basato su XML, è stato sviluppato principalmente presso il Laboratorio di Informatica Musicale (LIM) dell'Università degli studi di Milano, rispettando le linee guida IEEE P1599 (Recommended Practice Dealing With Applications and Representations of Symbolic Music Information Using the XML Language)[10]. Il punto di forza dello standard è quello di riuscire a codificare le informazioni afferenti alle diverse tipologie di rappresentazione dell'informazione musicale: la struttura di base di un file in formato IEEE 1599 è costituita dai sei layers, precedentemente descritti, utilizzabili per codificare le caratteristiche peculiari di un brano. Ogni layer può essere utilizzato per descrivere istanze multiple di un certo tipo di rappresentazione: ad esempio lo stesso file IEEE1599 potrà descrivere più esecuzioni audio/video di un determinato file musicale (layer Audio) e/o più trascrizioni della partitura del brano (layer Notational).

[10] https://standards.ieee.org/standard/1599-2008.html

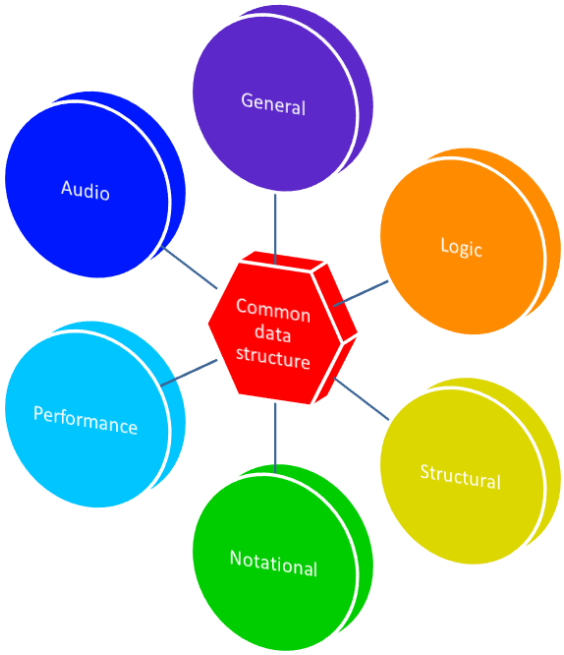


Figura 5: Tipologia di collegamento tra layer a stella

Il seguente frammento XML mostra l'elemento radice *ieee1599* e i sotto elementi che descrivono i vari livelli:

Listing 2.9: elementi figli di ieee1599

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "http://standards.ieee.org/downloads
   /1599/1599-2008/ieee1599.dtd">
<ieee1599 version="1.0">
  <general>[...]</general>
  <logic>[...]</logic>
  <structural>[...]</structural>
  <notational>[...]</notational>
  <performance>[...]</performance>
  <audio>[...]</audio>
</ieee1599>
```

Come già anticipato nella sezione 2.2, e rappresentato in Figura 5, IEEE 1599 adotta una topologia a stella per collegare i layer: il centro stella, ossia la struttura dati comune che permette l'interconnessione e la sincronizzazione tra i vari layer, è denominata *spine* ed è contenuta, a livello architetturale, all'interno del layer Logic ad affiancare l'alberatura dei Logical Object Symbols (LOS), come illustrato in Figura 6 (in [12] vengono confrontate le diverse topologie di connettività tra layer).

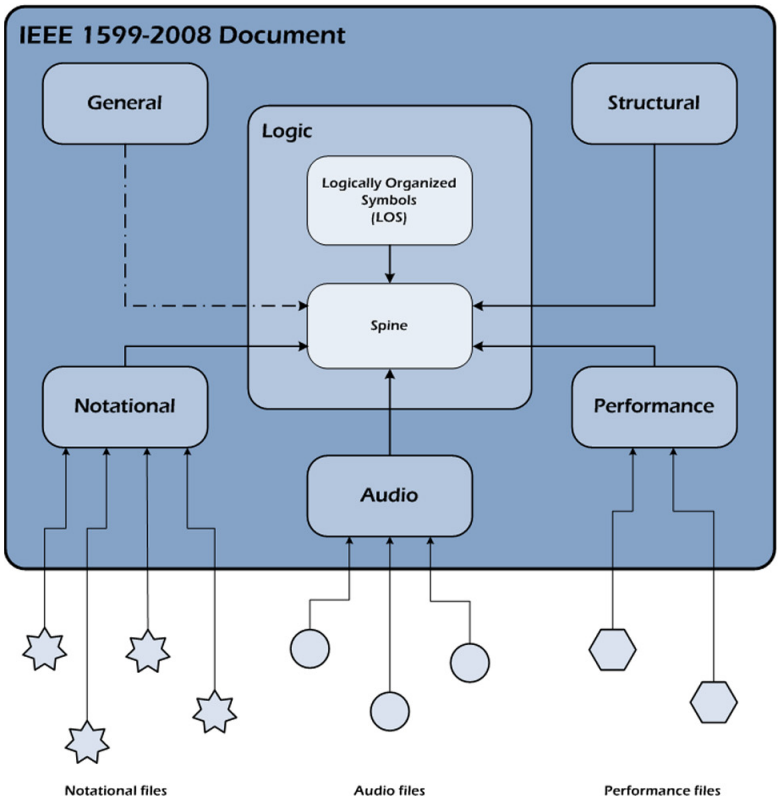


Figura 6: Architettura IEEE 1599

Nelle seguenti sottosezioni verranno descritti i principiali elementi che permettono di codificare le informazioni di pertinenza dei vari layer: per una descrizione di dettaglio sull'architettura del formato IEEE 1599 e alcuni casi pratici di utilizzo si rimanda a [13].

2.5.1 General

Il layer General fornisce una descrizione d'insieme dell'opera musicale, sintetizzando le informazioni comuni ai vari livelli: per questo motivo è l'unico strato a non dipendere strettamente dallo spine. Questo layer deve essere obbligatoriamente presente in un file IEEE 1599. Utilizzando gli elementi figli del nodo *description* è possibile classificare informazioni relative al brano descritto (*main_title*, *number*), all'opera principale alla quale afferisce il pezzo in analisi (*work_title*, *work_number*), agli autori (*author* name e type), etc. A questo livello è possibile inserire riferimenti a file esterni (*related_files*) che sono in relazione con l'opera nonostante non ne descrivano semanticamente il contenuto (ad esempio fotografie di esibizioni,

scenografie, immagini di copertina, ritratto dell'autore etc.). Si riporta a titolo di esempio il seguente frammento XML:

Listing 2.10: layer General

```
<general>
  <description>
    <main_title>Il mio ben quando verra'</main_title>
    <author type="composer">Paisiello, Giovanni</author>
    <author type="librettist">Carpani, Giuseppe</author>
    <work_title>Nina, ossia la pazza per amore</work_title>
  </description>
  <related_files>
    <related_file file_name="related_files/muti.jpg"
     file_format="image_jpeg" encoding_format="image_jpeg"
     spine_start_ref="nina_001_01" spine_end_ref="nina_003_01"
     description="Il Maestro Riccardo Muti"/>
    <related_file file_name="related_files/8700.jpg"
     file_format="image_jpeg" encoding_format="image_jpeg"
     spine_start_ref="nina_003_01" spine_end_ref="nina_005_01"
     description="Immagine di scena"/>
    <related_file file_name="related_filesantonacci.jpg"
     file_format="image_jpeg" encoding_format="image_jpeg"
     spine_start_ref="nina_005_01" spine_end_ref="nina_007_01"
     description="Caterina Antonacci"/>
    <related_file file_name="related_files/8699.jpg"
     file_format="image_jpeg" encoding_format="image_jpeg"
     spine_start_ref="nina_007_01" spine_end_ref="nina_009_01"
     description="Immagine di scena"/>
  </related_files>
</general>
```

2.5.2 Logic

Il layer Logic è deputato alla descrizione della componente simbolica della musica. Come nel caso del General, la presenza di questo livello è obbligatoria in IEEE 1599. L'elemento *logic* contiene:

- la struttura dati comune (*spine*) per la localizzazione e la sincronizzazione spazio-temporale di eventi;
- la descrizione degli elementi della partitura dal punto di vista simbolico (*los*);
- informazioni relative alla rappresentazione grafica generica dei simboli (*layout*).

Spine Tra gli elementi descrivibili con il layer Logic, lo spine riveste la maggiore rilevanza: per questo motivo la sua presenza è obbligatoria in un file IEEE 1559. È costituito da un insieme ordinato di eventi musicali, ciascuno dei quali viene descritto con un identificativo univoco al fine di poter essere richiamato senza ambiguità dagli altri layer. Il concetto di evento musicale è volutamente astratto e lasciato alla libera interpretazione dell'autore del file: questo rende lo standard particolarmente flessibile ed adattabile alle esigenze peculiari di ogni caso (ad esempio per codificare performance teatrali, come descritto in [14]). Nel caso più frequente, che consiste nella codifica di brani di musica occidentale, lo spine viene tipicamente popolato con eventi corrispondenti ad elementi simbolici delle partitura (note, pause, chiavi, alterazioni in chiave, indicazioni del tempo di misura, etc.). Ogni evento, oltre ad avere un id univoco, può presentare gli attributi *timing* e *hpos*. Il primo attributo, *timing*, è espresso in VTU (Virtual Time Units)[11] e viene utilizzato per descrivere la progressione degli eventi: rappresenta il delta temporale rispetto all'evento precedente; note che vengono eseguite nello stesso istante temporale avranno timing 0, indipendentemente dalla parte/voce di appartenenza. L'attributo *hpos* è espresso in VPX (Virtual PiXels)[12] e serve a descrivere la distribuzione spaziale degli eventi: si ottiene una vista astratta della partitura che, a differenza delle eventuali rappresentazioni concrete codificate con il layer Notational, non presenta limiti dimensionali sugli assi; gli eventi di una determinata parte, ad esempio, possono essere rappresentati su una singola linea.

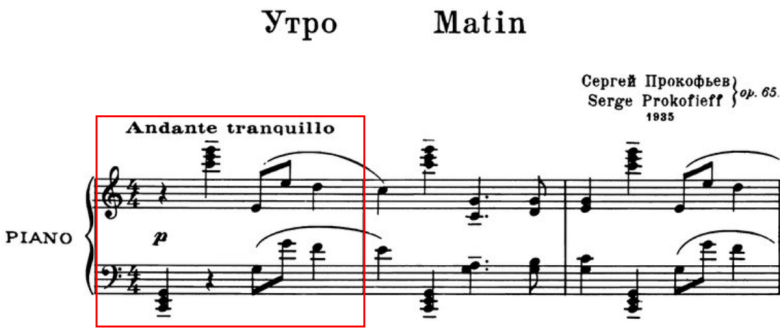


Figura 7: Le prime battute di Matin (Prokof'ev, Sergej Sergeevič)

Il seguente frammento XML descrive gli eventi ritenuti significativi all'interno delle prime due battute del brano *Matin* di Sergej Sergeevič Prokof'ev, evidenziate in rosso in Figura 7.

[11] i valori in VTU vengono espressi con numeri interi, solitamente multipli dell'evento con la minora durata; è ammesso il valore null quando la durata non è quantificabile

[12] i valori in VPX seguono le medesime convenzioni di quelli in VTU: una nota che dura il doppio di un'altra avrà il doppio del valore della prima

Listing 2.11: spine

```
<logic>
  <spine>
    <event id="clef_staff1_meas1_480" timing="0" hpos="0"/>
    <event id="keysig_staff1_meas1" timing="0" hpos="0"/>
    <event id="timesig_staff1_meas1" timing="0" hpos="0"/>
    <event id="clef_staff2_meas1_0" timing="0" hpos="0"/>
    <event id="keysig_staff2_meas1" timing="0" hpos="0"/>
    <event id="timesig_staff2_meas1" timing="0" hpos="0"/>
    <event id="piano1_meas1_voice1_ev1" timing="0" hpos="0"/>
    <event id="piano1_meas1_voice3_ev1" timing="0" hpos="0"/>
    <event id="piano1_meas1_voice1_ev2" timing="480" hpos="480"/>
    <event id="piano1_meas1_voice3_ev2" timing="0" hpos="0"/>
    <event id="piano1_meas1_voice1_ev3" timing="480" hpos="480"/>
    <event id="piano1_meas1_voice3_ev3" timing="0" hpos="0"/>
    <event id="piano1_meas1_voice1_ev4" timing="240" hpos="240"/>
    <event id="piano1_meas1_voice3_ev4" timing="0" hpos="0"/>
    <event id="piano1_meas1_voice1_ev5" timing="240" hpos="240"/>
    <event id="piano1_meas1_voice3_ev5" timing="0" hpos="0"/>
    [...]
```

Logical Organized Symbols L'elemento *los* viene utilizzato per la descrizione simbolica degli eventi che si verificano nella partitura. Gli elementi codificabili sono i seguenti:

- variazioni agogiche (*agogics*);
- indicazioni testuali generiche, come quelle relative alle articolazioni musicali (*text_field*);
- tempo di metronomo (*metronomic_indication*);
- elenco dei pentagrammi (*staff_list*);
- parti musicali (*part*);
- simboli di legatura, arpeggio, ripetizione, coda, etc. (*horizontal_symbols*);
- simboli di ornamento/abbellimento (*ornaments* come acciaccature, appoggiature, mordenti etc.);
- testo cantato del brano suddiviso in sillabe (*lyrics*).

Quando il LOS è presente, deve contenere obbligatoriamente almeno gli elementi *staff_list* e *part*. La (*staff_list*) è composta da una serie di elementi di tipo *staff*,

identificabili in maniera univoca, che descrivono i pentagrammi e le loro caratteristiche (tipologia di chiave, alterazioni in chiave, tempo di metronomo, eventuale descrizione con intavolatura etc.).

Listing 2.12: staff_list

```
<los>
  <staff_list>
    <staff id="staff1">
      <clef event_ref="clef_staff1_meas1_480" shape="G" staff_step
         ="2"/>
      [...]
      <key_signature event_ref="keysig_staff1_meas1">
        <sharp_num number="0"/>
      </key_signature>
      <time_signature event_ref="timesig_staff1_meas1">
        <time_indication num="4" den="4" vtu_amount="1920"/>
      </time_signature>
    </staff>
    <staff id="staff2">
      <clef event_ref="clef_staff2_meas1_0" shape="F" staff_step="
         6"/>
      [...]
      <key_signature event_ref="keysig_staff2_meas1">
        <sharp_num number="0"/>
      </key_signature>
      <time_signature event_ref="timesig_staff2_meas1">
        <time_indication num="4" den="4" vtu_amount="1920"/>
      </time_signature>
    </staff>
  </staff_list>
  [...]
```

Ciascuna parte (*part*) del brano, che viene descritta mediante un id univoco ed una serie di attributi opzionali (trasposizioni, numero di esecutori etc.), contiene l'elenco delle rispettive voci (*voice_list*).

Gli eventi simbolici appartengono ad una voce in una determinata misura afferente ad una specifica parte: ogni *part* è quindi composta da una o più misure (*measure*) che, per ogni elemento *voice* contenuto, permettono di codificare le occorrenze dei diversi tipi di eventi (*chord*, *rest*, *tablature_symbol*, *gregorian_symbol*).

La durata degli accordi (*chord*) e della pause (*rest*) viene espressa in notazione frazionaria mediante l'elemento *duration*; l'eventuale presenza e numerosità di punti di valore viene codificata con l'elemento *augmentation_dots*. I gruppi irregolari vengono rappresentati con il tag *tuplet_ratio* che codifica, mediante una struttura ad hoc, il rapporto tra la quantità di elementi appartenenti al gruppo irregolare e quella degli elementi che normalmente avrebbero posto all'interno della misura. Un elemento *tuplet_ratio* può contenere altri elementi del medesimo tipo: questa

soluzione permette di rappresentare gruppi irregolari complessi (come potrebbe essere una quintina eseguita sul secondo movimento di una terzina di quarti).

Ogni accordo contiene una o più note musicali (*notehead*) per le quali si codifica l'altezza con l'elemento *pitch* composto da:

- *step* - descrive in notazione anglosassone il nome della nota;
- *octave* - descrive l'ottava di appartenenza, per convenzione l'ottava centrale è la quinta;
- *actual_accidental* - descrive l'eventuale alterazione applicata alla nota (sia indicata esplicitamente sia derivante da alterazione presente su nota simile precedente o in chiave).

Lo standard prevede l'esistenza di ulteriori elementi e attributi, che non si riportano in maniera esaustiva per brevità, i quali consentono di codificare altre caratteristiche riguardanti la rappresentazione simbolica delle note (diteggiatura, stile di rappresentazione, etc.): per una panoramica completa si può fare riferimento alla documentazione ufficiale IEEE 1599.

Il seguente frammento XML descrive, mediante gli elementi presentati finora, gli eventi simbolici che, per ogni voce della parte di pianoforte, occorrono nella prima misura del brano in Figura 7.

Listing 2.13: voice_list

```
<los>
  [...]
  <part id="piano1">
    <voice_list>
      <voice_item id="piano1_voice1" staff_ref="staff1"/>
      <voice_item id="piano1_voice3" staff_ref="staff2"/>
      <voice_item id="piano1_voice2" staff_ref="staff1"/>
    </voice_list>
    <measure number="1">
      <voice voice_item_ref="piano1_voice1">
        <rest event_ref="piano1_meas1_voice1_ev1">
          <duration num="1" den="4"/>
        </rest>
        <chord event_ref="piano1_meas1_voice1_ev2">
          <duration num="1" den="4"/>
          <notehead>
            <pitch step="C" octave="7" actual_accidental="natural"
               />
          </notehead>
          <notehead>
            <pitch step="E" octave="7" actual_accidental="natural"
               />
```

```
      </notehead>
      <notehead>
        <pitch step="G" octave="7" actual_accidental="natural"
           />
      </notehead>
    </chord>
    <chord event_ref="piano1_meas1_voice1_ev3">
      <duration num="1" den="8"/>
      <notehead>
        <pitch step="E" octave="5" actual_accidental="natural"
           />
      </notehead>
    </chord>
    <chord event_ref="piano1_meas1_voice1_ev4">
      <duration num="1" den="8"/>
      <notehead>
        <pitch step="E" octave="6" actual_accidental="natural"
           />
      </notehead>
    </chord>
    <chord event_ref="piano1_meas1_voice1_ev5">
      <duration num="1" den="4"/>
      <notehead>
        <pitch step="D" octave="6" actual_accidental="natural"
           />
      </notehead>
    </chord>
  </voice>
  <voice voice_item_ref="piano1_voice3">
    <chord event_ref="piano1_meas1_voice3_ev1">
      <duration num="1" den="4"/>
      <notehead>
        <pitch step="C" octave="3" actual_accidental="natural"
           />
      </notehead>
      <notehead>
        <pitch step="E" octave="3" actual_accidental="natural"
           />
      </notehead>
      <notehead>
        <pitch step="G" octave="3" actual_accidental="natural"
           />
      </notehead>
    </chord>
    <rest event_ref="piano1_meas1_voice3_ev2">
      <duration num="1" den="4"/>
    </rest>
    <chord event_ref="piano1_meas1_voice3_ev3">
      <duration num="1" den="8"/>
```

```
        <notehead>
          <pitch step="G" octave="4" actual_accidental="natural"
             />
        </notehead>
      </chord>
      <chord event_ref="piano1_meas1_voice3_ev4">
        <duration num="1" den="8"/>
        <notehead>
          <pitch step="G" octave="5" actual_accidental="natural"
             />
        </notehead>
      </chord>
      <chord event_ref="piano1_meas1_voice3_ev5">
        <duration num="1" den="4"/>
        <notehead>
          <pitch step="F" octave="5" actual_accidental="natural"
             />
        </notehead>
      </chord>
    </voice>
  </measure>
  [...]
```

Layout L'elemento *layout* viene utilizzato per descrivere una rappresentazione grafica standard delle informazioni simboliche: ogni layout è composto da un certo numero di pagine, a loro volte suddivise in sistemi di pentagrammi.

2.5.3 Structural

Il layer Structural viene utilizzato per la descrizione di strutture musicali, che potrebbero essere risultanti dall'aggregazione di eventi, e delle relative relazioni che intercorrono tra di esse all'interno di un brano musicale. Questi oggetti sono tipicamente frutto di analisi compositiva e/o musicologica. È altresì possibile codificare strutture musicali derivanti da trasformazioni di elementi preesistenti. L'elemento *structural* permette di descrivere le seguenti rappresentazioni:

- *chord_grid* - contenitore di accordi descritti con elementi *chord_name*, ognuno dei quali fa riferimento a un evento dello spine;
- *analysis* - permette di modellare strutture frutto di analisi musicale, rappresentate con elementi *segmentation* composti da segmenti (*segment*) che raggruppano eventi (*segment_event*) e features (*feature_object*) che decrivono relazioni interne ai segmenti stessi. Le relazioni tra segmenti diversi possono essere indicate con gli elementi *relationships* e *feature_object_relationship*;

- *petri_nets* - contenitore per istanze di reti di Petri (*petri_net*) rappresentate con file esterni (in formato PNML) e descritte come insiemi di posti (*place* che possono contenere link ai differenti *segment*) e transizioni (*transition* che possono contenere collegamenti a *feature_object_relationship*);

- *mir* - contenitore di modelli formali (*mir_model*) per i processi di musical information retrieval: ogni modello, che opzionalmente può fare riferimento ad un file esterno (in formato GraphXML), è composto da oggetti; *mir_object* e da *mir_morphism* con i link alle loro eventuali posizioni nel file esterno. Ogni *mir_object*, con le relative *mir_feature* opzionali, è composto da *mir_subobject*, ciascuno dei quali sarà associato a delle *mir_feature* e potrà essere descritto con le corrispondenze relative al file esterno e a determinati segmenti descritti mediante la struttura *analysis*.

2.5.4 Notational

Il layer Notational, con l'elemento *notational*, permette di descrivere la rappresentazione grafica della partitura mediante due tipi di elementi contenitore:

- *graphic_instance_group* che gestisce il mapping tra eventi dello spine e file di immagine rappresentanti la partitura (ad esempio JPEG, PNG, PDF etc.). Le informazioni relative ad un file (nome, formato file, formato encoding, etc.) vengono descritte con gli attributi dell'elemento *graphic_instance*. Ogni *graphic_instance* contiene elementi di tipo *graphic_event* che mettono in relazione un determinato identificativo univoco di evento con una porzione dell'immagine delimitata da coordinate bidimensionali;

- *notation_instance_group* che gestisce il mapping tra eventi dello spine e file binari di descrizione della partitura (formati NIFF, ENIGMA etc.). Le informazioni relative ad un file (nome, formato file, formato encoding, etc.) vengono descritte con gli attributi dell'elemento *notation_instance*. Ogni *notation_instance* contiene elementi di tipo *notation_event* che mettono in relazione un determinato identificativo univoco di evento con una porzione del file delimitata da riferimenti di inizio e fine (dipendenti dal tipo di file).

A titolo d'esempio si riporta il frammento XML del layer Notational IEEE 1599 che codifica il mapping tra l'immagine JPEG in Figura 8 e gli eventi delle prime cinque misure presenti nello spine del brano *Music for khomus*.

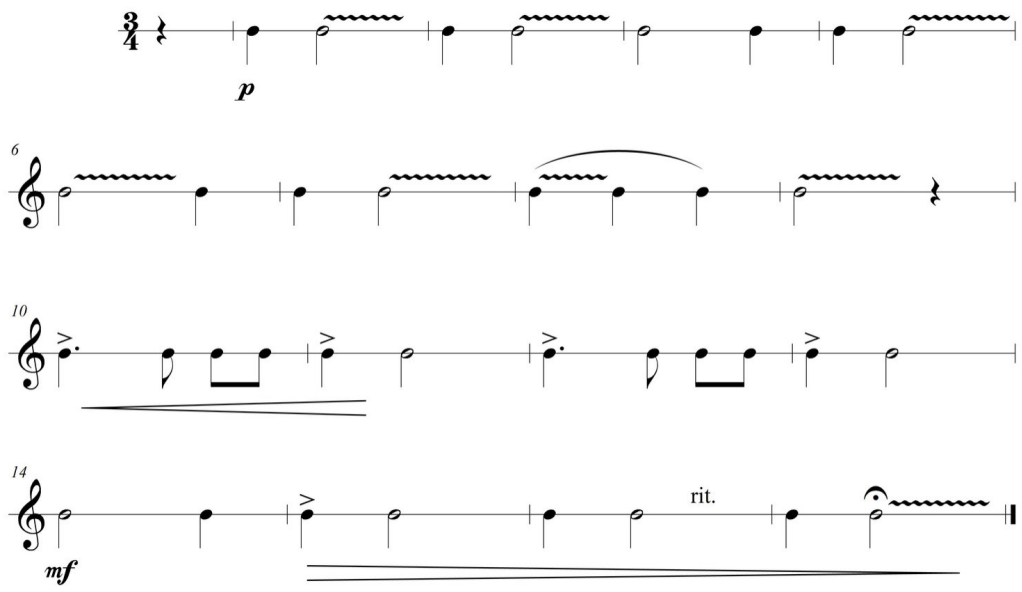


Figura 8: Partitura del brano Music for khomus

Listing 2.14: notational Khomus

```
<notational>
  <graphic_instance_group description="Finale transcription">
    <graphic_instance file_name="score_files/finale_transcription/
       khomus_finale_score.jpg" file_format="image_jpeg"
       encoding_format="image_jpeg" position_in_group="1"
       measurement_unit="pixels">
      <graphic_event event_ref="TimeSignature_part_1_1"
         upper_left_x="708" upper_left_y="252" lower_right_x="816"
          lower_right_y="458"/>
      <graphic_event event_ref="part_1_voice0_measure1_ev0"
         upper_left_x="858" upper_left_y="270" lower_right_x="938"
          lower_right_y="436"/>
      <graphic_event event_ref="part_1_voice0_measure2_ev0"
         upper_left_x="1246" upper_left_y="310" lower_right_x="
         1352" lower_right_y="526"/>
      <graphic_event event_ref="part_1_voice0_measure2_ev1"
         upper_left_x="1554" upper_left_y="310" lower_right_x="
         1660" lower_right_y="526"/>
      <graphic_event event_ref="part_1_voice0_measure3_ev0"
         upper_left_x="2112" upper_left_y="314" lower_right_x="
         2218" lower_right_y="530"/>
```

```
      <graphic_event event_ref="part_1_voice0_measure3_ev1"
         upper_left_x="2422" upper_left_y="320" lower_right_x="
         2528" lower_right_y="536"/>
      <graphic_event event_ref="part_1_voice0_measure4_ev0"
         upper_left_x="2978" upper_left_y="304" lower_right_x="
         3084" lower_right_y="520"/>
      <graphic_event event_ref="part_1_voice0_measure4_ev1"
         upper_left_x="3472" upper_left_y="306" lower_right_x="
         3578" lower_right_y="522"/>
      <graphic_event event_ref="part_1_voice0_measure5_ev0"
         upper_left_x="3850" upper_left_y="308" lower_right_x="
         3956" lower_right_y="524"/>
      <graphic_event event_ref="part_1_voice0_measure5_ev1"
         upper_left_x="4158" upper_left_y="312" lower_right_x="
         4264" lower_right_y="528"/>
      [...]
    </graphic_instance>
  </graphic_instance_group>
</notational>
```

2.5.5 Performance

Con l'elemento *performance* si possono rappresentare una o più occorrenze dei seguenti elementi:

- *midi_instance* - consente di fare riferimento ad un file esterno in formato MIDI, descrivendone i diritti di utilizzo (*rights*) e permettendo il mapping delle relative tracce e canali MIDI (*midi_mapping*) con parti e voci IEEE 1599. L'associazione avviene mediante attributi ad hoc e la descrizione di una o più sequenze (*midi_event_sequence*) di eventi (*midi_event*) o di messaggi di tipo System Exclusive (*sys_ex*);

- *csound_instance* - permette di collegare documenti esterni in formato CSound, tipicamente composti da un file che descrive la partitura (score .SCO) e da uno che descrive le parti strumentali (orchestra .ORC): l'elemento *csound_score* consente l'associazione con gli eventi dello spine e i corrispondenti eventi csound (*csound_spine_event*) codificati nel file score; l'elemento (*csound_orchestra*) permette il mapping (*csound_instrument_mapping*) tra le parti IEEE 1599 (*csound_spine_ref*) e quelle descritte nel file orchestra (*csound_part_ref*). Per entrambi gli elementi è possibile codificare informazioni relative ai diritti con l'elemento opzionale *rights*;

- *mpeg4_instance* - abilita il mapping con archivi MPEG4, tipicamente composti da un file che descrive la partitura (SASL - Structured Audio Score Language) e da uno che descrive le parti strumentali (SAOL - Structured Audio Orchestra Language). Analogamente a quanto già visto per la rappresentazione CSound, l'elemento *mpeg4_score* consente l'associazione con gli eventi dello spine e i corrispondenti eventi (*mpeg4_spine_event*) codificati nel file SASL; l'elemento (*csound_orchestra*) consente il mapping (*mpeg4_instrument_mapping*) tra le parti IEEE 1599 (*mpeg4_spine_ref*) e quelle descritte nel file orchestra (*mpeg4_part_ref*). Anche in questo caso è previsto l'uso opzionale dell'elemento *rights*.

Nel seguente frammento XML, estratto dalla codifica IEEE 1599 del brano *Gottes Macht und Vorsehung* di Ludwig van Beethoven, viene mostrato l'utilizzo del layer Performance per il mapping di un evento del layer Logic (una pausa) con il file MIDI esterno *beethoven_gottes_macht.mid*

Listing 2.15: performance MIDI

```
<logic>
  <spine>
    <event id="v1_e0" timing="0" hpos="6"/>
  </spine>
  <los>
    <part id="soprano">
      <measure number="1">
        <voice voice_item_ref="soprano_voice1">
          <rest event_ref="v1_e0" staff_ref="staff_1">
        </voice>
      </measure>
    </part>
  </los>
</logic>
<performance>
  <midi_instance file_name="beethoven_gottes_macht.mid" format="1"
     >
    <midi_mapping part_ref="soprano" track="1" channel="1">
      <midi_event_sequence division_type="timecode" division_value
         ="1024" measurement_unit="ticks">
        <midi_event event_ref="v1_e0" timing="0"/>
        [...]
      </midi_event_sequence>
    </midi_mapping>
  </midi_instance>
</performance>
```

2.5.6 Audio

L'elemento *audio* viene utilizzato per associare file contenenti materiale audio / video, in formato digitale, ad eventi codificati nello spine: tipicamente il mapping avviene specificando valori temporali assoluti. Il riferimento a un file esterno viene codificato con gli attributi dell'elemento *track* (*file_name*, *file_format*, *encoding_format*) che viene ripetuto tante volte quante sono le tracce da descrivere. Una traccia può essere dettagliata con i seguenti elementi opzionali:

- *track_general* - codifica i metadati relativi alla traccia audio (*recordings*, *genres*, *albums*, *performers*, *notes*, etc.);
- *track_indexing* - permette di descrivere regioni (*track_region*) ed eventi (*track_event*) della traccia che vengono associati agli eventi del layer Logic descritti nello spine. L'attributo *timing_type* descrive l'unità di misura con la quale avvengono i riferimenti temporali rispetto al tipo di file trattato (ad esempio *samples*, *time*, *seconds*, *time_frames*, *frames*, etc.).

Il seguente frammento XML esemplifica l'utilizzo del layer Audio per associare allo spine gli eventi del primo secondo di esecuzione di una registrazione audio in fomato mp3 del brano *Serie in 9/8* di Goffredo Haus.

Listing 2.16: audio MP3

```
<audio>
  <track file_name="audio_files/serie_9_8_sito.mp3" file_format="
     audio_mp3" encoding_format="audio_mp3">
    <track_general>
      <recordings>
        <recording date="1988" studio_name="LIM"/>
      </recordings>
      <performers>
        <performer name="Antonio J. Rodriguez Selles" type="
           computer music"/>
        <performer name="Goffredo Haus" type="computer music"/>
      </performers>
      <notes>Computer music master produced at LIM by Antonio J.
         Rodriguez Selles and Goffredo Haus (1988)</notes>
    </track_general>
    <track_indexing timing_type="seconds">
      <track_event event_ref="Flauto_1_voice0_measure1_ev0"
         start_time="0.5"/>
      <track_event event_ref="Marimba_2_voice0_measure1_ev0"
         start_time="0.5"/>
      <track_event event_ref="Violoncello_3_voice0_measure1_ev0"
         start_time="0.5"/>
      <track_event event_ref="Contrabbasso_4_voice0_measure1_ev0"
         start_time="0.5"/>
```

```
      <track_event event_ref="Percussioni_5_voice0_measure1_ev0"
         start_time="0.5"/>
      <track_event event_ref="Timpani_6_voice0_measure1_ev0"
         start_time="0.5"/>
      <track_event event_ref="TimeSignature_Flauto_1_1" start_time
         ="0.5"/>
      <track_event event_ref="TimeSignature_Marimba_2_1"
         start_time="0.5"/>
      <track_event event_ref="TimeSignature_Violoncello_3_1"
         start_time="0.5"/>
      <track_event event_ref="TimeSignature_Contrabbasso_4_1"
         start_time="0.5"/>
      <track_event event_ref="TimeSignature_Timpani_6_1"
         start_time="0.5"/>
      <track_event event_ref="TimeSignature_Percussioni_5_1"
         start_time="0.5"/>
      <track_event event_ref="KeySignature_Flauto_1_1" start_time=
         "0.5"/>
      <track_event event_ref="KeySignature_Marimba_2_1" start_time
         ="0.5"/>
      <track_event event_ref="KeySignature_Violoncello_3_1"
         start_time="0.5"/>
      <track_event event_ref="KeySignature_Contrabbasso_4_1"
         start_time="0.5"/>
      <track_event event_ref="KeySignature_Percussioni_5_1"
         start_time="0.5"/>
      <track_event event_ref="KeySignature_Timpani_6_1" start_time
         ="0.5"/>
      <track_event event_ref="Clef_Flauto_1_1" start_time="0.5"/>
      <track_event event_ref="Clef_Marimba_2_1" start_time="0.5"/>
      <track_event event_ref="Clef_Violoncello_3_1" start_time="
         0.5"/>
      <track_event event_ref="Clef_Contrabbasso_4_1" start_time="
         0.5"/>
      <track_event event_ref="Clef_Percussioni_5_1" start_time="
         0.5"/>
      <track_event event_ref="Clef_Timpani_6_1" start_time="0.5"/>
      <track_event event_ref="Violoncello_3_voice0_measure1_ev1"
         start_time="0.75"/>
      [...]
    </track_indexing>
  </track>
</audio>
```

2.6 Sviluppi recenti di IEEE 1599

Sin dalla nascita di IEEE 1599, il Laboratorio di Informatica Musicale (LIM) dell'Università degli Studi di Milano è stato il soggetto che ha maggiormente contribuito alla promozione dello standard, curando la redazione di articoli e contributi per la stampa specializzata[13] e lavorando in sinergia con soggetti nazionali ed internazionali. Sul sito Internet del LIM[14] è presente una ricca sezione di riepilogo delle dimostrazioni avvenute in contesti di primaria rilevanza, come quelle che si sono tenute nel corso del 2019 al *Museo Teatrale alla Scala* e al *Museo del Duomo* di Milano, al *Palazzo Bossi Bocchi* di Pavia e al *The Morgan Library & Museum* di New York. Anche lo sviluppo di software ad hoc è stato curato principalmente dallo staff del LIM, grazie anche alle esperienze pregresse maturate in diversi contesti (come quelli descritti in [3]). In tempi recenti sono state condotte alcune attività di analisi che hanno portato alla luce delle criticità legate allo standard (descritte in dettaglio in [7]), tra le principali si ricordano:

- la necessità di aderire a un formato proprietario per la rappresentazione dei simboli afferenti al layer Logic;
- possibili ambiguità nella definizione delle istanze di layer;
- la difficoltà nel descrivere alcuni tipi di composizioni formate da elementi collegati in maniera complessa;
- un supporto non completo alla gestione dei diritti digitali.

Per capitalizzare al meglio il lavoro svolto finora, è necessario risolvere queste criticità, elaborando strategie aggiuntive per la promozione dello standard, aumentando la disponibilità di strumenti di utilità e la mole di documenti musicali codificati in formato IEEE 1599. Con questo spirito, nel corso del 2019, presso l'Università degli Studi di Milano si è tenuto il *First International Workshop on Multilayer Music Representation and Processing (MMRP19)* che ha ospitato il gruppo di lavoro *IEEE WG_1599* riunito per avviare un processo di revisione dello standard.

[13] `https://ieee1599.lim.di.unimi.it/documentation_papers.php`
[14] `https://ieee1599.lim.di.unimi.it/practice_exhibitions.php`

2.6.1 Applicazioni

Tra i software principali sviluppati presso il LIM, liberamente utilizzabili, possiamo citare:

- un editor multi-piattaforma[15] per la modifica delle informazioni codificate nei layer di un file in formato IEEE 1599;
- plugins per l'esportazione di file IEEE 1599 da software di notazione musicale terzi (*Finale* e *MuseScore*[16]);
- un tool online per applicare modifiche, in modalità assistita, ai contenuti del layer Graphic[17];
- un player web multimediale per la fruizione di contenuti musicali in formato IEEE 1599 (descritto in dettaglio nel paragrafo seguente).

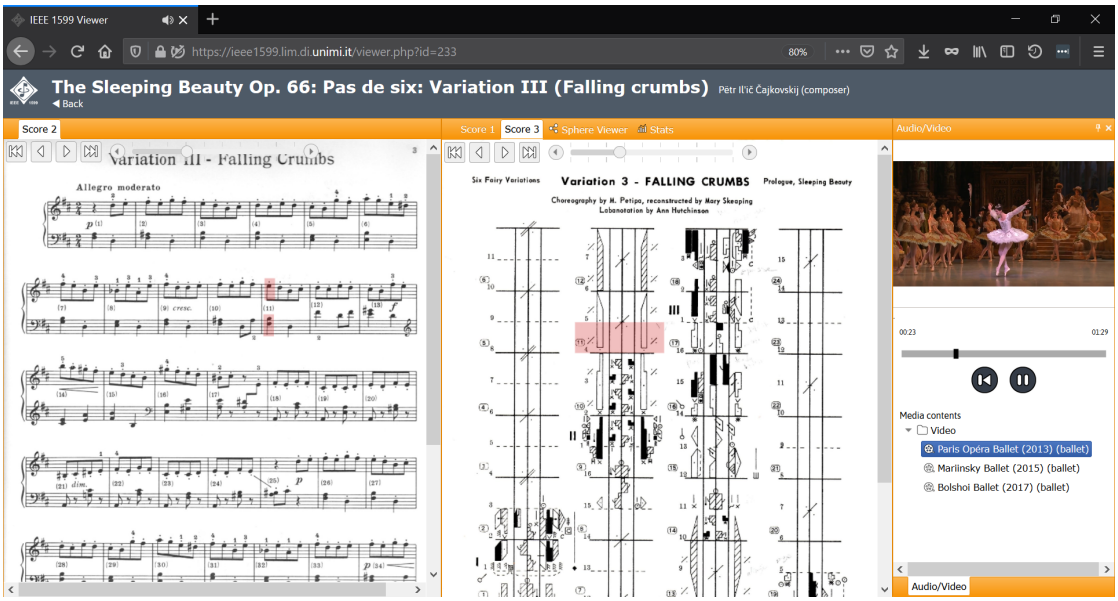


Figura 9: Player multimediale: graphic instances + video

[15] https://ieee1599.lim.di.unimi.it/practice_applications.php
[16] https://github.com/LIMUNIMI/musescorePlugins
[17] https://ieee1599.lim.di.unimi.it/utils_graphic_adjuster.php

Player multimediale L'applicativo consiste in uno strumento web pubblicato sul sito Internet del LIM: l'utente può selezionare uno dei brani in formato IEEE 1599 presenti nell'archivio musicale online (Figura 12) e fruirne i contenuti.

La soluzione infrastrutturale impiegata per l'erogazione del prodotto è composta da 2 elementi principali:

- db server (PostgreSQL) - archivio di documenti musicali IEEE 1599;
- web server (PHP server) - interfacce utente ed erogazione contenuti multimediali.

L'interfaccia del player è realizzata con una combinazione di PHP, codice JavaScript e utilizzo delle librerie jQuery. Il codice JS per il parsing di file IEEE 1599 (*IEEE1599.js*) fa parte di una suite di file denominata *ieee1599-framework* che viene eseguita, client-side, dal browser dell'utente che accede al sito.

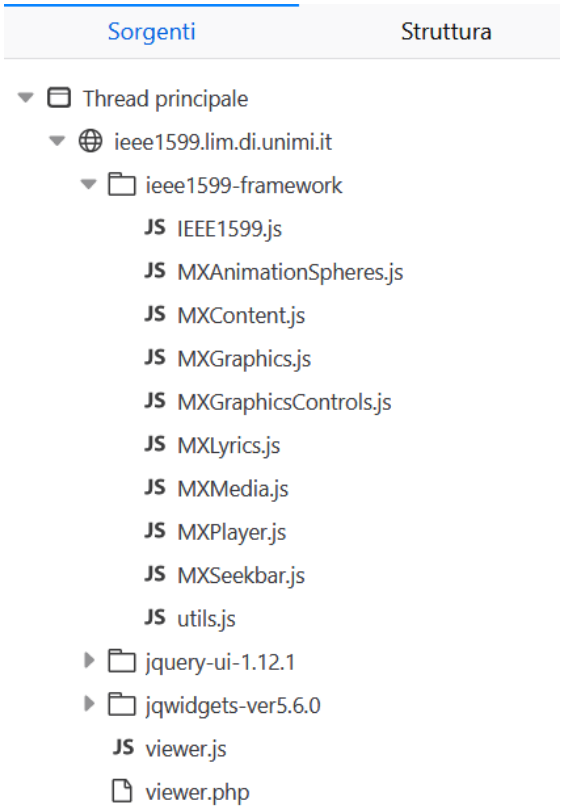


Figura 10: Player multimediale: alberatura del codice sorgente

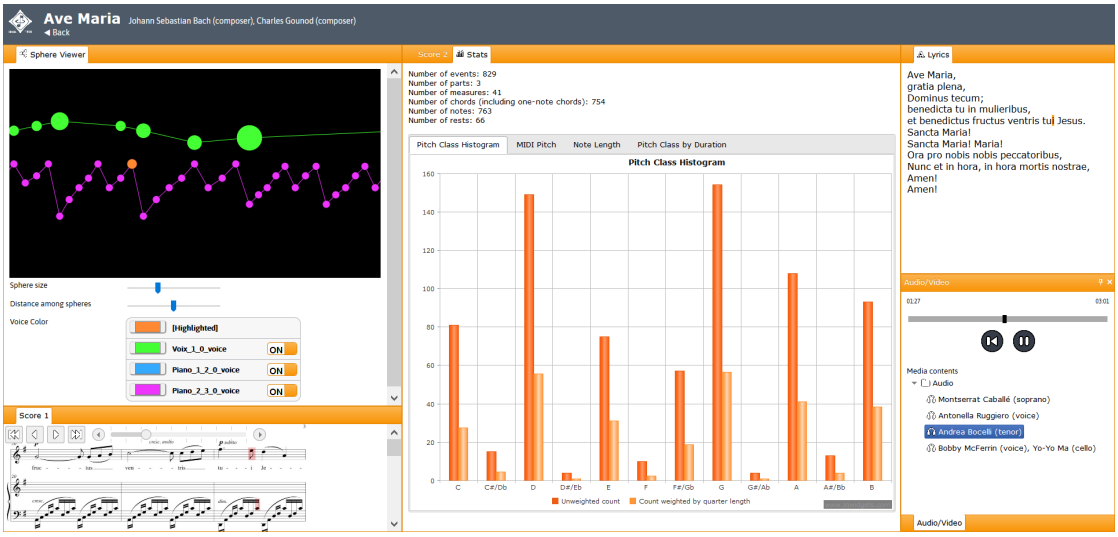


Figura 11: Player multimediale: sphere view, partitura, statistics, lyrics e audio

La pagina web che ospita il player viene popolata dinamicamente a seconda delle informazioni presenti nei layer del file 1599 processato:

- *General* - i metadati del brano vengono utilizzati per popolare l'header della pagina (titolo e numero brano, titolo opera principale, numerazione della composizione, compositore, tipo compositore etc.);
- *Logic* - gli eventi codificati nello *spine* vengono utilizzati per la sincronizzazione delle rappresentazioni. Le informazioni codificate nel LOS vengono utilizzate per il popolamento delle viste:

  Lyrics - contiene il testo del brano, la sillaba in esecuzione viene evidenziata da un cursore colorato;

  Statistics - contiene i risultati dell'analisi automatica sull'occorrenza di elementi (note, misure, parti) e relativi grafici;

  Sphere Viewer - offre una rappresentazione grafica alternativa del brano musicale: ogni nota viene raffigurata come una sfera che varia per colore (voce di appartenenza), dimensione (durata della nota) e posizione (altezza). Le sfere che indicano note in esecuzione sono rappresentate con un colore ad hoc;

- *Structural* - le eventuali rappresentazioni mediante reti di Petri vengono visualizzate in tab ad hoc;
- *Notational* - ogni gruppo di istanze grafiche viene renderizzato in una scheda dedicata. Sono presenti controlli che permettono di regolare lo zoom del

contenuto grafico correntemente visualizzato (istanza grafica corrente) e per lo spostamento tra contenuti grafici dello stesso gruppo (passare all'istanza grafica precedente/successiva, tornare alla prima o selezionare l'ultima, etc.). Nell'uso canonico vengono rappresentate partiture composte da pagine, ma questa non è l'unica applicazione possibile dato che lo standard è molto flessibile (ad esempio, come mostrato in Figura 9, può essere utilizzato per rappresentare i movimenti umani in formato *Labanotation* [15]). Durante l'esecuzione dei contenuti associati ad un determinato evento, le relative porzioni di immagine vengono evidenziate dinamicamente con un cursore colorato interattivo.

- *Performance* - attualmente non utilizzato;
- *Audio* - i contenuti audio/video vengono rappresentati in un pannello ad hoc: le tracce, mappate nel file, vengono suddivise per tipologia ed elencate in un menù ad albero. È possibile, in ogni istante, passare in tempo reale da una traccia all'altra semplicemente cliccando su quella di interesse. L'esecuzione può essere controllata con pulsanti (*Play/Pause*,*Rewind*) e barre a cursore.

La disposizione delle viste nella pagina può essere modificata a piacere dall'utilizzatore con una semplice operazione di *drag and drop*.

La riproduzione dei documenti multimediali avviene in maniera sincronizzata: l'utente, operando su una qualsiasi delle viste, può governare in tempo reale il flusso di riproduzione, passando da un tipo di rappresentazione all'altro. Oltre ad interagire con i file audio/video mediante i controlli del pannello ad hoc, è possibile, ad esempio, avviare la riproduzione a partire da una determinata nota della partitura (cliccando sulla porzione di immagine associata a quell'evento) o da un certa sillaba del testo cantato (cliccando sul testo nella vista Lyrics).

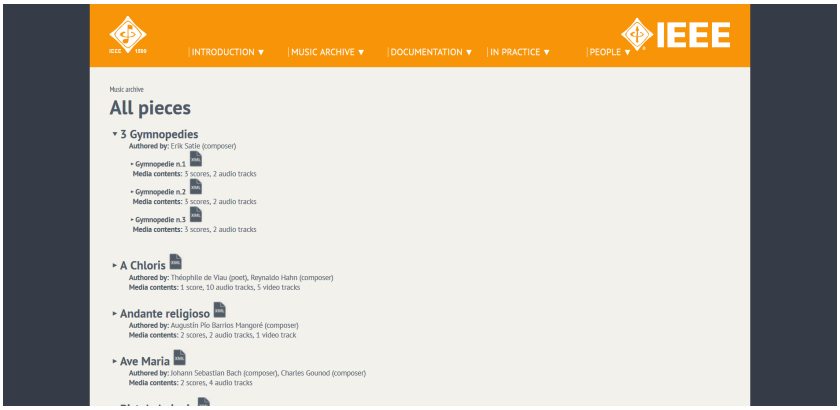


Figura 12: IEEE 1599 Music Library

Capitolo 3

Modellazione di informazione musicale simbolica

Come anticipato nei capitoli precedenti, lo standard IEEE 1599 sarà oggetto, nel corso del prossimi anni, di un'importante attività di revisione ad opera del gruppo di lavoro ricostituitosi nel 2019. Tra le tematiche che verranno affrontate nel corso del processo, con ogni probabilità, non mancherà la rimodulazione del layer Logic: come già auspicato in [7] e rappresentato in Figura 13, verosimilmente la struttura di questo strato verrà rivista per permettere la rappresentazione, nel LOS, di informazioni provenienti da formati terzi.

Il lavoro di tesi si sviluppa in questo contesto: viene proposta una soluzione che prevede la modellazione di una struttura dati generica (*MusicDocument*) per la codifica di informazioni relative ad un nucleo di elementi simbolici musicali; questo modello può essere liberamente esteso, per la rappresentazione di ulteriori elementi informativi, e contestualizzato per l'utilizzo in scenari concreti (a seconda di come le informazioni sono memorizzate nei formati di provenienza).

Questo capitolo introduce il lavoro svolto, partendo dalla specifica dei requisiti, illustrando le tecnologie adottate e fornendo una panoramica delle strutture astratte del modello base. In seguito vengono descritte le funzioni implementate per istanziare oggetti di strutture specifiche, derivate da quelle di base e presentate in dettaglio nel Capitolo 4, e i meccanismi di controllo introdotti per verificare l'integrità degli elementi ottenuti.

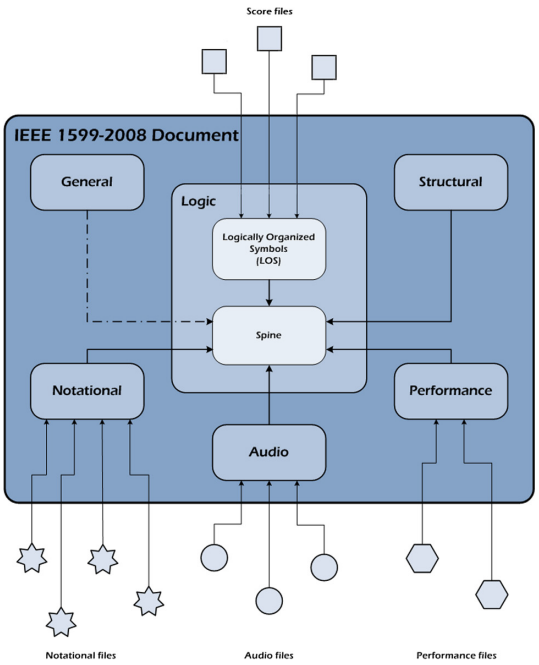


Figura 13: IEEE 1599: rappresentazione dell'evoluzione del layer Logic

3.1 Requisiti

L'idea del progetto è nata durante una sessione di brainstorming, riguardante le possibili evoluzioni di IEEE 1599, avvenuta ad inizio 2019 con alcuni membri dello staff del LIM. In un primo tempo sono stati definiti i marco-requisiti per l'applicazione:

- analisi di alcuni formati di rappresentazione dell'informazione musicale simbolica;
- progettazione di strutture dati comuni per la rappresentazione delle informazioni;
- implementazione di alcune specializzazioni d'esempio delle strutture dati comuni per rappresentare i contenuti dei file analizzati, definendo delle procedure di parsing ad hoc per il popolamento automatico.

Durante la fase di analisi dei requisiti, che ha richiesto alcune settimane, ci sono state nuove occasioni di confronto in cui sono stati rifiniti i vincoli implementativi:

- sviluppo di codice per il web (JavaScript);

- compatibilità con il codice preesistente (*ieee1599-framework*) per futura integrazione con il Player Multimediale.

L'approfondimento delle peculiarità dei formati di rappresentazione dell'informazione musicale simbolica è avvenuto ricorrendo alla lettura di documentazione tecnica, pubblicata sui portali Internet di riferimento dei produttori (manuali utente e per sviluppatori), e di diverse pubblicazioni di riferimento (riportate in bibliografia). Gli elementi salienti emersi da questa prima fase di analisi sono stati descritti nel Capitolo 2. Per lo studio del linguaggio JavaScript ci si è avvalsi della ricca documentazione disponibile gratuitamente in rete[1]. Una volta acquisita padronanza di questo linguaggio, ci si è concentrati sull'analisi del codice sorgente del Player Multimediale IEEE 1599 (il cui funzionamento è descritto nella sezione 2.6.1): si è appurato che, se gli sviluppatori del LIM avessero introdotto delle modifiche minime al codice originale per rettificare alcuni utilizzi di funzionalità deprecate (ad esempio chiamate XHR sincrone), sarebbe stato possibile garantire la compatibilità delle nuove strutture dati con l'applicativo preesistente (l'alternativa, meno consigliabile, sarebbe stata quella di scrivere codice retrocompatibile destinato a diventare presto inutilizzabile). Ci si è quindi concentrati sulla definizione delle strutture dati di base, modellandole con il duplice scopo di garantire la consistenza semantica dell'informazione simbolica e la compatibilità con gli utilizzi previsti dal Player Multimediale. Nella Sezione 3.3 vengono illustrate queste strutture con l'ausilio di diagrammi UML ed estratti del codice sorgente.

3.2 Tecnologie utilizzate

Per lo sviluppo della tesi ci si è avvalsi delle tecnologie di seguito descritte.

Linguaggi

JavaScript JavaScript, spesso abbreviato in JS, è un linguaggio di programmazione storicamente utilizzato per l'implementazione di script run-time[2] che controllano il comportamento di pagine web; di recente è stato impiegato anche in contesti che non contemplano l'utilizzo del browser (ad esempio nei PDF generati da AdobeAcrobat, o nell'ambiente Node.js che viene eseguito server-side, etc.). JS implementa lo standard ISO ECMA Script: è un linguaggio ad oggetti dinamico e multi-paradigma, con esecuzione single-thread, supporto alle first class-functions

[1] particolarmente utile il portale `https://developer.mozilla.org/en-US/docs/Web/JavaScript`

[2] compilazione Just In Time (JIT) che avviene cioè durante l'esecuzione del codice nel browser

e che si presta a differenti stili di programmazione (orientata agli oggetti, funzionale, imperativa). A differenza di altri linguaggi ad oggetti (ad esempio Java o C++), JavaScript implementa un meccanismo di ereditarietà basato su prototipi: gli oggetti possono essere istanziati direttamente o con costruttori (funzioni) che ereditano le proprietà e i metodi da oggetti preesistenti. Il codice JavaScript sviluppato per l'implementazione dei modelli proposti viene eseguito client-side nel browser dell'utente.

HTML HyperText Markup Language è un linguaggio dichiarativo la cui sintassi è stabilita dal World Wide Web Consortium (W3C): attraverso l'utilizzo di contrassegni predefiniti (tag) permette di definire la struttura degli elementi all'interno di una pagina web. Rappresenta uno dei principali linguaggi per la creazione di pagine Internet ed è spesso utilizzato in sinergia con CSS (Cascading Style Sheets, per la gestione della formattazione/impaginazione), XML (eXtensible Markup Language, per la memorizzazione/trasporto dei dati) e JS (JavaScript, per programmare il comportamento della pagina / gestione eventi). Nell'ambito del progetto di tesi, HTML è stato impiegato per creare:

- una pagina dimostrativa delle funzionalità implementate;
- una pagina di test che, sfruttando il framework qUnit, permette di verificare che gli oggetti generati siano ben formati e rispettino le convenzioni definite.

XML eXtensible Markup Language è un metalinguaggio autodescrittivo utilizzato per definire documenti / strutture dati mediante l'utilizzo di marcatori (tag) liberamente definibili: le informazioni risultano facilmente memorizzabili, trasportabili e interpretabili (sia da procedure automatizzate sia da esseri umani). Utilizzando l'XML DOM (Document Object Model) è possibile accedere in maniera dinamica alla struttura di un documento e sfogliarlo con una vista ad albero: questa interfaccia è stata sfruttata intensivamente per la scrittura del codice JavaScript che si occupa di processare le informazioni codificate nei file in formato IEEE 1599 e MusicXML.

LaTeX LaTeXè uno linguaggio di markup per la scrittura di documenti che si basa sull'utilizzo di contrassegni per definire struttura, impaginazione di elementi, formato del testo, riferimenti, citazioni etc. Il file di output, tipicamente in formato PDF, viene generato da un engine che si occupa della componente tipografica digitale. Il presente documento è stato scritto con l'editor TeXstudio e processato dal tool MiKTeX.

Editors

VisualStudioCode Visual Studio Code è un editor di codice sorgente multipiattaforma sviluppato da Microsoft. Include diverse funzionalità utili alla scrittura (autocomplete, syntax highlighting), alla gestione delle versioni (supporto a git) ed è estendibile mediante l'installazione di plugin: ad esempio è stato installato un plugin che ha svolto la funzionalità di webserver per l'erogazione dei contenuti sviluppati, con esecuzione del refresh automatico, a fronte di modifica del codice, delle pagine fruite via browser. Questo editor è stato utilizzato per tutte le attività di scrittura del codice sorgente applicativo (JavaScript e HTML).

TexStudio TeXstudio è un ambiente di scrittura integrato per la creazione di documenti LaTeX. Questo editor facilita la scrittura grazie alla presenza di strumenti di supporto (evidenziazione della sintassi, visualizzatore integrato, controllo dei riferimenti, procedure guidate per l'inserimento di immagini e tabelle etc.). È stato utilizzato per la scrittura del presente documento.

VisualParadigm Online Diagrams VisualParadigm Online Diagrams è un applicativo web[3] utilizzabile per la generazione di diversi tipi di diagramma. Ai fini della tesi è stato utilizzato per generare i diagrammi delle classi UML, a supporto della documentazione che descrive le strutture dati, e il flowchart di sintesi degli step compiuti dall'algoritmo di parsing dei file Humdrum **kern.

Framework di test

QUnit QUnit è progettato per eseguire unit testing jQuery e può anche essere utilizzato per verificare codice JavaScript generico. Il framework, liberamente scaricabile dal sito del produttore[4] viene fornito con una pagina HTML di esempio. Il funzionamento base consiste nella verifica di diversi tipi di assertion applicate alle unità di codice. Ai fini della tesi è stato utilizzato per la scrittura dei casi di test per la verifica della correttezza degli oggetti restituiti dalle funzioni JavaScript.

Source Code Management

Git Git è il più diffuso strumento opensource di gestione del codice sorgente ed è supportato nativamente da molti editor (Visual Studio Code non fa eccezione). Nell'ambito del progetto è stato utilizzato per tracciare le modifiche occorse durante lo sviluppo del codice sorgente.

[3] `https://online.visual-paradigm.com/app/diagrams/`
[4] `https://qunitjs.com/`

3.3 Prototipi strutture dati

Ispirandosi ai formati analizzati e in ottica di mantenere la compatibilità con il Player Multimediale, in fase di progettazione si è prevista la creazione di un insieme di strutture dati generiche/astratte destinate alla codifica delle caratteristiche peculiari di un documento musicale: queste strutture andranno concretizzate, a seconda del contesto di utilizzo, con strutture dati specializzate che ne erediteranno le proprietà. L'output della fase di progettazione è rappresentato in Figura 14.

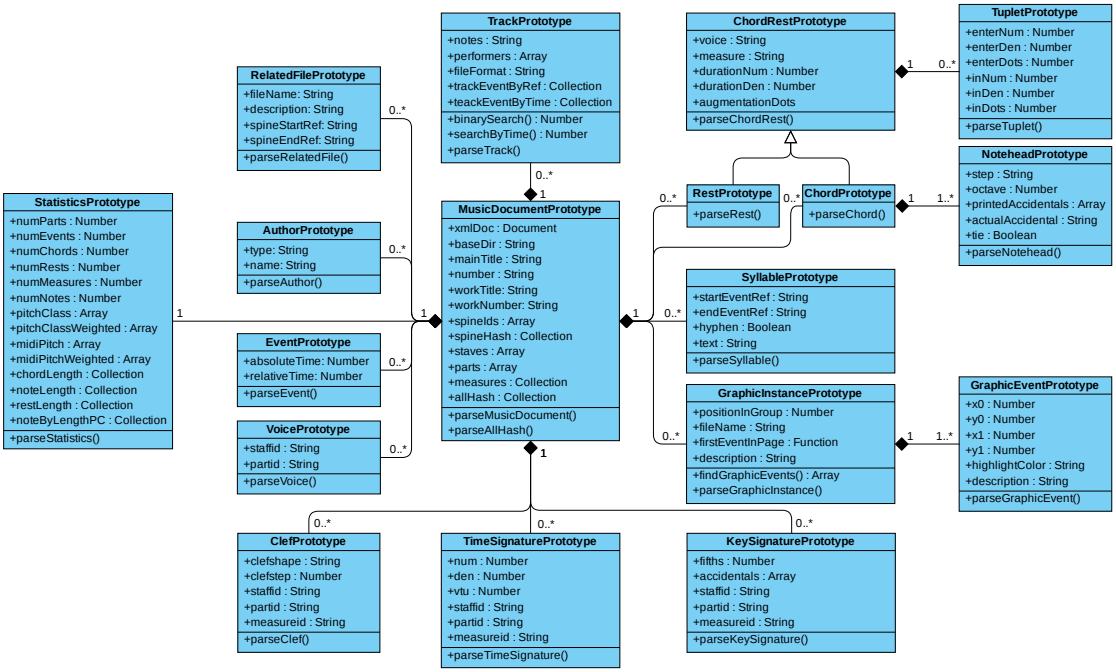


Figura 14: Prototipi di base - rappresentazione UML

Nel prosieguo di questa sezione vengono descritti i componenti del modello, corredando i diagrammi con estratti del codice che ne mostra l'implementazione JavaScipt.

MusicDocumentPrototype

Rappresenta il nucleo di base, una composizione astratta di elementi che descrivono le caratteristiche di un documento musicale: non è pensata per essere istanziata direttamente, anche se tecnicamente fattibile in JS, in quanto funge da prototipo per la creazione di specializzazioni. Dato che JS non permette nativamente di dichiarare classi astratte, si è stabilito di utilizzare il suffisso Prototype per tutti

gli oggetti di questo tipo. È possibile immaginare il seguente mapping tra gli attributi che costituiscono la struttura e i layer di IEEE 1599:

General metadati del brano e riferimenti a file esterni

- *mainTitle* - titolo del pezzo;
- *number* - posizione del pezzo nella collezione di appartenenza;
- *workTitle* - titolo della collezione di appartenenza;
- *workNumber* - numero di catalogo della collezione;
- *authors* - autori/compositori descritti con oggetti ad hoc la cui struttura di base è definita con il prototipo *AuthorPrototype*;
- *relatedFiles* - riferimenti a file esterni descritti con oggetti basati sul prototipo *RelatedFilePrototype*.

Logic eventi afferenti alla rappresentazione simbolica

Spine

- *spine* - descrive la sequenza temporale degli eventi mediante oggetti che estendono *EventPrototype*;
- *spineIds* - lista degli identificativi evento (per compatibilità con codice Player Multimediale);
- *spineHash* - tabella di hash della lista identificativi evento (per compatibilità con codice Player Multimediale).

LOS

- measures - descrive le misure presenti nel brano, ognuna della quale ha id univoco ed è associata ad un elenco delle parti in cui è presente;
- clefs - descrive gli eventi corrispondenti alla presenza di chiavi musicali associate ad una parte / pentagramma / misura mediante oggetti derivati da *ClefPrototype*;
- keys - descrive gli eventi corrispondenti alla presenza di alterazioni in armatura di chiave associate ad una parte / pentagramma / misura mediante oggetti che estendono *KeySignaturePrototype*;

- times - descrive gli eventi corrispondenti alle indicazioni di tempo associate ad una parte / pentagramma / misura mediante oggetti basati su *IimeSignaturePrototype*;
- voices - descrive le voci associandole alle parti / pentagrammi mediante oggetti derivati da *VoicePrototype*;
- chords - descrive gli eventi corrispondenti ad accordi rappresentati con oggetti basati su *ChordPrototype* che estendono una struttura di base *ChordRestPrototype* definita per fattorizzare le caratteristiche comuni alle pause;
- rests - descrive gli eventi corrispondenti alle pause rappresentate con oggetti derivati da *RestPrototype* che estendono una struttura di base *ChordRestPrototype* (fattorizza le caratteristiche comuni agli accordi);
- allHash - rappresenta l'insieme unione di pause ed accordi (introdotto per compatibilità con codice Player Multimediale);
- lyrics - rappresenta il testo cantato del brano le cui sillabe sono rappresentate con oggetti basati su SyllabePrototype.

Notational eventi afferenti alla rappresentazione grafica del contenuto musicale

- graphicInstances - rappresenta le differenti istanze grafice come collezione di oggetti che ereditano da *GraphicInstancePrototype*.

Audio eventi afferenti alle rappresentazioni audio/video

- tracks - rappresenta le tracce audio/video come collezione di oggetti basati su *TrackPrototype*.

I seguenti attributi sono stati codificati ai fini di compatibilità con il Player Multimediale:

- xmlDoc - contiene un oggetto di tipo *Document*, in genere un file XML accessibile con interfaccia *XMLDocument*;
- baseDir - indica il percorso URL (Uniform Resource Locator) dal quale è stato caricato il documento oggetto di parsing (senza il nome del file);
- stats - contiene i risultati delle elaborazioni di tipo statistico (conteggi, durate medie, etc.) effettuate sulle informazioni codificate nella struttura dati.

Sono stati definiti i seguenti metodi:

- parseMusicDocument - è un metodo generico che consente di popolare la struttura con valori dummy del tipo previsto; non è utilizzato direttamente in quanto la struttura dati non viene istanziata, contiene del codice d'esempio che è destinato all'override da parte delle specializzazioni;
- parseAllHash - esegue il merge delle informazioni codificate negli attributi *chords* e *rests* e le salva nell'attributo *allHash*.

Di seguito viene riportato il listato JavaScript in cui viene definita la struttura mediante l'utilizzo della sintassi *class*: questa parola chiave non consente di definire classi intese in senso canonico (come in Java o in altri linguaggi OO); si tratta di zucchero sintattico introdotto in EcmaScript6 che, abbinato alla parola chiave *extends*, permette di gestire in maniera semplice l'eredità mediante prototipo (sostituisce la scrittura esplicita di funzioni deputate alla generazione di oggetti in cui si setta manualmente il prototipo).

Listing 3.1: Definizione di musicDocumentPrototype in JS

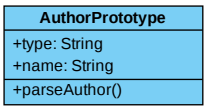
```
1  class MusicDocumentPrototype {   // MusicDocument Object
       Prototype
2    constructor() {
3      this.xmlDoc  // (XMLDocument): XMLDocument to be processed
         (if any)
4      this.baseDir;  // (String): file's path (without filename
         and extension)
5      this.mainTitle;  // (String): title of the piece
6      this.number;  // (String): position of the piece within the
         sorted collection of pieces it belongs to
7      this.workTitle;  // (String): title of the collection the
         piece belongs to
8      this.workNumber;  // (String): catalog number of the whole
         composition the piece belongs to
9      this.authors;  // (Array of Author Objects): list of all
         specified authors
10     this.relatedFiles;  // (Array of RelatedFile Objects): list
         of related files (optional)
11     this.spine;  // (Collection of Event Objects): spine[ev#]
         returns the associated event
12     this.spineIds;  // (Array of Strings): list of all the
         events ids
13     this.spineHash;  // (Collection of Numbers): spineHash["ev
         #"] returns the position in the spine list of ev#
14     this.staves;  // (Array of Strings): list of all the staves
         ids
15     this.parts;  // (Array of Strings): list of all the parts
         ids
16     this.measures;  // (Collection of Array of Strings):
         measure[number] returns the belonging parts
```

```
17      this.clefs;  // (Collection of Clef Objects): clefs["ev#"]
           returns the clef
18      this.keys;   // (Collection of KeySignature objects): keys
           ["ev#"] returns the signature
19      this.times;  // (Collection of Array of TimeSignature
           Objects): times["ev#"] returns the time (or composed
           times)
20      this.voices;  // (Collection of Voice Objects): voices["id
           #"] return the associated voice
21      this.chords;  // (Collection of Chord Objects): chords["ev
           #"] returns the corresponding chord
22      this.rests;  // (Collection of Rest Objects): rests["ev#"]
           returns the corresponding rest
23      this.allHash;  // (Collection of Chords and Rests Objects):
            allHash["ev#"] returns the corresponding chord or rest
24      this.stats;  // (Statistics Object): statistics on notes,
           rests, chords, etc.
25      this.lyrics;  // (Collection of Array of Syllable Objects):
            lyrics["voice"][syllable#] returns the corresponding
           Syllable
26      this.graphicInstances;  // (Collection of Array of
           GraphicInstance Objects): graphicInstances["description
           "][page#] returns the corresponding GraphicInstance
27      this.tracks;  // (Collection of Track Objects): tracks["
           filename"] returns the corresponding Track
28    }
29    parseMusicDocument() {
30      try {
31        this.xmlDoc = new Document();
32        this.baseDir = "";
33        this.mainTitle = "";
34        this.number = "";
35        this.workTitle = "";
36        this.workNumber = "";
37        this.authors = [new AuthorPrototype()];
38        this.relatedFiles = [new RelatedFilePrototype()];
39        this.spine = { "": new EventPrototype() };
40        this.spineIds = [""];
41        this.spineHash = { "": NaN };
42        this.staves = [""];
43        this.parts = [""];
44        this.measures = { "": [""] };
45        this.clefs = { "": new ClefPrototype() };
46        this.keys = { "": new KeySignaturePrototype() };
47        this.times = { "": [new TimeSignaturePrototype()] };
48        this.voices = { "": new VoicePrototype() };
49        this.chords = { "": new ChordPrototype() };
50        this.rests = { "": new RestPrototype() };
51        this.allHash = { "": new ChordRestPrototype() };
```

```
52        this.stats = new StatisticsPrototype();
53        this.lyrics = { "": [new SyllablePrototype()] };
54        this.graphicInstances = { "": [new
              GraphicInstancePrototype()] };
55        this.tracks = { "": new TrackPrototype };
56      }
57      catch (error) {
58        console.log(error);
59      }
60    }
61    parseAllHash() {
62      try {
63        this.allHash = {};
64        this.allHash = Object.assign({}, this.chords, this.rests)
              ;
65      }
66      catch (error) {
67        console.log(error);
68      }
69    }
70  }
```

AuthorPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative agli autori.



Attributi:

- type - tipologia del contributo fornito dall'autore;
- name - nome dell'autore.

Metodi:

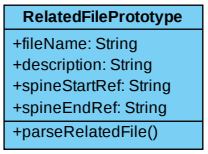
- parseAuthor - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.2: Definizione di AuthorPrototype in JS

```
1
2  class AuthorPrototype {  // Author Object Prototype
3    constructor() {
4      this.type;  // (String): the role of the author within the
           composition
5      this.name;  // (String): name of the author of the piece
6    }
7    parseAuthor() {
8      try {
9        this.type = "";
10       this.name = "";
11     }
12     catch (error) {
13       console.log(error);
14     }
15   }
16 }
```

RelatedFilePrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative a file esterni, collegati all'opera, che non costituiscono una rappresentazione afferente ad altri layer.



Attributi:

- fileName - percorso completo per l'identificazione del file;
- description - descrizione testuale del contenuto;
- spineStartRef - riferimento ad un evento iniziale collegato al file;
- spineEndRef - riferimento ad un evento di fine collegato al file.

Metodi:

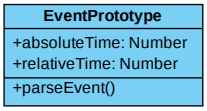
- parseRelatedFile - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.3: Definizione di RelatedFilePrototype in JS

```
1  class RelatedFilePrototype {  // RelatedFile Object Prototype
2    constructor() {
3      this.fileName;  // (String): complete path to identify the
           digital object
4      this.description;  // (String): textual description of the
           content
5      this.spineStartRef;  // (String): identifiers of music
           events listed in spine - appearance of digital objects
           (optional)
6      this.spineEndRef;  // (String): identifiers of music events
            listed in spine - disappearance of digital objects (
           optional)
7    }
8    parseRelatedFile() {
9      try {
10       this.fileName = "";
11       this.description = "";
12       this.spineStartRef;
13       this.spineEndRef;
14     }
15     catch (error) {
16       console.log(error);
17     }
18   }
19 }
```

EventPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla memorizzazione della sequenza temporale degli eventi.



Attributi:

- absoluteTime - coordinate temporali dell'evento (contatore);
- relativeTime - coordinate temporali dell'evento (delta rispetto a evento precedente).

Metodi:

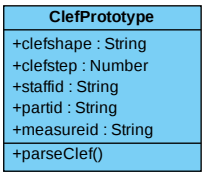
- parseEvent - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.4: Definizione di EventPrototype in JS

```
1  class EventPrototype {  // Event Object Prototype
2    constructor() {
3      this.absoluteTime;  // (Number): temporal coordinate of the
           event: time offset from t0 in vtu
4      this.relativeTime;  // (Number): temporal coordinate of the
           event: time offset from previus event in vtu
5    }
6    parseEvent() {
7      try {
8        this.absoluteTime = NaN;
9        this.relativeTime = NaN;
10     }
11     catch (error) {
12       console.log(error);
13     }
14   }
15 }
```

ClefPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative alle chiavi musicali.



Attributi:

- clefshape - tipo di chiave (chiave di sol, chiave di fa, etc.);
- clefstep - posizione della chiave sul pentagramma;
- staffid - pentagramma di riferimento;
- partid - parte di riferimento;
- measureid - misura di riferimento.

Metodi:

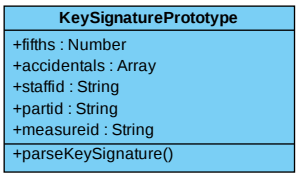
- parseClef - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.5: Definizione di ClefPrototype in JS

```
1  class ClefPrototype {  // Clef Object Prototype
2    constructor() {
3      this.clefshape;  // (String): shape of the clef (treble,
           bass, C clefs, tabguitar, percussion, etc.)
4      this.clefstep;  // (Number): vertical position of the clef
           (lines and spaces from the bottom line)
5      this.staffid;  // (String): staff to which the clef belongs
            to (alternative to partid + measureid)
6      this.partid;  // (String): part to which the clef belongs
           to (alternative to staffid)
7      this.measureid;  // (String): measure to which the clef
           belongs to (alternative to staffid)
8    }
9    parseClef() {
10     try {
11       this.clefshape = "";
12       this.clefstep = NaN;
13       this.staffid;
14       this.partid;
15       this.measureid;
16     }
17     catch (error) {
18       console.log(error);
19     }
20   }
21 }
```

KeySignaturePrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative alle alterazioni in chiave.



Attributi:

- fifths - numero di alterazioni in chiave presenti (valori positivi per i *diesis*, negativi per i *bemolle*);
- accidentals - indicazione esplicita delle alterazioni presenti;
- staffid - pentagramma di riferimento;
- partid - parte di riferimento;
- measureid - misura di riferimento.

Metodi:

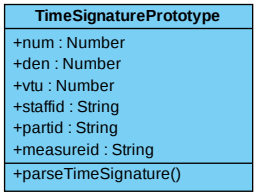
- parseKeySignature - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.6: Definizione di KeySignaturePrototype in JS

```
1  class KeySignaturePrototype {  // KeySignature Object Prototype
2    constructor() {
3      this.fifths;  // (Number): number of sharp (positive) or
           flat (negative) [-7, ... ,7] (alternative to
           accidentals)
4      this.accidentals;  // (Array of Strings): custom key (
           alternative to fifths)
5      this.staffid;  // (String): staff to which the key
           signature belongs to (alternative to partid + measureid
           )
6      this.partid;  // (String): part to which the key signature
           belongs to (alternative to staffid)
7      this.measureid;  // (String): measure to which the key
           signature belongs to (alternative to staffid)
8    }
9    parseKeySignature() {
10     try {
11       this.fifths;
12       this.accidentals;
13       this.staffid;
14       this.partid;
15       this.measureid;
16     }
17     catch (error) {
18       console.log(error);
19     }
20   }
21 }
```

TimeSignaturePrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative al tempo.



Attributi:

- num - numeratore della frazione che rappresenta il tempo;
- den - denominatore della frazione che rappresenta il tempo;
- vtu - durata in VTU secondo quanto definito nello spine;
- staffid - pentagramma di riferimento;
- partid - parte di riferimento;
- measureid - misura di riferimento.

Metodi:

- parseTimeSignature - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

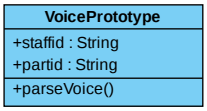
Listing 3.7: Definizione di TimeSignaturePrototype in JS

```
1 class TimeSignaturePrototype {  // TimeSignature Object
     Prototype
2   constructor() {
3     this.num;  // (Number): numerator of the fractional
        representation of tempo
4     this.den;  // (Number): denominator of the fractional
        representation of tempo (optional)
5     this.vtu;  // (Number): length of the measure in Virtual
        Time Unit according to spine (optional)
6     this.staffid;  // (String): staff to which the time
        signature belongs to (alternative to partid + measureid
        )
```

```
7       this.partid;  // (String): part to which the time signature
            belongs to (alternative to staffid)
8       this.measureid;  // (String): measure to which time
            signature belongs to (alternative to staffid)
9     }
10    parseTimeSignature() {
11      try {
12        this.num = NaN;
13        this.den;
14        this.vtu;
15        this.staffid;
16        this.partid;
17        this.measureid;
18      }
19      catch (error) {
20        console.log(error);
21      }
22    }
23  }
```

VoicePrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative alle voci di una parte musicale.



Attributi:

- staffid - pentagramma di riferimento;
- partid - parte di riferimento.

Metodi:

- parseVoice - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

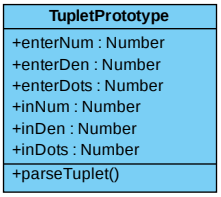
Listing 3.8: Definizione di VoicePrototype in JS

```
1  class VoicePrototype {  // Voice Object Prototype
2    constructor() {
3      this.staffid;  // (String): staff to which the voice
           belongs
```

```
4       this.partid;  // (String): part to which the voice belongs
5     }
6     parseVoice() {
7       try {
8         this.staffid = "";
9         this.partid = "";
10      }
11      catch (error) {
12        console.log(error);
13      }
14    }
15  }
```

TupletPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative a gruppi irregolari di note e pause per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.



Gli attributi ricalcano quelli utilizzati in IEEE 1599 per la definizione delle tuple:

- enterNum - numeratore della frazione che indica la tipologia di gruppo irregolare;
- enterDen - denominatore della frazione che indica la tipologia di gruppo irregolare;
- enterDots - eventuali punti per alterazione durata della tipologia del gruppo irregolare;
- inNum - numeratore della frazione che indica la durata degli elementi del gruppo regolare nel tempo corrente;
- inDen - denominatore della frazione che indica la durata degli elementi del gruppo regolare nel tempo corrente;

- inDots - eventuali punti per alterazione durata degli elementi del gruppo irregolare nel tempo corrente.

Metodi:

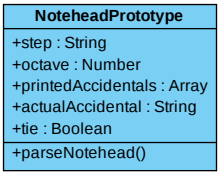
- parseTuplet - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.9: Definizione di TupletPrototype in JS

```
1  class TupletPrototype {  // Tuplet Object Prototype
2    constructor() {
3      this.enterNum;  // (Number): numerator of the fraction
          representing the number of tuplet values
4      this.enterDen;  // (Number): denominator of the fraction
          representing the number of tuplet values
5      this.enterDots;  // (Number): dotted values in tuplet
          values (optional)
6      this.inNum;  // (Number): numerator of the fraction
          representing the number of actual values in the current
           time signature
7      this.inDen;  // (Number): denominator of the fraction
          representing the number of actual values in the current
           time signature
8      this.inDots;  // (Number): dotted values in actual values (
          optional)
9    }
10   parseTuplet() {
11     try {
12       this.enterNum = NaN;
13       this.enterDen = NaN;
14       this.enterDots;
15       this.inNum = NaN;
16       this.inDen = NaN;
17       this.inDots;
18     }
19     catch (error) {
20       console.log(error);
21     }
22   }
23 }
```

NoteheadPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative alle note musicali per l'utilizzo in strutture dati derivate da *ChordPrototype*.



Attributi:

- step - nome della nota in notazione anglosassone [A-G];
- octave - altezza espressa in ottave;
- printedAccidentals - alterazioni grafiche associate alla nota;
- actualAccidental - valore reale delle alterazioni;
- tie - presenza della legatura di valore.

Metodi:

- parseNotehead - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

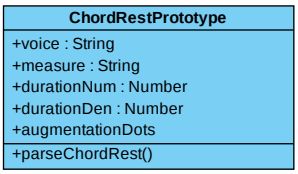
Listing 3.10: Definizione di NoteheadPrototype in JS

```
1  class NoteheadPrototype {  // Notehead Object Prototype
2    constructor() {
3      this.step;  // (String): name for the notehead, according
           to letter names [A-G]
4      this.octave;  // (Number): otave information.
           Conventionally, the central octave is the fifth one
5      this.printedAccidentals;  // (Array of String): accidental
           graphic signs assigned to the current notehead in the
           score, including courtesy accidentals
6      this.actualAccidental;  // (String): real inflecting of a
           note (default "normal")
7      this.tie;  // (Boolean): presence of a tie starting from
           the current notehead (default "false")
8    }
9    parseNotehead() {
10     try {
11       this.step = "";
12       this.octave = NaN;
13       this.printedAccidentals = [""];
```

```
14          this.actualAccidental = "";
15          this.tie = false;
16        }
17        catch (error) {
18          console.log(error);
19        }
20      }
21  }
```

ChordRestPrototype

Questo prototipo fattorizza le caratteristiche comuni alle note e alle pause.



Attributi:

- voice - voce di riferimento;
- measure - misura in cui occorre l'evento;
- durationNum - numeratore della frazione che indica la durata dell'evento;
- durationDen - denominatore della frazione che indica la durata dell'evento;
- tuplet - specifica se è presente un gruppo irregolare mediante oggetti derivati da *TupletPrototype*;
- augmentationDots - numero dei punti di valore presenti.

Metodi:

- parseChordRest - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

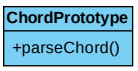
Listing 3.11: Definizione di ChordRestPrototype in JS

```
1  class ChordRestPrototype {  // Prototype for Chord and Rest
       Objects
2    constructor() {
```

```
3       this.voice;  // (String): voice name of a specified chord
            or rest
4       this.measure;  // (String): measure number of a specified
            chord or rest
5       this.durationNum;  // (Number): duration numerator of a
            specified chord or rest [8,4,2,1]
6       this.durationDen;  // (Number): duration denominator of a
            specified chord or rest
            [1,2,4,8,16,32,64,128,256,512,1024]
7       this.tuplet;  // (Tuplet Object): tuplet of a specified
            chord (optional)
8       this.augmentationDots;  // (Number): number of augmentation
             dots for a chord (optional)
9     }
10    parseChordRest() {
11      try {
12        this.voice = "";
13        this.measure = "";
14        this.durationNum = NaN;
15        this.durationDen = NaN;
16        this.tuplet;
17        this.augmentationDots;
18      }
19      catch (error) {
20        console.log(error);
21      }
22    }
23  }
```

ChordPrototype

Questo prototipo costituisce un esempio di implementazione della struttura *ChordRestPrototype* per la rappresentazione di accordi composti da elementi basati sul tipo *NoteheadPrototype*.



Attributi:

- noteheads - note che compongono l'accordo.

Metodi:

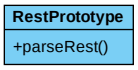
- parseChord - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.12: Definizione di ChordPrototype in JS

```
1  class ChordPrototype extends ChordRestPrototype {  // Chord
       Object Prototype
2    constructor() {
3      super();
4      this.noteheads;  // (Array of Notehead Objects): list of
         chord's notes
5    }
6    parseChord() {
7      try {
8        this.noteheads = [new NoteheadPrototype];
9      }
10     catch (error) {
11       console.log(error);
12     }
13   }
14 }
```

RestPrototype

Questo prototipo costituisce un esempio di implementazione della struttura *ChordRestPrototype* per la rappresentazione di pause.



Attributi: nessuno aggiuntivo rispetto a quelli ereditati da *ChordRestPrototype*.
Metodi:

- parseRest - metodo destinato all'override.

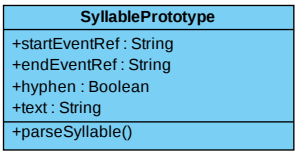
Listing 3.13: Definizione di RestPrototype in JS

```
1  class RestPrototype extends ChordRestPrototype {  // Rest
       Object Prototype
2    constructor() {
3      super();
4    }
5    parseRest() { }
6  }
```

SyllablePrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative alle sillabe che compongono il testo cantato di un brano

musicale.



Attributi:

- startEventRef - evento dello spine in cui la sillaba ha inizio;
- endEventRef - evento dello spine in cui la sillaba ha fine;
- hyphen - presenza del trattino di separazione delle sillabe;
- text - testo della sillaba.

Metodi:

- parseSyllable - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

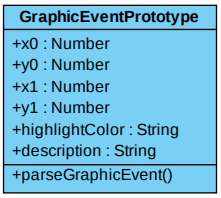
Listing 3.14: Definizione di SyllablePrototype in JS

```
1  class SyllablePrototype {  // Syllabe Object Prototype
2    constructor() {
3      this.startEventRef;  // (String): first spine event
           corresponding to the music event where the syllable
           occurs
4      this.endEventRef;  // (String): last spine event to which
           the syllable should be associated (optional)
5      this.hyphen;  // (Boolean): syllable has to be followed by
           a hyphenation sign
6      this.text;  // (String): text to be associated to a music
           symbol
7    }
8    parseSyllable() {
9      try {
10       this.startEventRef = "";
11       this.endEventRef;
12       this.hyphen = "yes";
13       this.text = "";
14     }
15     catch (error) {
16       console.log(error);
```

```
17        }
18      }
19  }
```

GraphicEventPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative agli eventi grafici appartenenti ad una determinata istanza descritta da una struttura derivata da *GraphicInstancePrototype*.



Attributi:

- x0 - coordinata x0 dell'evento grafico;
- y0 - coordinata y0 dell'evento grafico;
- x1 - coordinata x1 dell'evento grafico;
- y1 - coordinata y1 dell'evento grafico;
- highlightColor - colore del cursore delimitato dalla coordinate;
- description - descrizione dell'evento grafico.

Metodi:

- parseGraphicEvent - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

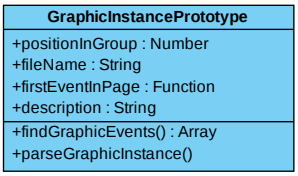
Listing 3.15: Definizione di GraphicEventPrototype in JS

```
1 class GraphicEventPrototype {  // GraphicEvent Object Prototype
2   constructor() {
3     this.x0;  // (Number): coordinate x0 of a specified graphic
          event
4     this.y0;  // (Number): coordinate y0 of a specified graphic
          event
```

```
5       this.x1;  // (Number): coordinate x1 of a specified graphic
            event
6       this.y1;  // (Number): coordinate y1 of a specified graphic
            event
7       this.highlightColor;  // (String): color for the bounding
            box, by using an RGB code both in decimal and in
            hexadecimal values (optional)
8       this.description;  // (String): description of a specified
            graphic event (optional)
9     }
10    parseGraphicEvent() {
11      try {
12        this.x0 = NaN;
13        this.y0 = NaN;
14        this.x1 = NaN;
15        this.y1 = NaN;
16        this.highlightColor;
17        this.description;
18      }
19      catch (error) {
20        console.log(error);
21      }
22    }
23  }
```

GraphicInstancePrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative agli eventi grafici appartenenti ad una determinata istanza descritta da una struttura derivata da *GraphicInstancePrototype*.



Attributi:

- positionInGroup - posizione dell'istanza grafica all'interno di un gruppo di istanze (come da specifiche layer Notational IEEE 1599);
- fileName - percorso completo del file;

- firstEventInPage - richiama una funzione che restituisce l'id del primo evento che occorre nell'istanza (per compatibilità con Player Multimediale);
- description - descrizione dell'istanza;
- graphicEvents - insieme degli eventi grafici presenti nell'istanza, rappresentati da strutture che implementano il prototipo *GraphicEventPrototype*.

Metodi:

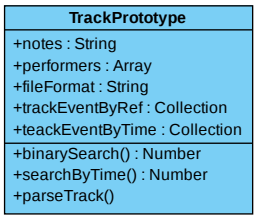
- findGraphicEvents - metodo per la ricerca di eventi grafici che contengono un determinato punto dell'immagine (per compatibilità con Player Multimediale);
- parseGraphicInstance - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.16: Definizione di GraphicInstancePrototype in JS

```
1  class GraphicInstancePrototype {  // GraphicInstance Object
       Prototype
2    constructor() {
3      this.positionInGroup;  // (Number): numbering of the
         sub-parts within a graphic instance group
4      this.fileName;  // (String): complete path to identify the
         digital object
5      this.firstEventInPage;  // (Function): returns the ID of
         the first event in the specified graphic instance page
6      this.description;  // (String): textual description of the
         file (optional)
7      this.graphicEvents;  // (Collection of Array of
         GraphicEvent Objects): list of graphic events
8    }
9    findGraphicEvents() { }  // (Function): given a point,
       returns an Array of GraphicEvent Objects whose bounding
       boxes contains that point
10   parseGraphicInstance() {
11     try {
12       this.positionInGroup = NaN;
13       this.fileName = "";
14       this.firstEventInPage = function () { };
15       this.description;
16       this.graphicEvents = { "": [new GraphicEventPrototype] };
17     }
18     catch (error) {
19       console.log(error);
20     }
21   }
22 }
```

TrackPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni relative alle tracce audio/video.



Attributi:

- notes - descrizione di metadati generici riguardanti la traccia;
- performers - elenco dei musicisti/interpreti;
- fileFormat - formato di codifica della traccia;
- trackEventsByRef - associa gli eventi audio agli istanti di esecuzione;
- trackEventsByTime - associa gli istanti di esecuzione agli eventi audio.

Metodi:

- binarySearch - ricerca binaria di un istante temporale all'interno di segmenti temporali (utilizzato da Player Multimediale);
- searchByTime - ricerca dell'evento che occorre in un determinato istante temporale (utilizzato da Player Multimediale);
- parseTrack - metodo d'esempio che inizializza gli attributi obbligatori con valori vuoti del tipo previsto.

Listing 3.17: Definizione di TrackPrototype in JS

```
1 class TrackPrototype {  // Track Object Prototype
2   constructor() {
3     this.notes;  // (String): general notes about the piece,
        additional catalog information, software applications
        and hardware devices etc. (optional)
```

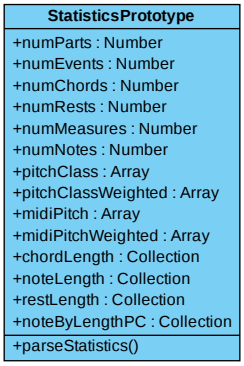
```
4       this.performers;  // (Array of Strings): performers name +
            type (optional)
5       this.fileFormat;  // (String): file format of a specified
            track
6       this.trackEventsByRef;  // (Collection of Numbers):
            trackEventsByRef["eventID"] start time (Number) of a
            specified event of a specified track
7       this.trackEventsByTime;  // (Collection of Array of Strings
            ): trackEventsByTime[time] event id of the i-th event
            starting at the given time
8     }
9     binarySearch() { }  // (Function): given a list and a value,
          returns the index of the list element containing the
          value
10    searchByTime() { }  // (Function): given a time, return the
          event id of the event occourring at that given time
11    parseTrack() {
12      try {
13        this.notes;
14        this.performers;
15        this.fileFormat = "";
16        this.trackEventsByRef = { "": NaN };
17        this.trackEventsByTime = { "": [""] };
18      }
19      catch (error) {
20        console.log(error);
21      }
22    }
23  }
```

StatisticsPrototype

Questo prototipo definisce la struttura base degli oggetti deputati alla codifica delle informazioni di tipo statistico ad uso della vista relativa del Player Multimediale. Il metodo *parseStatistics* è una rivisitazione della controparte presente nell'*ieee1599-framework* per l'utilizzo delle nuove strutture dati: il codice sorgente completo è consultabile in Appendice B

Attributi:

- numParts - conteggio delle parti;
- numEvents - conteggio degli eventi;
- numChords - conteggio degli accordi;
- numRests - conteggio delle pause;



- numMeasures - conteggio delle misure;
- numNotes - conteggio delle note;
- pitchClass - conteggio delle note per altezza;
- pitchClassWeighted - conteggio delle note per altezza e per durata;
- midiPitch - conteggio delle note corrispondenti alle altezze MIDI;
- midiPitchWeighted - conteggio delle note corrispondenti alle altezze MIDI per durata;
- chordLength - conteggio degli accordi per durata;
- noteLength - conteggio delle note per durata;
- restLength - conteggio delle pause per durata;
- noteByLengthPC - conteggio delle note per durata e per altezza.

Metodi:

- parseStatistics - popola gli attributi di tipo statistico di un oggetto derivato da *MusicDocumentPrototype* che viene passato come parametro.

Listing 3.18: Definizione di StatisticsPrototype in JS

```
1 class StatisticsPrototype {     // Statistics Object Prototype
2   constructor() {
3     this.numParts;        // (Number): number of parts
```

```
4       this.numEvents;         // (Number): number of events
5       this.numChords;         // (Number): number of chords
6       this.numRests;         // (Number): number of rests
7       this.numMeasures;        // (Number): number of measures
8       this.numNotes;         // (Number): number of notes
9       this.pitchClass;        // (Array of Numbers): pitchClass[i]
           number of notes having pitch class i
10      this.pitchClassWeighted;  // (Array of Numbers):
           pitchClassWeihted[i] number of notes having pitch class
            i weighted on duration
11      this.midiPitch;         // (Array of Numbers): midiPitch[i]
           number of notes having midi pitch i
12      this.midiPitchWeighted;    // (Array of Numbers):
           midiPitchWeighted[i] number of notes having midi pitch
           i weighted on duration
13      this.chordLength;        // (Collection of Numbers): number
           of chords per duration (unit: quarter)
14      this.noteLength;        // (Collection of Numbers): number of
            notes per duration (unit: quarter)
15      this.restLength;        // (Collection of Numbers): number of
            rest per duration (unit: quarter)
16      this.noteByLengthPC;     // (Collection of collections of
           numbers): noteByLengthPC[i][j] number of notes having
           wighted duration i and pitch j
17    }
18    parseStatistics(MusicDocument) {
19      try {
20        [...]
21      }
22      catch (error) {
23        console.log(error);
24      }
25    }
26  }
```

3.4 Creazione di documenti

Questa sezione descrive le funzioni che sono state implementate per istanziare dinamicamente differenti tipologie di oggetti, derivati da *MusicDocumentPrototype*, a partire da un file accessibile mediante percorso URL.

Funzione getUrl

Questa funzione accede ad un URL pubblicato da un WebServer, sfruttando l'oggetto XMLHttpRequest (XHR), per scaricare il file da processare client-side. La

richiesta di download (get) deve avvenire in maniera asincrona, nel rispetto delle best-practice consultabili sul sito Internet dello standard XHR[5] (l'utilizzo di chiamate sincrone è deprecato in quanto impatta negativamente sull'esperienza utente). Per rispettare questo vincolo, e mantenere al contempo il controllo sullo stato dell'esecuzione, si è pensato di ricorrere all'utilizzo delle *JavaScript Promise*: si tratta di oggetti che rappresentano un'operazione asincrona (nel caso specifico una richiesta/get di un file) il cui esito (successo o fallimento) viene gestito nel momento in cui l'attività richiesta risulta completata; le azioni da compiere nei due casi vengono definite con funzioni ad hoc.

Listing 3.19: Promise based XHR get

```
function getUrl(url) {
    return new Promise(function (resolve, reject) {
        const xhr = new XMLHttpRequest();
        xhr.onreadystatechange = function () {
            if (xhr.readyState === 4) {
                if (xhr.status === 200) {
                    resolve(xhr)
                } else {
                    reject("XHR error " + xhr.status + " (" +
                        xhr.statusText + ") " + "for url '" +
                        url + "'");
                }
            }
        }
        xhr.ontimeout = function () {
            reject("XHR timeout for url '" + url + "'");
        }
        xhr.open("get", url, true)
        xhr.send()
    });
}
```

Oggetti extMap e fileTypeMap

Il tipo di oggetto da istanziare viene determinato sulla base di due elementi: estensione e formato file. Vengono definiti i seguenti oggetti:

- *extMap* - associa un'estensione file ad una delle factory function implementate per la creazione dei documenti;
- *fileTypeMap* - associa un tipo di file ad una funzione che restituisce una *promise* per la creazione dell'istanza di un oggetto basato su *MusicDocumentPrototype*.

[5] https://xhr.spec.whatwg.org/

Nel caso in cui emergesse la necessità di introdurre ulteriori specializzazioni di *MusicDocumentPrototype*, per trattare nuove tipologie di file, sarà sufficiente aggiungere nuove associazioni a questi oggetti.

Per la definizione delle funzioni in *fileTypeMap* si è fatto ricorso all'uso della notazione di tipo *arrow* (compatta e particolarmente indicata nel caso di funzioni anonime) e della sintassi *async* che, oltre a strutturare automaticamente i valori di ritorno in forma di *promise*, permette di utilizzare il costrutto *await* (attesa risoluzione di un'operazione asincrona). Ai fini delle attività di interesse, per citare un esempio concreto, *await* viene utilizzato per gestire la *promise* restituita dalla funzione *getUrl* per il download del foglio XSL da utilizzarsi per la trasformazione XSLT di documenti MusicXML di tipo *score-timewise*: la conversione in rappresentazione *score-partwise* avviene solo a fronte del corretto download del file.

Listing 3.20: extMap e fileTypeMap

```
1  const extMap = {  // maps an extension to a document factory
2    "xml": XmlMusicDocumentFactory,
3    "musicxml": XmlMusicDocumentFactory,
4    "krn": TxtMusicDocumentFactory
5  };
6
7  const fileTypeMap = {  // maps a file type to a function
       returning a document promise
8    "ieee1599": async (doc, url) => new IEEE1599Document(doc, url
       ),
9    "score-partwise": async (doc, url) => new MusicXmlDocument(
       doc, url),
10   "score-timewise": async (doc, url) => {
11     let xsl = await getUrl("/xsl/timepart.xsl");
12     let xsltProcessor = new XSLTProcessor();
13     xsltProcessor.importStylesheet(xsl.responseXML);
14     let partwiseDoc = xsltProcessor.transformToDocument(doc);
15     return new MusicXmlDocument(partwiseDoc, url)
16   },
17   "krn": async (doc, url) => new KernDocument(doc, url),
18 };
```

Factory functions

Quando una chiamata XMLHttpRequest get ha successo viene generata una risposta che, a seconda della tipologia dell'elemento richiesto, può essere ricondotta a specifiche tipologie di oggetti JS (ArrayBuffer, Blob, Document, DOMString) invocando metodi ad hoc.

Le funzioni *XmlMusicDocumentFactory* e *TxtMusicDocumentFactory* processano rispettivamente gli oggetti XHR relativi a get di file in formato XML, dai quali si ricavano oggetti Document (metodo *responseXML*), e a get di file in formato testuale, dai quali si ricavano oggetti di tipo DOMString (metodo *responseText*). Analizzando gli oggetti XHR, vengono identificati i tipi di file ed invocate la funzioni associate, in *fileTypeMap*, per la creazione delle promise di costruzione delle specializzazioni di oggetti derivati da *MusicDocumentPrototype*: ai relativi costruttori vengono fornite le rappresentazioni di pertinenza (Document o DOMString, a seconda della funzione), unitamente al percorso url del file di origine.

Questo approccio consente al modello di essere facilmente estendibile, per trattare altri tipi di risposte, mediante l'implementazione di nuove funzioni ad hoc.

Listing 3.21: XmlMusicDocumentFactory e TxtMusicDocumentFactory

```
1  function XmlMusicDocumentFactory(xhr) {
2    let type = xhr.responseXML.doctype.name;
3    if (type in fileTypeMap) {
4      return fileTypeMap[type](xhr.responseXML, xhr.responseURL);
5    }
6    else {
7      throw ("Unknown DocType : '" + type + "' for url '" +
           xhr.responseURL + "'")
8    }
9  }
10
11 function TxtMusicDocumentFactory(xhr) {
12   let type = xhr.responseURL.split('.').pop();
13   if (type in fileTypeMap) {
14     return fileTypeMap[type](xhr.responseText, xhr.responseURL)
         ;
15   }
16   else {
17     throw ("Unknown DocType '" + type + "' for url '" +
           xhr.responseURL + "'")
18   }
19 }
```

Funzione getMusicDocumentPromise

getMusicDocumentPromise è la funzione asincrona che viene invocata dal codice chiamante per avviare il download di file da percorsi url ed istanziare oggetti derivati da *MusicDocumentPrototype*. In prima istanza si occupa di:

- effettuare un check sulla corretta formattazione del percorso url fornito come argomento;

- estrarre l'estensione del file da processare;
- invocare la funzione *getUrl*, attendendo che la promise XHR restituita venga risolta (utilizzo di *await*).

L'estensione del file viene quindi utilizzata per invocare la factory function associata (in *extMap*): come parametro viene passato l'oggetto XHR restituito da *getUrl*. La promise restituita dalla funzione invocata verrà resa al codice chiamante.

Listing 3.22: getMusicDocumentPromise

```
1  async function getMusicDocumentPromise(url) {
2    if (url === "") {
3      throw ("Empty url");
4    }
5    if (typeof (url) !== typeof ("")) {
6      throw ("Url should be a string but is '" + typeof (url) + "
         '");
7    }
8    if (!url.includes('.')) {
9      throw ("Missing file extension for url '" + url + "'");
10   }
11   let ext = url.split('.').pop();
12   let xhr = await getUrl(url);
13   if (ext in extMap) {
14     return extMap[ext](xhr);
15   }
16   else {
17     throw ("Unknown file extension '" + ext + "' for url '" +
         url + "'");
18   }
19 }
```

3.5 Suite di test

Javascript è un linguaggio interpretato a tipizzazione debole: le variabili, durante il loro ciclo di vita, possono essere associate dinamicamente ad elementi primitivi di diverso tipo (e anche ad oggetti, funzioni etc). Durante la fase di sviluppo, si è ritenuto opportuno introdurre degli unit-test che permettessero di verificare che gli oggetti istanziati rispettassero le convenzioni stabilite per le strutture dati ideate per il modello. Con l'avanzare della scrittura del codice, i vari casi di test sono stati riuniti in una suite e la pagina HTML di test, fornita a corredo del toolkit di qUnit, è stata adattata per mostrare i risultati della procedura di verifica: la Figura 15 illustra gli esiti della validazione effettuata sulla struttura ottenuta dal

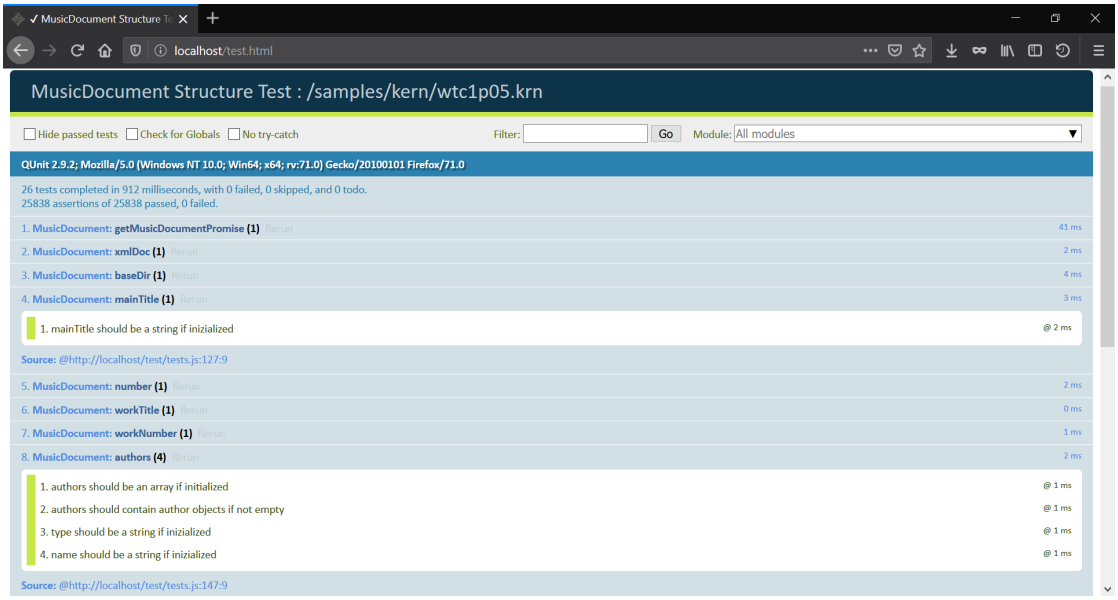


Figura 15: test.html - risultati della validazione di un oggetto MusicDocument

parsing di un documento musicale. Il codice sorgente della pagina HTML di test è consultabile in Appendice C.

La disponibilità di una suite di test si è rivelata di grande utilità, in particolar modo nella fase di codifica delle strutture specialistiche / implementazione algoritmi di parsing descritta nel Capitolo 4: l'esecuzione periodica dei casi di test ha permesso di evidenziare incongruenze tra gli elementi, facilitando il debug e la ricerca delle porzioni di codice sorgente sulle quali si rendeva necessario intervenire in correzione.

Nel prosieguo della sezione vengono illustrati i casi di test implementati.

getMusicDocumentPromise Verifica che la *promise* restituita dalla funzione *getMusicDocumentPromise* venga risolta correttamente con la creazione di un oggetto derivato da *MusicDocumentPrototype*

```
let url = "/samples/kern/sample.krn";
document.getElementsByTagName("title")[0].text += " : " + url;
let myMusicDocumentPromise = getMusicDocumentPromise(url);
let myMusicDocument;
QUnit.test("getMusicDocumentPromise", function (assert) {
  assert.expect(1);
  let done = assert.async();
  myMusicDocumentPromise
    .then((result) => {
      myMusicDocument = result;
```

```
11      })
12      .catch((err) => {
13        console.log(err);
14      })
15      .finally(() => {
16        // myMusicDocument
17        assert.ok(myMusicDocument instanceof
              MusicDocumentPrototype, 'musicDocument should be
              based on MusicDocumentPrototype');
18        done();
19      });
20  });
```

xmlDoc Verifica che l'attributo *xmlDoc* contenga un oggetto di tipo *Document*.

```
1  QUnit.test("xmlDoc", function (assert) {
2    // myMusicDocument.xmlDoc
3    assert.ok(myMusicDocument.xmlDoc instanceof Document, 'xmlDoc
         should be an XMLDocument');
4  });
```

baseDir Verifica che l'attributo *baseDir* contenga una stringa.

```
1  QUnit.test("baseDir", function (assert) {
2    // myMusicDocument.baseDir
3    assert.deepEqual(typeof (myMusicDocument.baseDir), typeof ("
         "), 'baseDir should be a string if inizialized');
4  });
```

mainTitle Verifica che l'attributo *mainTitle* contenga una stringa.

```
1  QUnit.test("mainTitle", function (assert) {
2    // myMusicDocument.mainTitle
3    assert.deepEqual(typeof (myMusicDocument.mainTitle), typeof (
         ""), 'mainTitle should be a string if inizialized');
4  });
```

number Verifica che l'attributo *number* contenga una stringa.

```
1  QUnit.test("number", function (assert) {
2    // myMusicDocument.number
3    assert.deepEqual(typeof (myMusicDocument.number), typeof ("")
         , 'number should be a string if inizialized');
4  });
```

workTitle Verifica che l'attributo *workTitle* contenga una stringa.

```
1 QUnit.test("workTitle", function (assert) {
2   // myMusicDocument.workTitle
3   assert.deepEqual(typeof (myMusicDocument.workTitle), typeof (
      ""), 'workTitle should be a string if inizialized');
4 });
```

workNumber Verifica che l'attributo *workNumber* contenga una stringa.

```
1 QUnit.test("workNumber", function (assert) {
2   // myMusicDocument.workNumber
3   assert.deepEqual(typeof (myMusicDocument.workNumber), typeof
      (""), 'workNumber should be a string if inizialized');
4 });
```

authors Verifica che l'attributo *authors* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *AuthorPrototype* aventi attributi di tipo stringa (*type* e *name*).

```
1  QUnit.test("authors", function (assert) {
2    // myMusicDocument.authors
3    assert.ok(myMusicDocument.authors instanceof Array, 'authors
       should be an array if initialized');
4    for (let i = 0; i < myMusicDocument.authors.length; i++) {
5      let myAuthor = myMusicDocument.authors[i];
6      assert.ok(myAuthor instanceof AuthorPrototype, 'authors
         should contain author objects if not empty');
7      assert.deepEqual(typeof (myAuthor.type), typeof (""), 'type
         should be a string if inizialized');
8      assert.deepEqual(typeof (myAuthor.name), typeof (""), 'name
         should be a string if inizialized');
9    }
10 });
```

relatedFiles Verifica che l'attributo *relatedFiles* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *RelatedFilePrototype* aventi attributi di tipo stringa (*fileName*, *description*, *spineStartRef* e *spineEndRef*).

```
1 QUnit.test("relatedFiles", function (assert) {
2   // myMusicDocument.relatedFiles
3   assert.ok(myMusicDocument.relatedFiles instanceof Array, '
      relatedFiles should be an array if initialized');
4   for (let i = 0; i < myMusicDocument.relatedFiles.length; i++)
      {
```

```
5       let myRelatedFile = myMusicDocument.relatedFiles[i];
6       assert.ok(myRelatedFile instanceof RelatedFilePrototype, '
            relatedFiles should contain relatedfile objects if not
            empty');
7       assert.deepEqual(typeof (myRelatedFile.fileName), typeof ("
            "), 'fileName should be a string if inizialized');
8       assert.deepEqual(typeof (myRelatedFile.description), typeof
             (""), 'description should be a string if inizialized')
            ;
9       if (myRelatedFile.spineStartRef !== undefined) {
10        assert.deepEqual(typeof (myRelatedFile.spineStartRef),
              typeof (""), 'spineStartRef should be a string if
              inizialized (optional attribute)');
11      }
12      if (myRelatedFile.spineEndRef !== undefined) {
13        assert.deepEqual(typeof (myRelatedFile.spineEndRef),
              typeof (""), 'spineEndRef should be a string if
              inizialized (optional attribute)');
14      }
15    }
16  });
```

spine Verifica che l'attributo *spine* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *EventPrototype* aventi attributi *absoluteTime* e *relativeTime* di tipo numerico.

```
1   QUnit.test("spine", function (assert) {
2     // myMusicDocument.spine
3     assert.deepEqual(typeof (myMusicDocument.spine), typeof ({}),
          'spine should be an object if initialized');
4     let spineKeys = Object.keys(myMusicDocument.spine);
5     for (let i = 0; i < spineKeys.length; i++) {
6       let eventId = spineKeys[i];
7       assert.deepEqual(typeof (eventId), typeof (""), 'spine
            eventids should be strings');
8       let myEvent = myMusicDocument.spine[eventId];
9       assert.ok(myEvent instanceof EventPrototype, 'events values
            should be objects');
10      assert.deepEqual(typeof (myEvent.absoluteTime), typeof (0),
            'absoluteTime should be a number if inizialized');
11      assert.deepEqual(typeof (myEvent.relativeTime), typeof (0),
            'relativeTime should be a number if inizialized');
12    }
13  });
```

spineIds Verifica che l'attributo *spineIds* contenga array di stringhe.

```
1  QUnit.test("spineIds", function (assert) {
2    // myMusicDocument.spineIds
3    assert.ok(myMusicDocument.spineIds instanceof Array, '
        spineIds should be an array if initialized');
4    for (let i = 0; i < myMusicDocument.spineIds.length; i++) {
5      assert.deepEqual(typeof (myMusicDocument.spineIds[i]),
          typeof (""), 'spineIds should contain strings if not
          empty');
6    }
7  });
```

spineHash Verifica che l'attributo *spineHash* contenga oggetti le cui chiavi siano stringhe e i cui valori siano numerici.

```
1  QUnit.test("spineHash", function (assert) {
2    // myMusicDocument.spineHash
3    assert.deepEqual(typeof (myMusicDocument.spineHash), typeof
        ({}), 'spineHash should be an object if initialized');
4    let spineHashes = Object.keys(myMusicDocument.spineHash);
5    for (let i = 0; i < spineHashes.length; i++) {
6      let eventId = spineHashes[i];
7      assert.deepEqual(typeof (eventId), typeof (""), 'spineHash
          eventids should be strings');
8      let eventNumber = myMusicDocument.spineHash[eventId];
9      assert.deepEqual(typeof (eventNumber), typeof (0), '
          spineHash eventNumber should be a number');
10   }
11 });
```

staves Verifica che l'attributo *staves* contenga array di stringhe.

```
1  QUnit.test("staves", function (assert) {
2    // myMusicDocument.staves
3    assert.ok(myMusicDocument.staves instanceof Array, 'staves
        should be an array if initialized');
4    for (let i = 0; i < myMusicDocument.staves.length; i++) {
5      assert.deepEqual(typeof (myMusicDocument.staves[i]), typeof
          (""), 'staves should contain strings if not empty');
6    }
7  });
```

parts Verifica che l'attributo *parts* contenga array di stringhe.

```
1  QUnit.test("parts", function (assert) {
2    // myMusicDocument.parts
3    assert.ok(myMusicDocument.parts instanceof Array, 'parts
        should be an array if initialized');
```

```
4     for (let i = 0; i < myMusicDocument.parts.length; i++) {
5       assert.deepEqual(typeof (myMusicDocument.parts[i]), typeof
            (""), 'parts should contain strings if not empty');
6     }
7   });
```

measures Verifica che l'attributo *measures* contenga oggetti le cui chiavi siano stringhe e i cui valori siano array di stringhe.

```
1   QUnit.test("measures", function (assert) {
2     // myMusicDocument.measures
3     assert.deepEqual(typeof (myMusicDocument.measures), typeof
          ({}), 'measures should be an object if initialized');
4     let measure = Object.keys(myMusicDocument.measures);
5     for (let i = 0; i < measure.length; i++) {
6       let measureId = measure[i];
7       assert.deepEqual(typeof (measureId), typeof (""), 'measures
            keys should be strings');
8       let values = myMusicDocument.measures[measureId];
9       assert.ok(values instanceof Array, 'measures values should
            be arrays');
10      for (let j = 0; j < values.length; j++) {
11        assert.deepEqual(typeof (values[j]), typeof (""), '
              measures parts should be strings');
12      }
13    }
14  });
```

clefs Verifica che l'attributo *clefs* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *ClefPrototype*. Gli attributi degli oggetti *ClefPrototype* vengono controllati per verificare che siano coerenti:

- *clefshape* deve contenere una stringa;
- *clefstep* deve contenere valori di tipo numerico;
- *staffid*, *partid* e *measureid* devono contenere stringhe;
- se staffid non è popolato, lo devono essere partid+measureid (e viceversa).

```
1   QUnit.test("clefs", function (assert) {
2     // myMusicDocument.clefs
3     assert.deepEqual(typeof (myMusicDocument.clefs), typeof ({}),
          'clefs should be an object if initialized');
4     let clefs = Object.keys(myMusicDocument.clefs);
5     for (let i = 0; i < clefs.length; i++) {
```

```
6       let clefId = clefs[i];
7       assert.deepEqual(typeof (clefId), typeof (""), 'clefs ids
            should be strings');
8       let myClef = myMusicDocument.clefs[clefId];
9       assert.ok(myClef instanceof ClefPrototype, 'clefs values
            should be clef objects');
10      assert.deepEqual(typeof (myClef.clefshape), typeof (""), '
            clefshape should be a string if inizialized');
11      assert.deepEqual(typeof (myClef.clefstep), typeof (0), '
            clefstep should be a number if inizialized');
12      assert.ok(myClef.staffid !== undefined || (myClef.partid !
            == undefined && myClef.measureid !== undefined), '
            staffid or partid+measureid should be initialized');
13      if (myClef.staffid !== undefined) {
14        assert.deepEqual(typeof (myClef.staffid), typeof (""), '
              staffid should be a string if inizialized (optional
              attribute)');
15      }
16      if (myClef.partid !== undefined) {
17        assert.deepEqual(typeof (myClef.partid), typeof (""), '
              partid should be a string if inizialized (optional
              attribute)');
18      }
19      if (myClef.measureid !== undefined) {
20        assert.deepEqual(typeof (myClef.measureid), typeof (""),
              'measureid should be a string if inizialized (
              optional attribute)');
21      }
22    }
23  });
```

keys Verifica che l'attributo *keys* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *KeySignaturePrototype*. Gli attributi degli oggetti *KeySignaturePrototype* vengono controllati per verificare che siano coerenti:

- *fifths* deve contenere valori di tipo numerico;
- *accidentals* deve contenere un array di stringhe;
- almeno un elemento tra *fifths* e *accidentals* deve essere popolato;
- *staffid*, *partid* e *measureid* devono contenere stringhe;
- se staffid non è popolato, lo devono essere partid+measureid (e viceversa).

```
1  QUnit.test("keys", function (assert) {
2    // myMusicDocument.keys
3    assert.deepEqual(typeof (myMusicDocument.keys), typeof ({}),
        'keys should be an object if initialized');
4    let keys = Object.keys(myMusicDocument.keys);
5    for (let i = 0; i < keys.length; i++) {
6      let keyId = keys[i];
7      assert.deepEqual(typeof (keyId), typeof (""), 'keys ids
          should be strings');
8      let myKey = myMusicDocument.keys[keyId];
9      assert.ok(myKey instanceof KeySignaturePrototype, 'key
          values should be keysignature objects');
10     assert.ok(myKey.fifths !== undefined || myKey.accidentals !
          == undefined, 'fifths or accidentals should be
          initialized');
11     if (myKey.fifths !== undefined) {
12       assert.deepEqual(typeof (myKey.fifths), typeof (0), '
            fifths should be a number if inizialized (optional
            attribute)');
13     }
14     if (myKey.accidentals !== undefined) {
15       let accidentals = myKey.accidentals;
16       assert.ok(accidentals instanceof Array, 'accidentals
            should be an array if inizialized (optional attribute
            )');
17       for (let j = 0; j < accidentals.length; j++) {
18         assert.deepEqual(typeof (accidentals[j]), typeof (""),
              'accidentals items should be strings');
19       }
20     }
21     assert.ok(myKey.staffid !== undefined || (myKey.partid !==
          undefined && myKey.measureid !== undefined), 'staffid
          or partid+measureid should be initialized');
22     if (myKey.staffid !== undefined) {
23       assert.deepEqual(typeof (myKey.staffid), typeof (""), '
            staffid should be a string if inizialized (optional
            attribute)');
24     }
25     if (myKey.partid !== undefined) {
26       assert.deepEqual(typeof (myKey.partid), typeof (""), '
            partid should be a string if inizialized (optional
            attribute)');
27     }
28     if (myKey.measureid !== undefined) {
29       assert.deepEqual(typeof (myKey.measureid), typeof (""), '
            measureid should be a string if inizialized (optional
             attribute)');
30     }
31   }
```

```
32 });
```

times Verifica che l'attributo *times* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *TimeSignaturePrototype*. Gli attributi degli oggetti *TimeSignaturePrototype* vengono controllati per verificare che siano coerenti:

- *num*, *den* e *vtu* devono contenere valori di tipo numerico,
- *staffid*, *partid* e *measureid* devono contenere stringhe;
- se staffid non è popolato, lo devono essere partid+measureid (e viceversa).

```
1  QUnit.test("times", function (assert) {
2    // myMusicDocument.times
3    assert.deepEqual(typeof (myMusicDocument.times), typeof ({}),
         'times should be an object if initialized');
4    let times = Object.keys(myMusicDocument.times);
5    for (let i = 0; i < times.length; i++) {
6      let timeId = times[i];
7      assert.deepEqual(typeof (timeId), typeof (""), 'times ids
           should be strings');
8      let values = myMusicDocument.times[timeId];
9      assert.ok(values instanceof Array, 'times values should be
           arrays');
10     for (let j = 0; j < values.length; j++) {
11       let myTime = values[j];
12       assert.ok(myTime instanceof TimeSignaturePrototype, '
             times signatures should be timesignature objects');
13       assert.deepEqual(typeof (myTime.num), typeof (0), 'num
             should be a number if inizialized');
14       if (myTime.den !== undefined) {
15         assert.deepEqual(typeof (myTime.den), typeof (0), 'den
               should be a number if inizialized (optional
               attribute)');
16       }
17       if (myTime.vtu !== undefined) {
18         assert.deepEqual(typeof (myTime.vtu), typeof (0), 'vtu
               should be a number if inizialized (optional
               attribute)');
19       }
20       assert.ok(myTime.staffid !== undefined || (myTime.partid
             !== undefined && myTime.measureid !== undefined), '
             staffid or partid+measureid should be initialized');
21       if (myTime.staffid !== undefined) {
22         assert.deepEqual(typeof (myTime.staffid), typeof (""),
               'staffid should be a string if inizialized (
               optional attribute)');
```

```
23        }
24        if (myTime.partid !== undefined) {
25          assert.deepEqual(typeof (myTime.partid), typeof (""), '
              partid should be a string if inizialized (optional
              attribute)');
26        }
27        if (myTime.measureid !== undefined) {
28          assert.deepEqual(typeof (myTime.measureid), typeof ("")
              , 'measureid should be a string if inizialized (
              optional attribute)');
29        }
30      }
31    }
32  });
```

voices Verifica che l'attributo *voices* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *VoicePrototype* con attributi *staffid* e *partid* popolati da valori di tipo stringa.

```
1  QUnit.test("voices", function (assert) {
2    // myMusicDocument.voices
3    assert.deepEqual(typeof (myMusicDocument.voices), typeof ({})
       , 'voices should be an object if initialized');
4    let voices = Object.keys(myMusicDocument.voices);
5    for (let i = 0; i < voices.length; i++) {
6      let voiceId = voices[i];
7      assert.deepEqual(typeof (voiceId), typeof (""), 'voices ids
         should be strings');
8      let myVoice = myMusicDocument.voices[voiceId];
9      assert.ok(myVoice instanceof VoicePrototype, 'voices values
         should be voice objects');
10     assert.deepEqual(typeof (myVoice.staffid), typeof (""), '
         staffid should be a string if inizialized ');
11     assert.deepEqual(typeof (myVoice.partid), typeof (""), '
         partid should be a string if inizialized ');
12   }
13 });
```

rests Verifica che l'attributo *rests* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *ChordRestPrototype*. Gli attributi degli oggetti *ChordRestPrototype* vengono controllati per verificare che siano coerenti:

- *voice* e *measure* devono contenere stringhe;

- *durationNum*, *durationDen* e *augmentationDots*(opzionale) devono contenere valori numerici;
- *tuplet* deve contenere oggetti derivati da *TupletPrototype* composti da attributi contenenti valori numerici (*enterNum*, *enterDen*, *enterDots*, *inNum*, *inDen*, *inDots*).

```
1  QUnit.test("rests", function (assert) {
2    // myMusicDocument.rests
3    assert.deepEqual(typeof (myMusicDocument.rests), typeof ({}),
         'rests should be an object if initialized');
4    let rests = Object.keys(myMusicDocument.rests);
5    for (let i = 0; i < rests.length; i++) {
6      let restId = rests[i];
7      assert.deepEqual(typeof (restId), typeof (""), 'rests ids
           should be strings');
8      let myRest = myMusicDocument.rests[restId];
9      assert.ok(myRest instanceof ChordRestPrototype, 'rests
           values should be rest objects');
10     assert.deepEqual(typeof (myRest.voice), typeof (""), 'voice
           should be a string if inizialized');
11     assert.deepEqual(typeof (myRest.measure), typeof (""), '
           measure should be a string if inizialized');
12     assert.deepEqual(typeof (myRest.durationNum), typeof (0), '
           durationNum should be a number if inizialized');
13     assert.deepEqual(typeof (myRest.durationDen), typeof (0), '
           durationDen should be a number if inizialized');
14     if (myRest.tuplet !== undefined) {
15       myTuplet = myRest.tuplet;
16       assert.ok(myTuplet instanceof TupletPrototype, 'tuplet
             should be a TupletPrototype if inizialized (optional
             attribute)');
17       assert.deepEqual(typeof (myTuplet.enterNum), typeof (0),
             'enterNum should be a number if inizialized');
18       assert.deepEqual(typeof (myTuplet.enterDen), typeof (0),
             'enterDen should be a number if inizialized');
19       if (myTuplet.enterDots !== undefined) {
20         assert.deepEqual(typeof (myTuplet.enterDots), typeof
               (0), 'enterDots should be a number if inizialized (
               optional attribute)');
21       }
22       assert.deepEqual(typeof (myTuplet.inNum), typeof (0), '
             inNum should be a number if inizialized');
23       assert.deepEqual(typeof (myTuplet.inDen), typeof (0), '
             inDen should be a number if inizialized');
24       if (myTuplet.inDots !== undefined) {
25         assert.deepEqual(typeof (myTuplet.inDots), typeof (0),
             'inDots should be a number if inizialized (optional
              attribute)');
```

```
26        }
27      }
28      if (myRest.augmentationDots !== undefined) {
29        assert.deepEqual(typeof (myRest.augmentationDots), typeof
              (0), 'augmentationDots should be a number if
             inizialized (optional attribute)');
30      }
31    }
32  });
```

chords Verifica che l'attributo *chords* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *ChordRestPrototype*. Gli attributi di questo tipo di oggetto vengono controllati in maniera analoga a quanto avviene per *rest*, con l'aggiunta dell'attributo *notehead* che deve essere un array contenente oggetti derivati da *NoteheadPrototype*. Gli oggetti derivati da *NoteheadPrototype* vengono a loro volta controllati per verificare che gli attributi siano ben strutturati:

- *step* e *actualAccidental* devono contenere stringhe;
- *octave* deve contenere un valore numerico;
- *tie* deve essere di tipo booleano;
- *printedAccidentals* deve essere un array di stringhe.

```
1  QUnit.test("chords", function (assert) {
2    // myMusicDocument.chords
3    assert.deepEqual(typeof (myMusicDocument.chords), typeof ({})
         , 'chords should be an object if initialized');
4    let chords = Object.keys(myMusicDocument.chords);
5    for (let i = 0; i < chords.length; i++) {
6      let chordId = chords[i];
7      assert.deepEqual(typeof (chordId), typeof (""), 'chords ids
           should be strings');
8      let myChord = myMusicDocument.chords[chordId];
9      assert.ok(myChord instanceof ChordRestPrototype, 'chords
         values should be chord objects');
10     assert.deepEqual(typeof (myChord.voice), typeof (""), '
         voice should be a string if inizialized');
11     assert.deepEqual(typeof (myChord.measure), typeof (""), '
         measure should be a string if inizialized');
12     assert.deepEqual(typeof (myChord.durationNum), typeof (0),
         'durationNum should be a number if inizialized');
13     assert.deepEqual(typeof (myChord.durationDen), typeof (0),
         'durationDen should be a number if inizialized');
```

```
14      if (myChord.tuplet !== undefined) {
15        let myTuplet = myChord.tuplet;
16        assert.ok(myTuplet instanceof TupletPrototype, 'tuplet
              should be a tuplet object if inizialized (optional
              attribute)');
17        assert.deepEqual(typeof (myTuplet.enterNum), typeof (0),
              'enterNum should be a number if inizialized');
18        assert.deepEqual(typeof (myTuplet.enterDen), typeof (0),
              'enterDen should be a number if inizialized');
19        if (myTuplet.enterDots !== undefined) {
20          assert.deepEqual(typeof (myTuplet.enterDots), typeof
                (0), 'enterDots should be a number if inizialized (
                optional attribute)');
21        }
22        assert.deepEqual(typeof (myTuplet.inNum), typeof (0), '
              inNum should be a number if inizialized');
23        assert.deepEqual(typeof (myTuplet.inDen), typeof (0), '
              inDen should be a number if inizialized');
24        if (myTuplet.inDots !== undefined) {
25          assert.deepEqual(typeof (myTuplet.inDots), typeof (0),
                'inDots should be a number if inizialized (optional
                 attribute)');
26        }
27      }
28      if (myChord.augmentationDots !== undefined) {
29        assert.deepEqual(typeof (myChord.augmentationDots),
              typeof (0), 'augmentationDots should be a number if
              inizialized (optional attribute)');
30      }
31      assert.ok(myChord.noteheads instanceof Array, 'noteheads
            should be an array if inizialized');
32      for (let i = 0; i < myChord.noteheads.length; i++) {
33        let myNotehead = myChord.noteheads[i];
34        assert.ok(myNotehead instanceof NoteheadPrototype, '
              noteheads should contain noteheads objects');
35        assert.deepEqual(typeof (myNotehead.step), typeof (""), '
              step should be a string if inizialized');
36        assert.deepEqual(typeof (myNotehead.octave), typeof (0),
              'octave should be a number if inizialized');
37        assert.ok(myNotehead.printedAccidentals instanceof Array,
               'printedAccidentals should be an array if
              inizialized');
38        for (let i = 0; i < myNotehead.printedAccidentals.length;
               i++) {
39          assert.deepEqual(typeof (myNotehead.printedAccidentals[
                i]), typeof (""), 'printedAccidentals should
                contain strings if not empty');
40        }
```

```
41        assert.deepEqual(typeof (myNotehead.actualAccidental),
              typeof (""), 'actualAccidental should be a string if
              inizialized');
42        assert.deepEqual(typeof (myNotehead.tie), typeof (true),
              'tie should be a boolean if inizialized');
43      }
44    }
45  });
```

allHash Verifica che l'attributo *allHash* contenga oggetti le cui chiavi siano stringhe e i cui valori siano oggetti derivati dal prototipo *ChordRestPrototype*: gli attributi di quest'ultimo vengono controllati per verificare che siano coerenti (in maniera analoga a quanto già descritto per i test effettuati su *chord* e *rests*).

```
1  QUnit.test("allHash", function (assert) {
2    // myMusicDocument.allHash
3    assert.deepEqual(typeof (myMusicDocument.allHash), typeof
         ({}), 'allHash should be an object if initialized');
4    let hashes = Object.keys(myMusicDocument.allHash);
5    for (let i = 0; i < hashes.length; i++) {
6      let hashId = hashes[i];
7      assert.deepEqual(typeof (hashId), typeof (""), 'allHash ids
           should be strings');
8      let myHash = myMusicDocument.allHash[hashId];
9      assert.ok(myHash instanceof ChordRestPrototype, 'allHash
           values should be chordrest objects');
10     assert.deepEqual(typeof (myHash.voice), typeof (""), 'voice
           should be a string if inizialized');
11     assert.deepEqual(typeof (myHash.measure), typeof (""), '
           measure should be a string if inizialized');
12     assert.deepEqual(typeof (myHash.durationNum), typeof (0), '
           durationNum should be a number if inizialized');
13     assert.deepEqual(typeof (myHash.durationDen), typeof (0), '
           durationDen should be a number if inizialized');
14     if (myHash.tuplet !== undefined) {
15       let myTuplet = myHash.tuplet;
16       assert.ok(myTuplet instanceof TupletPrototype, 'tuplet
             should be a tuplet object if inizialized (optional
             attribute)');
17       assert.deepEqual(typeof (myTuplet.enterNum), typeof (0),
             'enterNum should be a number if inizialized');
18       assert.deepEqual(typeof (myTuplet.enterDen), typeof (0),
             'enterDen should be a number if inizialized');
19       if (myTuplet.enterDots !== undefined) {
20         assert.deepEqual(typeof (myTuplet.enterDots), typeof
               (0), 'enterDots should be a number if inizialized (
               optional attribute)');
21       }
```

```
22        assert.deepEqual(typeof (myTuplet.inNum), typeof (0), '
              inNum should be a number if inizialized');
23        assert.deepEqual(typeof (myTuplet.inDen), typeof (0), '
              inDen should be a number if inizialized');
24        if (myTuplet.inDots !== undefined) {
25          assert.deepEqual(typeof (myTuplet.inDots), typeof (0),
                'inDots should be a number if inizialized (optional
                 attribute)');
26        }
27      }
28      if (myHash.augmentationDots !== undefined) {
29        assert.deepEqual(typeof (myHash.augmentationDots), typeof
              (0), 'augmentationDots should be a number if
              inizialized (optional attribute)');
30      }
31      if (myHash.noteheads !== undefined) {
32        assert.ok(myHash.noteheads instanceof Array, 'noteheads
              should be an array if inizialized');
33        for (let i = 0; i < myHash.noteheads.length; i++) {
34          let myNotehead = myHash.noteheads[i];
35          assert.ok(myNotehead instanceof NoteheadPrototype, '
                noteheads should contain noteheads objects');
36          assert.deepEqual(typeof (myNotehead.step), typeof (""),
                 'step should be a string if inizialized');
37          assert.deepEqual(typeof (myNotehead.octave), typeof (0)
                , 'octave should be a number if inizialized');
38          assert.ok(myNotehead.printedAccidentals instanceof
                Array, 'printedAccidentals should be an array if
                inizialized');
39          for (let i = 0; i <
                myNotehead.printedAccidentals.length; i++) {
40            assert.deepEqual(typeof (
                  myNotehead.printedAccidentals[i]), typeof (""), '
                  printedAccidentals should contain strings if not
                  empty');
41          }
42          assert.deepEqual(typeof (myNotehead.actualAccidental),
                typeof (""), 'actualAccidental should be a string
                if inizialized');
43          assert.deepEqual(typeof (myNotehead.tie), typeof (true)
                , 'tie should be a boolean if inizialized');
44        }
45      }
46    }
47  });
```

stats Verifica che l'attributo *stats* contenga un oggetto derivato da *StatisticsPrototype*: i relativi attributi dovranno essere di tipo numerico (*numParts*, *numEvents*, *numChords*, *numRests*, *numMeasures*, *numNotes*), di tipo array (*numNotes*, *pitchClassWeighted*, *midiPitch*, *midiPitchWeighted*) e di tipo oggetto (*chordLength*, *noteLength*, *restLength*, *noteByLengthPC*).

```
1  QUnit.test("stats", function (assert) {
2    // myMusicDocument.stats
3    assert.ok(myMusicDocument.stats instanceof
         StatisticsPrototype, 'stats should be a statistics object
          if initialized');
4    assert.deepEqual(typeof (myMusicDocument.stats.numParts),
         typeof (0), 'numParts should be a number if initialized')
         ;
5    assert.deepEqual(typeof (myMusicDocument.stats.numEvents),
         typeof (0), 'numEvents should be a number if initialized'
         );
6    assert.deepEqual(typeof (myMusicDocument.stats.numChords),
         typeof (0), 'numChords should be a number if initialized'
         );
7    assert.deepEqual(typeof (myMusicDocument.stats.numRests),
         typeof (0), 'numRests should be a number if initialized')
         ;
8    assert.deepEqual(typeof (myMusicDocument.stats.numMeasures),
         typeof (0), 'numMeasures should be a number if
         initialized');
9    assert.deepEqual(typeof (myMusicDocument.stats.numNotes),
         typeof (0), 'numNotes should be a number if initialized')
         ;
10   assert.ok(myMusicDocument.stats.pitchClass instanceof Array,
         'numNotes should be an array if initialized');
11   assert.ok(myMusicDocument.stats.pitchClassWeighted instanceof
          Array, 'pitchClassWeighted should be an array if
         initialized');
12   assert.ok(myMusicDocument.stats.midiPitch instanceof Array, '
         midiPitch should be an array if initialized');
13   assert.ok(myMusicDocument.stats.midiPitchWeighted instanceof
         Array, 'midiPitchWeighted should be an array if
         initialized');
14   assert.deepEqual(typeof (myMusicDocument.stats.chordLength),
         typeof ({}), 'chordLength should be an object if
         initialized');
15   assert.deepEqual(typeof (myMusicDocument.stats.noteLength),
         typeof ({}), 'noteLength should be an object if
         initialized');
16   assert.deepEqual(typeof (myMusicDocument.stats.restLength),
         typeof ({}), 'restLength should be an object if
         initialized');
```

```
17    assert.deepEqual(typeof (myMusicDocument.stats.noteByLengthPC
          ), typeof ({}), 'noteByLengthPC should be an object if
          initialized');
18    let durations = Object.keys(
          myMusicDocument.stats.noteByLengthPC);
19    for (let i = 0; i < durations.length; i++) {
20      let duration = durations[i];
21      assert.deepEqual(typeof (duration), typeof (""), '
            noteByLengthPC durations should be strings');
22      let values = myMusicDocument.stats.noteByLengthPC[duration
            ];
23      assert.deepEqual(typeof (values), typeof ({}), '
            noteByLengthPC values should be objects');
24      let pitchKeys = Object.keys(values);
25      for (let j = 0; j < pitchKeys.length; j++) {
26        let noteKey = pitchKeys[j];
27        assert.deepEqual(typeof (noteKey), typeof (""), '
              noteByLengthPC noteKeys should be strings');
28        let noteCounter = values[noteKey];
29        assert.deepEqual(typeof (noteCounter), typeof (0), '
              noteByLengthPC notecounter for a pitch should be a
              number');
30      }
31    }
32  });
```

lyrics Verifica che l'attributo *lyrics* contenga oggetti con chiavi di tipo stringa e valori di tipo array contenenti oggetti derivati da *SyllablePrototype*; questi ultimi devono avere attributi con valori di tipo stringa(*startEventRef*, *endEventRef*, *text*) e boolano (*hyphen*).

```
1   QUnit.test("lyrics", function (assert) {
2     // myMusicDocument.lyrics
3     assert.deepEqual(typeof (myMusicDocument.lyrics), typeof ({})
          , 'lyrics should be an object if initialized');
4     let lyrics = Object.keys(myMusicDocument.lyrics);
5     for (let i = 0; i < lyrics.length; i++) {
6       let lyricId = lyrics[i];
7       assert.deepEqual(typeof (lyricId), typeof (""), 'lyrics
            keys should be strings');
8       let values = myMusicDocument.lyrics[lyricId];
9       assert.ok(values instanceof Array, 'lyrics values should be
              arrays');
10      for (let j = 0; j < values.length; j++) {
11        let mySyllable = values[j];
12        assert.ok(mySyllable instanceof SyllablePrototype, '
              lyrics arrays should be syllable objects');
```

```
13        assert.deepEqual(typeof (mySyllable.startEventRef),
              typeof (""), 'startEventRef should be a string if
              inizialized');
14        if (mySyllable.endEventRef !== undefined) {
15          assert.deepEqual(typeof (mySyllable.endEventRef),
                typeof (""), 'endEventRef should be a string if
                inizialized (optional attribute)');
16        }
17        assert.ok(mySyllable.hyphen === true | mySyllable.hyphen
              === false, 'hyphen should be true or false if
              inizialized');
18        assert.deepEqual(typeof (mySyllable.text), typeof (""), '
              text should be a string if inizialized');
19      }
20    }
21  });
```

graphicInstances Verifica che l'attributo *graphicInstances* contenga oggetti con chiavi di tipo stringa e valori di tipo array contenenti oggetti derivati da *GraphicInstancePrototype*. Gli attributi degli oggetti *GraphicInstancePrototype* vengono controllati per la corretta formattazione:

- *positionInGroup* deve contenere un valore numerico;
- *fileName* deve essere una stringa;
- *firstEventInPage* deve essere una funzione;
- *description* deve essere di tipo stringa;
- *graphicEvents* deve contenere oggetti con chiavi di tipo stringa e valori di tipo array contenenti oggetti derivati da *GraphicEventPrototype*; gli attributi di questi oggetti devono contenere valori di tipo numerico (*x0*, *y0*, *x1*, *y1*) e di tipo stringa (*highlightColor*, *description*).

```
1  QUnit.test("graphicInstances", function (assert) {
2    // myMusicDocument.graphicInstances
3    assert.deepEqual(typeof (myMusicDocument.graphicInstances),
         typeof ({}), 'graphicInstances should be an object if
         initialized');
4    let graphicInstances = Object.keys(
         myMusicDocument.graphicInstances);
5    for (let i = 0; i < graphicInstances.length; i++) {
6      let graphicInstanceId = graphicInstances[i];
7      assert.deepEqual(typeof (graphicInstanceId), typeof (""), '
           graphicInstances keys should be strings');
```

```
8       let values = myMusicDocument.graphicInstances[
            graphicInstanceId];
9       assert.ok(values instanceof Array, 'graphicInstances values
            should be arrays');
10      for (let j = 0; j < values.length; j++) {
11        let myGraphicInstance = values[j];
12        assert.ok(myGraphicInstance instanceof
              GraphicInstancePrototype, 'graphicInstances arrays
              should be GraphicInstance objects');
13        assert.deepEqual(typeof (
              myGraphicInstance.positionInGroup), typeof (0), '
              positionInGroup should be a number if inizialized');
14        assert.deepEqual(typeof (myGraphicInstance.fileName),
              typeof (""), 'fileName should be a string if
              inizialized');
15        assert.deepEqual(typeof (
              myGraphicInstance.firstEventInPage), typeof (Function
              ), 'firstEventInPage should be a function if
              inizialized');
16        if (myGraphicInstance.description !== undefined) {
17          assert.deepEqual(typeof (myGraphicInstance.description)
                , typeof (""), 'description should be a string if
                inizialized (optional attribute)');
18        }
19        assert.deepEqual(typeof (myGraphicInstance.graphicEvents)
              , typeof ({}), 'graphicEvents should be an object if
              inizialized');
20        let keys = Object.keys(myGraphicInstance.graphicEvents);
21        for (let i = 0; i < keys.length; i++) {
22          let key = keys[i];
23          assert.deepEqual(typeof (key), typeof (""), '
                graphicEvents keys should be strings');
24          let events = myGraphicInstance.graphicEvents[key];
25          assert.ok(events instanceof Array, 'graphicEvents
                values should be arrays');
26          for (let j = 0; j < events.length; j++) {
27            let myGraphicEvent = events[j];
28            assert.ok(events[j] instanceof GraphicEventPrototype,
                  'graphicEvents events should be GraphicEvent
                  objects');
29            assert.deepEqual(typeof (myGraphicEvent.x0), typeof
                  (0), 'x0 should be a number if inizialized');
30            assert.deepEqual(typeof (myGraphicEvent.y0), typeof
                  (0), 'y0 should be a number if inizialized');
31            assert.deepEqual(typeof (myGraphicEvent.x1), typeof
                  (0), 'x1 should be a number if inizialized');
32            assert.deepEqual(typeof (myGraphicEvent.y1), typeof
                  (0), 'y1 should be a number if inizialized');
33            if (myGraphicEvent.highlightColor !== undefined) {
```

```
34              assert.deepEqual(typeof (
                    myGraphicEvent.highlightColor), typeof (""), '
                    highlightColor should be a string if
                    inizialized');
35            }
36            if (myGraphicEvent.description !== undefined) {
37              assert.deepEqual(typeof (myGraphicEvent.description
                    ), typeof (""), 'description should be a string
                     if inizialized');
38            }
39          }
40        }
41      }
42    }
43  });
```

tracks Verifica che l'attributo *tracks* sia popolato da oggetti con chiavi di tipo stringa e valori composti da oggetti derivati da *TrackPrototype*. Ogni oggetto *TrackPrototype* viene controllato per verificare se i relativi attributi sono stati correttamente formattati (qualora inizializzati):

- *note* deve essere di tipo stringa (opzionale);
- *performers* deve essere un array di stringhe;
- *fileFormat* deve essere di tipo stringa;
- *trackEventsByRef* deve essere un oggetto composto da chiavi di tipo stringa e valori numerici;
- *trackEventsByTime* deve essere un oggetto composto da chiavi di tipo stringa e valori array di stringhe.

```
1  QUnit.test("tracks", function (assert) {
2    // myMusicDocument.tracks
3    assert.deepEqual(typeof (myMusicDocument.tracks), typeof ({})
        , 'tracks should be an object if initialized');
4    let tracks = Object.keys(myMusicDocument.tracks);
5    for (let i = 0; i < tracks.length; i++) {
6      let trackId = tracks[i];
7      assert.deepEqual(typeof (trackId), typeof (""), 'tracks ids
          should be strings');
8      let myTrack = myMusicDocument.tracks[trackId];
9      assert.ok(myTrack instanceof TrackPrototype, 'tracks values
          should be track objects');
10     if (myTrack.notes !== undefined) {
```

```
11        assert.deepEqual(typeof (myTrack.notes), typeof (""), '
              notes should be a string if inizialized (optional
              attribute)');
12      }
13      if (myTrack.performers !== undefined) {
14        assert.ok(myTrack.performers instanceof Array, '
              performers should be an array if inizialized (
              optional attribute)');
15        let size = myTrack.performers.length;
16        for (let i = 0; i < size; i++) {
17          assert.deepEqual(typeof (myTrack.performers[i]), typeof
                ("), 'performers should contain strings');
18        }
19      }
20      assert.deepEqual(typeof (myTrack.fileFormat), typeof (""),
            'fileFormat should be a string if inizialized');
21      assert.deepEqual(typeof (myTrack.trackEventsByRef), typeof
            ({}), 'trackEventsByRef should be an object if
            inizialized');
22      let keys = Object.keys(myTrack.trackEventsByRef);
23      for (let i = 0; i < keys.length; i++) {
24        let key = keys[i];
25        assert.deepEqual(typeof (key), typeof (""), '
              trackEventsByRef keys should be strings');
26        let events = myTrack.trackEventsByRef[key];
27        assert.deepEqual(typeof (events), typeof (0), '
              trackEventsByRef values should be numbers');
28      }
29      assert.deepEqual(typeof (myTrack.trackEventsByTime), typeof
            ({}), 'trackEventsByTime should be an object if
            inizialized');
30      keys = Object.keys(myTrack.trackEventsByTime);
31      for (let i = 0; i < keys.length; i++) {
32        let key = keys[i];
33        assert.deepEqual(typeof (key), typeof (""), '
              trackEventsByTime keys should be strings');
34        let events = myTrack.trackEventsByTime[key];
35        assert.ok(events instanceof Array, 'trackEventsByTime
              values should be arrays');
36        for (let j = 0; j < events.length; j++) {
37          assert.deepEqual(typeof (events[j]), typeof (""), '
                trackEventsByTime values should contain strings');
38        }
39      }
40    }
41  });
```

3.6 Funzioni di utilità

Questa sezione descrive l'implementazione di alcune funzioni, in parte mutuate dall'*ieee1599-framework*, di utilità trasversale al codice sorgente sviluppato.

attributeArray

La funzione *attributeArray* restituisce un Array composto dagli attributi (*attribute*) degli elementi afferenti ad una HTMLCollection (*collection*).

```
1  function attributeArray(collection, attribute) {
2    let result = [];
3    let size = collection.length;
4    for (let i = 0; i < size; i++) {
5      result[i] = collection[i].getAttribute(attribute);
6    }
7    return result;
8  }
```

makeHash

La funzione *makeHash* restituisce una collezione di oggetti con chiavi ricavate dagli attributi (*keyAttribute*) degli elementi afferenti ad una HTMLCollection (*collection*) e con valori calcolati da una funzione passata come parametro (*valueFunction*). Un parametro di tipo booleano (*isUnique*) determina se, in corrispondenza della chiave, il valore restituito dalla funzione verrà memorizzato in maniera univoca o aggiunto ad un Array di elementi.

```
1  function makeHash(collection, keyAttribute, valueFunction,
       isUnique) {
2    let result = {};
3    for (let i = 0; i < collection.length; i++) {
4      let key = collection[i].getAttribute(keyAttribute);
5      let value = valueFunction(i);
6      if (isUnique) {
7        result[key] = value;
8      }
9      else {
10       if (result[key] == undefined) {
11         result[key] = [];
12       }
13       result[key].push(value);
14     }
15   }
16   return result;
17 }
```

makePath

La funzione *makeHash* restituisce una stringa che è il risultato dell'operazione di sostituzione di tutte le occorrenze del carattere backslash '\' con il carattere slash '/' effettuata sulla stringa passata come parametro (*filename*).

```
1  function makePath(filename) {
2    return filename.replace(/\\/g, "/");
3  }
```

getChildTextContent

La funzione *getChildTextContent* restituisce una stringa che corrisponde al contenuto testuale di un determinato nodo (*children*) che viene cercato tra i figli di un certo elemento (*father*). Se il nodo figlio non esiste viene restituita una stringa vuota.

```
1  function getChildTextContent(father, children) {
2    if (father.getElementsByTagName(children).length > 0) {
3      if (father.getElementsByTagName(children)[0].textContent !
           == undefined) {
4        return father.getElementsByTagName(children)[0].
           textContent;
5      }
6      else {
7        return father.getElementsByTagName(children)[0].text;
8      }
9    }
10   else {
11     return "";
12   }
13 }
```

getChildIntTextContent

La funzione *getChildIntTextContent* restituisce un valore numerico di tipo *int* risultante dalla conversione del contenuto testuale di un determinato nodo (*children*) che viene cercato tra i figli di un certo elemento (*father*). Se il nodo figlio non esiste viene restituito il valore di default (*undefined*).

```
1  function getChildIntTextContent(father, children) {
2    if (father.getElementsByTagName(children).length > 0) {
3      if (father.getElementsByTagName(children)[0].textContent !
           == undefined) {
4        return parseInt(father.getElementsByTagName(children)[0].
           textContent);
5      }
```

```
6       else {
7         return parseInt(father.getElementsByTagName(children)[0].
              text);
8       }
9     }
10    return;
11  }
```

formatNumberLength

La funzione *formatNumberLength* restituisce una stringa di lunghezza determinata dal parametro *length*, composta dall'anteposizione di occorrenze del carattere '0' alla sequenza di caratteri ottenuta dalla conversione implicita del valore numerico (*num*) passato per parametro.

```
1  function formatNumberLength(num, length) {
2    var r = "" + num;
3    while (r.length < length) {
4      r = "0" + r;
5    }
6    return r;
7  }
```

splitPath

La funzione *splitPath* utilizza un'espressione regolare per suddividere una stringa che rappresenta il percorso completo di un file (*path*): vengono restituiti il nome del file (*filePart* caratteri che seguono l'ultimo slash della stringa) e il percorso escluso il nome del file (*dirPart*, caratteri da inizio stringa fino all'ultimo slash).

```
1  function splitPath(path) {
2    var dirPart, filePart;
3    path.replace(/^(.*\/)?([^/]*)$/, function (_, dir, file) {
4      dirPart = dir; filePart = file;
5    });
6    return { dirPart: dirPart, filePart: filePart };
7  }
```

dotEval

La funzione *dotEval* restituisce il moltiplicatore da applicare alla durata di una certa nota, calcolato sul numero di punti passato come parametro (*dotNumbers*).

```
1  function dotEval(dotNumbers) {
2    let result = 1;
```

```
3     for (let i = 1; i <= dotNumbers; i++) {
4       result += 2 ** (-i);
5     }
6     return result;
7   }
```

lcmArray

La funzione *lcmArray* restituisce il minimo comune multiplo tra i valori contenuti in un array passato come parametro (*arr*).

```
1   function lcmArray(arr) {
2     let rest1 = 0, rest2 = 0;
3     let l = arr.length;
4     for (let i = 0; i < l; i++) {
5       rest1 = arr[i] % arr[i + 1];
6       if (rest1 === 0) {
7         arr[i + 1] = (arr[i] * arr[i + 1]) / arr[i + 1];
8       }
9       else {
10        rest2 = arr[i + 1] % rest1;
11        if (rest2 === 0) {
12          arr[i + 1] = (arr[i] * arr[i + 1]) / rest1;
13        }
14        else {
15          arr[i + 1] = (arr[i] * arr[i + 1]) / rest2;
16        }
17      }
18    }
19    return arr[l - 1];
20  }
```

Capitolo 4

Implementazioni specifiche

Al fine di poter utilizzare concretamente le strutture dati introdotte nel Capitolo 3 si rende necessario costruirne delle specializzazioni. Si è quindi valutato di procedere all'implementazione di customizzazioni per importare le informazioni codificate con i formati file che sono stati trattati nel Capitolo 2, fattorizzando in un livello intermedio le funzionalità comuni. Nello specifico sono state implementate:

- strutture di prototipo intermedie per la fattorizzazione di codice per la gestione di informazioni provenienti da file in formato testuale generico;
- strutture concrete per la gestione di informazioni in formato Humdrum **kern;
- strutture di prototipo intermedie per la fattorizzazione di codice per la gestione di informazioni provenienti da file in formato XML generico;
- strutture concrete per la gestione di informazioni in formato IEEE 1599;
- strutture concrete per la gestione di informazioni in formato MusicXML.

In seguito a confronto con lo staff del LIM, si è concordato di adottare le seguenti convenzioni:

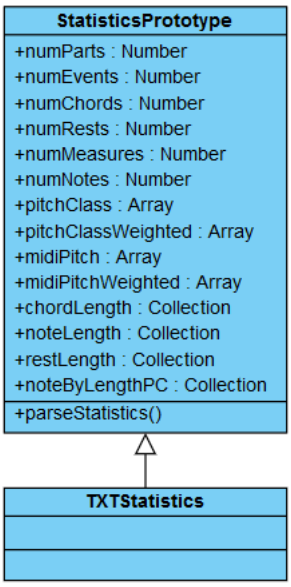
- tutti gli attributi delle strutture concrete derivate da *MusicDocumentPrototype* sono obbligatori: vanno popolati con elementi del tipo previsto (sono ammissibili elementi vuoti quando non è possibile determinare con certezza come valorizzare un attributo);
- tutti gli attributi non opzionali delle strutture concrete derivate da altri tipi di prototipo vanno popolati con elementi della tipologia prevista;
- tutti gli attributi opzionali delle altre strutture vengono inizializzati e valorizzati con elementi del tipo previsto solo se rappresentano informazione consistente (no elementi vuoti).

Per verificare che gli oggetti istanziati rispettino le caratteristiche previste, è possibile utilizzare la suite di test sviluppata a corredo del progetto e descritta nella Sezione 3.5.

4.1 File di testo generici

Questa sezione contiene la descrizione delle strutture di livello intermedio sviluppate per la rappresentazione di informazioni afferenti a file in formato testuale.

TXTStatistics È una struttura concreta che estende *StatisticsPrototype*.

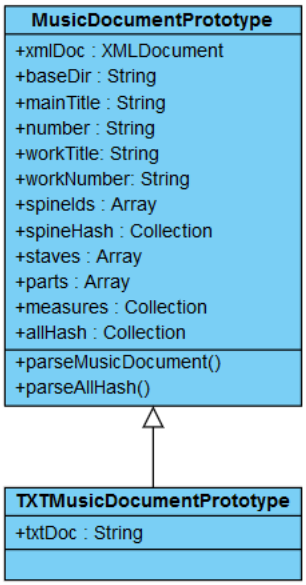


Il costruttore richiama l'omologa funzione del prototipo, con la sintassi *super*, passando come argomento un oggetto basato sul prototipo *TXTMusicDocumentPrototype*: gli attributi di questo oggetto vengono popolati dal metodo *parseStatistics*.

Listing 4.1: Definizione di TXTStatistics in JS

```
1 class TXTStatistics extends StatisticsPrototype {
2   constructor(txtMusicDocument) {
3     super();
4     this.parseStatistics(txtMusicDocument);
5   }
6 }
```

TXTMusicDocumentPrototype È una struttura astratta che estende le funzionalità di *MusicDocumentPrototype*.



Il costruttore istanzia gli attributi definiti nella struttura originale, assegna agli attributi *txtDoc* (specifico di questa implementazione, rappresenta il contenuto del file di testo originale) e *basedir* (percorso dal quale il file è stato caricato) i valori passati come parametri. L'attributo *xmlDoc* viene popolato con un documento vuoto come previsto dalle convenzioni. Infine viene invocato il metodo *parseMusicDocument* che sarà oggetto di override nelle strutture specialistiche: a runtime, grazie al polimorfismo e al dinamic-binding, verrà eseguito il codice specifico dell'effettiva struttura concreta.

Listing 4.2: Definizione di TXTMusicDocumentPrototype in JS

```
1 class TXTMusicDocumentPrototype extends MusicDocumentPrototype
    {
2   constructor(doc, url) {
3     super();
4     this.xmlDoc = new Document();
5     this.baseDir = splitPath(url).dirPart;
6     this.txtDoc = doc;
7     this.parseMusicDocument();
8   }
9 }
```

4.2 File Humdrum **kern

4.2.1 Strutture dati specialistiche

Questa sezione contiene la descrizione delle strutture di specializzazione utilizzate per rappresentare documenti musicali ottenuti a partire da file testuali Humdrum **kern (.krn) e dei metodi per popolarle.

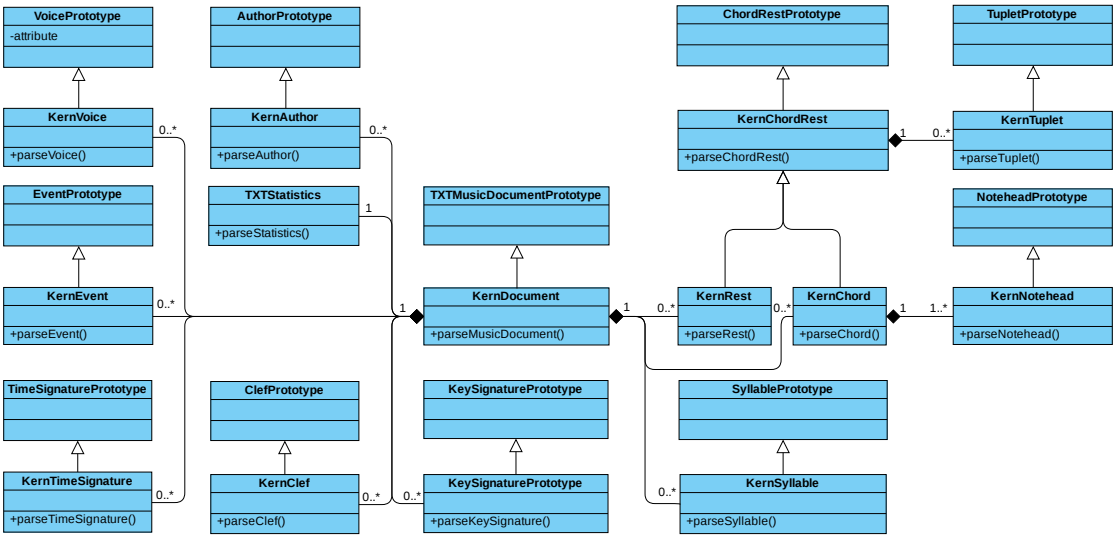
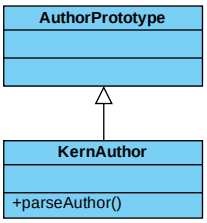


Figura 16: Specializzazione Humdrum **kern - rappresentazione UML

In Figura 16 sono rappresentati gli elementi che costituiscono questa implementazione: per compattezza non sono stati elencati per esteso gli attributi e i metodi delle strutture già descritte in precedenza. Nel prosieguo della sezione vengono illustrati in dettaglio gli elementi costitutivi.

KernAuthor È una struttura concreta che estende *AuthorPrototype*.

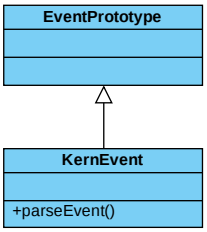


Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseAuthor* che li popola con i valori passati come parametri.

Listing 4.3: Definizione di KernAuthor in JS

```
1  class KernAuthor extends AuthorPrototype {
2    constructor(author, type) {
3      super();
4      this.parseAuthor(author, type);
5    }
6    parseAuthor(author, type) {
7      try {
8        if (author == undefined) {
9          return;
10       }
11       this.type = type;
12       this.name = author;
13     }
14     catch (error) {
15       console.log(error);
16     }
17   }
18 }
```

KernEvent È una struttura concreta che estende *EventPrototype*.



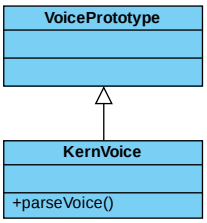
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseEvent* che li popola con i valori passati come parametri.

Listing 4.4: Definizione di KernEvent in JS

```
1  class KernEvent extends EventPrototype {
2    constructor(abstiming, reltiming) {
3      super();
4      this.parseEvent(abstiming, reltiming);
5    }
6    parseEvent(abstiming, reltiming) {
```

```
7       try {
8         this.absoluteTime = parseInt(abstiming);
9         this.relativeTime = parseInt(reltiming);
10      }
11      catch (error) {
12        console.log(error);
13      }
14    }
15  }
```

KernVoice È una struttura concreta che estende *VoicePrototype*.

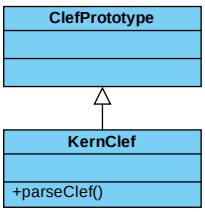


Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseVoice* che li popola con i valori passati come parametri.

Listing 4.5: Definizione di KernVoice in JS

```
1  class KernVoice extends VoicePrototype {
2    constructor(staffid, partid) {
3      super();
4      this.parseVoice(staffid, partid);
5    }
6    parseVoice(staffid, partid) {
7      try {
8        this.staffid = staffid;
9        this.partid = partid;
10     }
11     catch (error) {
12       console.log(error);
13     }
14   }
15 }
```

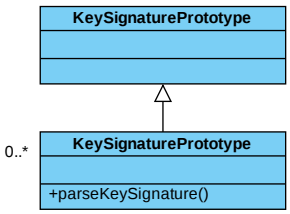
KernClef È una struttura concreta che estende *ClefPrototype*.
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseClef* che li popola con i valori passati come parametri.



Listing 4.6: Definizione di KernClef in JS

```
1  class KernClef extends ClefPrototype {
2    constructor(clefshape, clefstep, staffid, partid, measureid)
       {
3      super();
4      this.parseClef(clefshape, clefstep, staffid, partid,
         measureid);
5    }
6    parseClef(clefshape, clefstep, staffid, partid, measureid) {
7      try {
8        this.clefshape = clefshape;
9        this.clefstep = parseInt(clefstep);
10       if (staffid !== undefined) {
11         this.staffid = staffid;
12       }
13       if (partid !== undefined) {
14         this.partid = partid;
15       }
16       if (measureid !== undefined) {
17         this.measureid = measureid;
18       }
19     }
20     catch (error) {
21       console.log(error);
22     }
23   }
24 }
```

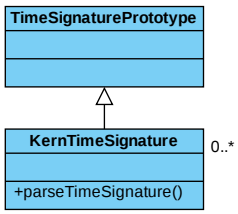
KernKeySignature È una struttura concreta che estende *KeySignaturePrototype*. Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseKeySignature* che li popola con i valori passati come parametri.



Listing 4.7: Definizione di KernKeySignature in JS

```
1  class KernKeySignature extends KeySignaturePrototype {
2    constructor(fifths, accidentals, staffid, partid, measureid)
       {
3      super();
4      this.parseKeySignature(fifths, accidentals, staffid, partid
         , measureid);
5    }
6    parseKeySignature(fifths, accidentals, staffid, partid,
       measureid) {
7      try {
8        if ((fifths !== undefined) && (fifths !== null)) {
9          this.fifths = parseInt(fifths);
10       }
11       if (accidentals !== undefined) {
12         this.accidentals = accidentals;
13       }
14       if (staffid !== undefined) {
15         this.staffid = staffid;
16       }
17       if (partid !== undefined) {
18         this.partid = partid;
19       }
20       if (measureid !== undefined) {
21         this.measureid = measureid;
22       }
23     }
24     catch (error) {
25       console.log(error);
26     }
27   }
28 }
```

KernTimeSignature È una struttura concreta che estende *TimeSignature-Prototype*.

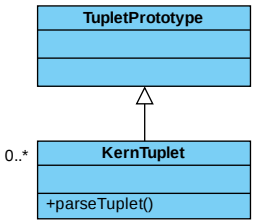


Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseTimeSignature* che li popola con i valori passati come parametri.

Listing 4.8: Definizione di KernTimeSignature in JS

```
1  class KernTimeSignature extends TimeSignaturePrototype {
2    constructor(num, den, vtu, staffid, partid, measureid) {
3      super();
4      this.parseTimeSignature(num, den, vtu, staffid, partid,
           measureid);
5    }
6    parseTimeSignature(num, den, vtu, staffid, partid, measureid)
         {
7      try {
8        this.num = parseInt(num);
9        if ((den !== undefined) && (den !== null)) {
10         this.den = parseInt(den);
11       }
12       if ((vtu !== undefined) && (vtu !== null)) {
13         this.vtu = parseInt(vtu);
14       }
15       if (staffid !== undefined) {
16         this.staffid = staffid;
17       }
18       if (partid !== undefined) {
19         this.partid = partid;
20       }
21       if (measureid !== undefined) {
22         this.measureid = measureid;
23       }
24     }
25     catch (error) {
26       console.log(error);
27     }
28   }
29 }
```

KernTuplet È una struttura concreta che estende *TupletPrototype*, per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.

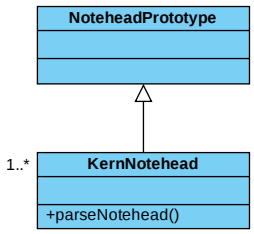


Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseTuplet* che li popola con i valori passati come parametri.

Listing 4.9: Definizione di KernTuplet in JS

```
1  class KernTuplet extends TupletPrototype {
2    constructor(enterNum, enterDen, enterDots, inNum, inDen,
       inDots) {
3      super();
4      this.parseTuplet(enterNum, enterDen, enterDots, inNum,
         inDen, inDots);
5    }
6    parseTuplet(enterNum, enterDen, enterDots, inNum, inDen,
       inDots) {
7      try {
8        this.enterNum = enterNum;
9        this.enterDen = enterDen;
10       if (enterDots !== null) {
11         this.enterDots = enterDots;
12       }
13       this.inNum = inNum;
14       this.inDen = inDen;
15       if (inDots !== null) {
16         this.inDots = inDots;
17       }
18     }
19     catch (error) {
20       console.log(error);
21     }
22   }
23 }
```

KernNotehead È una struttura concreta che estende *NoteheadPrototype*, per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.



Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseNotehead* che li popola: la stringa passata come parametro (*stop*) viene confrontata con una serie di espressioni regolari, redatte in base alla sintassi di Humdrum **kern, per estrarre le sottostringhe corrispondenti alle informazioni da codificare; il formato delle informazioni viene quindi standardizzato secondo le modalità di rappresentazione previste dal modello

- l'altezza delle note viene convertita in caratteri maiuscoli in codifica anglosassone;
- le ottave vengono espresse con un valore numerico;
- le alterazioni vengono indicate per esteso;
- la presenza di legatura di valore viene espressa con valore booleano.

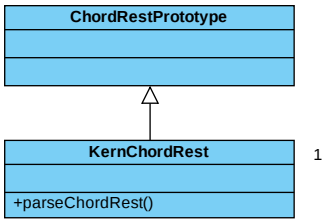
Per i dettagli implementativi si può fare riferimento al seguente frammento di codice.

Listing 4.10: Definizione di KernNotehead in JS

```
1  class KernNotehead extends NoteheadPrototype {
2    constructor(stop) {
3      super();
4      this.parseNotehead(stop);
5    }
6    parseNotehead(stop) {
7      try {
8        this.step;
9        this.octave;
10       this.printedAccidentals = [];
11       this.actualAccidental = "natural";
12       let stepsUp = stop.match(/[A-G]/g);
13       let stepsDown = stop.match(/[a-g]/g);
14       this.step = stepsUp ? stepsUp[0] : stepsDown[0].
             toUpperCase();
```

```
15        this.octave = 5 + (stepsUp ? stepsUp.length - 1 :
              -stepsDown.length);
16        let accidentals = stop.match(/[#|n|\-]/g);
17        if (accidentals) {
18          switch (accidentals[0]) {
19            case "#": accidentals.length === 1 ?
                  this.actualAccidental = "sharp" :
                  this.actualAccidental = "double_sharp"; break;
20            case "n": this.actualAccidental = "natural"; break;
21            case "-": accidentals.length === 1 ?
                  this.actualAccidental = "flat" :
                  this.actualAccidental = "double_flat";
22          }
23        }
24        this.tie = stop.match(/\[/) ? true : false;
25      }
26      catch (error) {
27        console.log(error);
28      }
29    }
30  }
```

KernChordRest È una struttura astratta che estende *ChordRestPrototype*.



Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseChordRest* che li popola sulla base dei valori passati come parametri. Al fine di codificare la durata secondo la notazione prevista per il modello (frazione), il relativo valore **kern viene confrontato con una lookup table di conversione (denominata *notesMap*): se la entry risulta presente in tabella, vengono estratti i corrispondenti valori di numeratore e denominatore. Per converso, se la ricerca nella LUT fallisce, si presuppone che il valore rappresenti la durata di un elemento appartenente ad un gruppo irregolare (per eccesso) che viene trattato applicando i seguenti step di pseudocodice:

- ricerca del più piccolo divisore d non potenza di 2 (le durate delle note standard sono espresse come potenze di 2);

- si ricava $r = \frac{durata}{d}$: saranno cioè presenti d divisioni della nota di durata $\frac{1}{r}$;
- si ricavano *innum* e *inden* cercando nella lookuptable le note corrispondenti a 2*r: il numeratore avrà il doppio del valore estratto dalla tabella;
- ricerca nella LUT della nota v il cui valore di durata sia inferiore a quello di duration;
- *enterNum*=d e *enterDen*=v;
- ricerca nella lookup table di v per ottenere i valori di *durationNum* e *durationDen*;
- l'eventuale presenza di punti viene codificata in *indots*;
- costruzione di un oggetto *KernTuplet* passando come parametri i valori ottenuti.

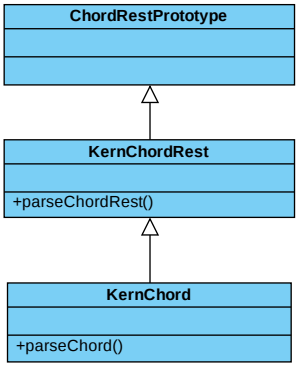
Esempio: una duration di 24 corrisponde alla terzina di sedicesimi, cioè ad una nota della durata di $\frac{1}{24}$ risultante dalla suddivisione in $d = 3$ parti uguali di una nota da $\frac{1}{8}$ ($r = 8$): saranno quindi presenti 3 note (*enterNum=d=3*, *enterDen=v=16*) da $\frac{1}{16}$ (*durationNum=1*, *durationDen=16*) nella durata normale di $\frac{2}{16}$ (*innum=2*, *inden=16*).

Listing 4.11: Definizione di KernChordRest in JS

```
1  class KernChordRest extends ChordRestPrototype {
2    constructor(voice, measure, duration, dots) {
3      super();
4      this.parseChordRest(voice, measure, duration, dots);
5    }
6    parseChordRest(voice, measure, duration, dots) {
7      try {
8        const notesMap = {
9          0: [8, 4],
10         1: [4, 4],
11         2: [2, 4],
12         4: [1, 4],
13         8: [1, 8],
14         16: [1, 16],
15         32: [1, 32],
16         64: [1, 64],
17         128: [1, 128]
18       };
19       this.voice;
20       this.measure;
21       this.durationNum;
22       this.durationDen;
```

```
23          if (voice !== undefined) {
24            this.voice = voice;
25          }
26          this.measure = measure;
27          if (duration in notesMap) {
28            this.durationNum = parseInt(notesMap[duration][0]);
29            this.durationDen = parseInt(notesMap[duration][1]);
30            if (dots !== null) {
31              this.augmentationDots = dots.length;
32            }
33          }
34          else {
35            // tuplet
36            let d = 3;
37            for (let i = d; duration % d !== 0; i++) {
38              if (Math.log2(i) % 1 !== 0)
39                d = i;
40            }
41            let r = duration / d;
42            let inNum = 2 * parseInt(notesMap[2 * r][0]);
43            let inDen = parseInt(notesMap[2 * r][1]);
44            let dur = Object.keys(notesMap);
45            let v = 0;
46            for (let i = 1; i < dur.length; i++) {
47              if (parseInt(dur[i]) > duration) {
48                v = dur[i - 1];
49                break;
50              }
51            }
52            let enterNum = d;
53            let enterDen = parseInt(v);
54            this.durationNum = parseInt(notesMap[v][0]);
55            this.durationDen = parseInt(notesMap[v][1]);
56            let inDots;
57            if (dots !== null) {
58              inDots = dots.length;
59            }
60            this.tuplet = new KernTuplet(enterNum, enterDen,
                undefined, inNum, inDen, inDots);
61          }
62        }
63        catch (error) {
64          console.log(error);
65        }
66      }
67  }
```

KernChord È una struttura concreta che estende *KernChordRest*.



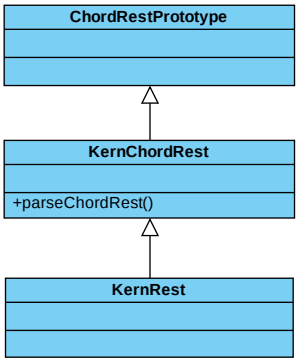
Il costruttore richiama quello del prototipo ed istanzia gli attributi ereditati, popolandoli con i valori passati per parametro (secondo le logiche descritte nel paragrafo precedente). Quindi invoca i metodi *parseChord* e *pushNoteheads*, definiti nella specializzazione della struttura, per istanziare la lista delle note (*notehead*) e popolarla con un oggetto di tipo *KernNotehead*. Il metodo *pushNoteheads* è invocabile, anche in un tempo diverso da quello della creazione di oggetti *KernChord*, per aggiungere note ad accordi esistenti.

Listing 4.12: Definizione di KernChord in JS

```
1  class KernChord extends KernChordRest {
2    constructor(voice, measure, duration, dots, stop) {
3      super(voice, measure, duration, dots);
4      this.parseChord();
5      this.pushNoteheads(stop);
6    }
7    parseChord() {
8      try {
9        this.noteheads = [];
10     }
11     catch (error) {
12       console.log(error);
13     }
14   }
15   pushNoteheads(stop) {
16     try {
17       this.noteheads.push(new KernNotehead(stop));
18     }
19     catch (error) {
```

```
20          console.log(error);
21        }
22      }
23  }
```

KernRest È una struttura concreta che estende *KernChordRest*.



Il costruttore si limita a richiamare l'omologa funzione del prototipo passando i parametri per il popolamento degli attributi.

Listing 4.13: Definizione di KernRest in JS

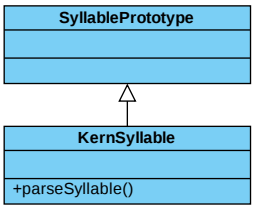
```
1  class KernRest extends KernChordRest {
2    constructor(voice, measure, duration, dots) {
3      super(voice, measure, duration, dots);
4    }
5  }
```

KernSyllable È una struttura concreta che estende il prototipo *SyllablePrototype*.

Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseSyllable* che li popola con i valori passati come parametri.

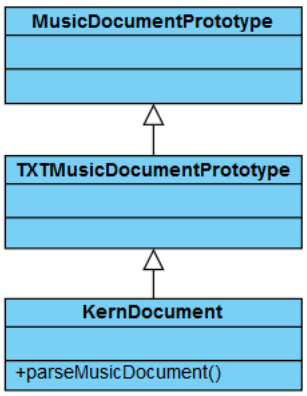
Listing 4.14: Definizione di KernSyllable in JS

```
1  class KernSyllable extends SyllablePrototype {
2    constructor(startEventRef, endEventRef, hyphen, text) {
3      super();
4      this.parseSyllable(startEventRef, endEventRef, hyphen, text
         );
```




```
5     }
6     parseSyllable(startEventRef, endEventRef, hyphen, text) {
7       try {
8         this.startEventRef = startEventRef;
9         if (endEventRef !== undefined)
10          this.endEventRef = endEventRef;
11        this.hyphen = hyphen;
12        this.text = text;
13      }
14      catch (error) {
15        console.log(error);
16      }
17    }
18  }
```

KernDocument È una struttura di specializzazione che estende le funzionalità di *TXTDocumentPrototype*.



Il costruttore richiama l'omologa funzione di *TXTMusicDocumentPrototype* con la sintassi *super*: vengono istanziati gli attributi, ereditati dalla catena di prototipi, i

quali vengono popolati dall'esecuzione del metodo di parsing *parseMusicDocument* descritto nella seguente Sezione 4.2.2.

Listing 4.15: Definizione di KernDocument in JS

```
1  class KernDocument extends TXTMusicDocumentPrototype {
2    constructor(doc, url) {
3      super(doc, url);
4    }
5    parseMusicDocument() {
6      try {
7        [...]
8      }
9      catch (error) {
10       console.log(error);
11     }
12   }
13 }
```

4.2.2 Algoritmo di parsing file

In questa sezione viene analizzata l'implementazione del metodo *parseMusicDocument* della struttura dati *KernDocument*: il codice sorgente è consultabile in Appendice A.1.

Le informazioni contenute in un documento Humdrum **kern sono strutturate in forma tabellare, come descritto nella Sezione 2.3. I file di testo vengono letti riga per riga: da ogni record di testo vengono estratti i token mediante split eseguito con il carattere di tabulazione. I token ottenuti vengono filtrati sulla base delle tipologie di record di appartenenza:

- Comment records - token che iniziano con il carattere '!';
- Interpretation records - token che iniziano con il carattere '*';
- Data records - token di altro tipo che appartengono ad uno spine rilevante ai fini del parsing.

Comment records I token che ricadono nella prima tipologia vengono ulteriormente vagliati, con l'utilizzo di espressioni regolari, per verificare la presenza di reference records (token di dimensioni pari a quella del record, che iniziano con la sequenza '!!!:'): quando si riscontra un record di questo tipo, il token viene suddiviso nella parte che identifica il tipo di reference record (*referenceCode*) e in quella che ne descrive il valore (*referenceValue*). Il *referenceCode* viene quindi verificato

e, se matcha una tipologia di interesse, gli attributi del documento vengono popolati con i valori del caso. La Tabella 2 mostra l'associazione tra attributi del documento e tipi di record.

Tabella 2: Mapping tra referenceCode e attributi

attributo	**referenceCode**
mainTitle	OTL
number	ONM
workTitle	OPR
workNumber	OMV / OPS
authors	vedi Tabella 3

La lista degli autori viene popolata con oggetti di tipo *KernAuthor*, sulla base delle informazioni estratte dai reference record, secondo le logiche rappresentate in Tabella 3.

Tabella 3: Mapping tra referenceCode e tipologia di autore

tipologia di contributo	**referenceCode**
Composer	COM
Attributed Composer	COA
Suspected Composer	COS
Composer abbreviated	COL
Composer(s) corporate name	COC
Lyricist	LYR
Librettist	LIB
Arranger	LAR
Orchestrator	LOR
Translator	TRN

Interpretation records I token che risultano appartenenti a questa seconda tipologia vengono dapprima verificati per controllare se contengono gli exclusive records ***kern* o ***text*/***silbe*:

- ***kern* - la presenza di questo token indica uno spine di interesse per il parsing: vengono generati nuovi id per la parte, sparito e voce; gli attributi relativi del documento (*parts*, *staves*, *voices*) vengono popolati con gli id generati (per la nuova voce viene associato un oggetto *KernVoice*). Per mantenere aggiornato il mapping tra parti di interesse e colonne del file, che evolvono dinamicamente con gli *spine-paths*, viene utilizzata una struttura

di supporto (*parts*) la cui cella j-esima contiene il nome della parte e il cui indice corrisponde a quello del j-esimo spine del file (le celle corrispondenti a spine non rilevanti avranno valore *undefined*). Analogamente, per tracciare il numero di voci di una parte, viene introdotto l'array di supporto *voices* che contiene, in corrispondenza della cella j-esima, un contatore del numero di voci per quella parte (impostato, in questa fase, a 1). Lo stesso avviene per il conteggio degli eventi di una determinata misura afferenti ad una parte: il contatore (*eventsCounters*) viene impostato a 0 nella j-esima cella relativa;

- ***text/**silbe* - la presenza di questo token indica uno spine di interesse che conterrà informazioni sul testo cantato. Viene generato un nuovo LyricId da usarsi per aggiornare la struttura delle parti rilevanti (*parts*) e per aggiungere all'attributo *lyrics* del documento una nuova lista, che conterrà la sillaba della parte cantata afferente allo spine in analisi.

Terminata questa verifica, si procede a controllare se il token afferisce alternativamente ad una chiave, ad un'armatura in chiave oppure ad un'indicazione di tempo. Gli step eseguiti al verificarsi di una delle tre casistiche sono simili:

- creazione di un nuovo id per l'evento utilizzando i contatori delle parti, delle misure e degli eventi;
- estrazione delle informazioni dal token con i metodi dell'oggetto *String*;
- aggiunta di un nuovo oggetto del tipo corrispondente (*KernClef*, *KernKeySignature* o *KernTimeSignature*) al rispettivo attributo del documento (*clefs*, *keys* o *times*);
- aggiunta alla cella i-esima della lista di supporto *spineArray* dei riferimenti all'evento (id associato e durata indefinita).

  È utile ricordare che il formato Humdrum **kern non prevede una temporizzazione esplicita degli eventi: questa va ricavata dalla struttura a matrice del file, trattando come concorrenti gli eventi associati ai token di un certo record e come consecutivi quelli afferenti a righe successive. La lista *spineArray*, citata al punto precedente, viene introdotta per gestire questa attività: ha dimensione pari alle righe del file di origine e contiene oggetti che descrivono le caratteristiche di durata degli eventi (id-evento, durata note/pause, presenza punti di valore).

Nel caso in cui non ci sia ancora stato un match, si verifica se il token contiene uno spine path indicator. Qualora si riscontrino indicatori di exchange, lo scambio di spine viene gestito operando direttamente sulle strutture di supporto: i contenuti delle celle corrispondenti agli indici marchiati per il change vengono scambiati

sul posto. Se ci si imbatte in un altro tipo di path indicator (new, add, split, delete), ci si avvale di un'ulteriore struttura di supporto (un array denominato *operations*) in modo da poter gestire in maniera corretta, al termine del parsing della riga corrente, tutte le attività richieste (cancellazioni/aggiunte/shift sulle strutture di supporto). Nel caso in cui si verifichi lo spilit o l'add di una parte, viene conseguentemente aggiunta una nuova *KernVoice* all'attributo relativo del documento.

Data records I token che non sono ricaduti nelle tipologie precedenti, vengono analizzati nell'ultimo step di filtraggio. Viene controllato se il contenuto descrive una misura (occorrenze del carattere '=' seguito da un identificativo), in qual caso viene estratto l'id corrispondente per l'aggiornamento dell'attributo *measures* dell'oggetto *KernDocument* e viene resettato il contatore *eventsCounters* della parte relativa.

Giunti a questo punto, se il token ha superato tutti i check precedenti senza essere intercettato, si presume che contenga informazioni afferenti ad eventi delle parti di interesse (note/pause o sillabe del testo cantato): il contatore degli eventi viene incrementato. Dato che un token può contenere più stop, viene eseguita un'operazione preliminare di split (utilizzando il carattere di spazio come separatore). Per ogni stop che non assolve alla funzione di placeholder (carattere '.') e che contiene informazioni su note/pause, vengono estratte le informazioni relative alla durata e alla presenza di punti di valore: sulla base di questi elementi, viene calcolata la durata effettiva dell'elemento per l'inserimento in una struttura di supporto di tipo *Set*, che contiene valori univoci, denominata *durations*. Se lo stop corrisponde ad una pausa, viene creato un oggetto KernRest per popolare l'attributo *rests* di KernDocument e lo *spineArray* viene aggiornato. Gli stessi passi vengono ripetuti per la creazione di un nuovo *KernChord* nel caso in cui si presenti una nota di un nuovo accordo. Qualora la nota risulti invece afferente ad un accordo preesistente (stop successivi al primo di un certo token nota), questa viene semplicemente aggiunta al *KernChord* di pertinenza invocando il relativo metodo *pushNoteheads*.

Infine, qualora non si tratti nè di note nè di pause, si verifica se il token contiene le sillabe del testo cantato: in caso affermativo, dopo aver verificato la presenza di trattini di separazione, le si aggiunge alla lista relativa in *lyrics* mediante la creazione di oggetti di tipo *KernSyllable*.

Al termine del parsing di tutti i record della rappresentazione testuale del file (contenuta in *txtDoc*), viene invocato il metodo *parseAllHash* dell'oggetto *KernDocument* e il relativo attributo del documento, *allHash*, viene popolato di conseguenza.

A questo punto avviene la fase di consolidamento degli eventi ai fini del popolamento degli attributi legati allo spine (inteso nell'accezione IEEE 1599):

- viene ottenuto il valore base da utilizzare per il calcolo della durata in VTU mediante la ricerca del minimo comune multiplo tra i valori presenti nel Set *durations*;
- per ogni evento contenuto in *spineArray* viene creato un nuovo oggetto *KernEvent*, da aggiungere alla lista di eventi dell'attributo *spine*, e viene aggiunto il relativo id a *spineIds*. I contatori temporali (delta + progressivo) vengono aggiornati;
- l'attributo *spineHash* viene popolato con i valori relativi.

L'algoritmo termina con la creazione di un nuovo oggetto *TXTStatistics* per la memorizzazione delle informazioni nell'attributo *stats*.

In Figura 17 sono rappresentati gli step dell'algoritmo descritti in forma di flowchart.

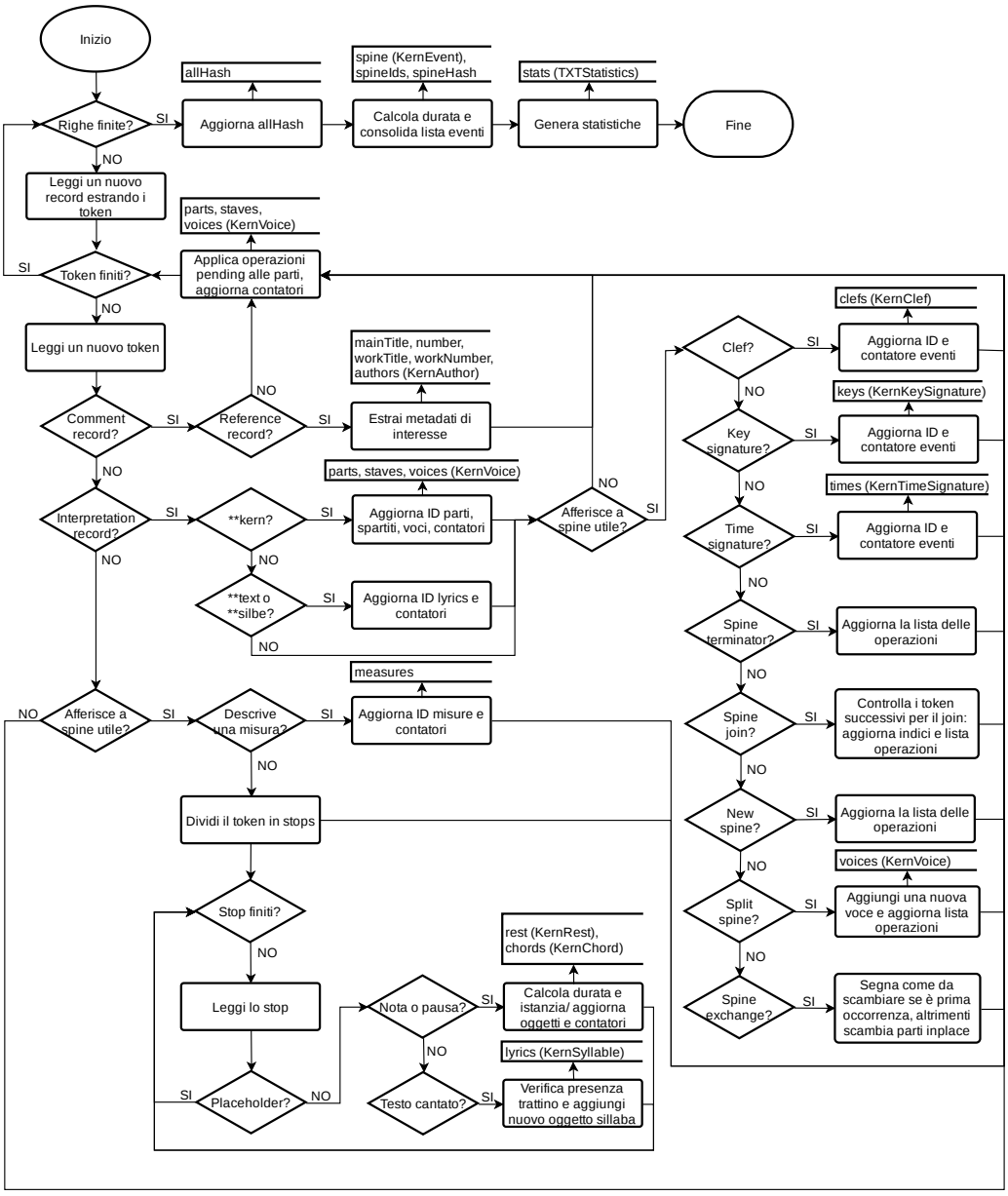
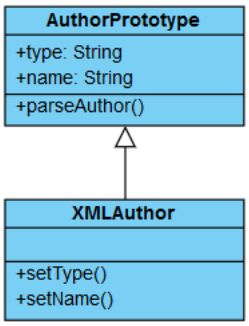


Figura 17: KernDocument: parseMusicDocument flowchart

4.3 File XML generici

Questa sezione contiene la descrizione delle strutture di livello intermedio sviluppate per la rappresentazione di informazioni afferenti a file in formato XML.

XMLAuthor È una struttura concreta che estende *AuthorPrototype*.

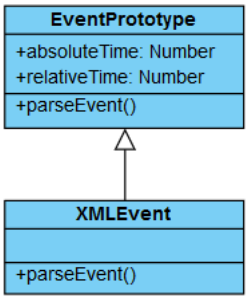


Fattorizza il codice comune alle implementazioni concrete trattate (MusicXML + IEEE 1599): il costruttore istanzia gli attributi, definiti nel prototipo, e richiama i metodi *setType* e *setName* che si occupano di popolare gli attributi estraendo le informazioni dall'oggetto passato come parametro

Listing 4.16: Definizione di XMLAuthor in JS

```
1  class XMLAuthor extends AuthorPrototype {
2    constructor(author) {
3      super();
4      this.setType(author);
5      this.setName(author);
6    }
7    setType(author) {
8      this.type = author.getAttribute("type");
9    }
10   setName(author) {
11     if (author.textContent != undefined) {
12       this.name = author.textContent;
13     }
14     else if (author.text != undefined) {
15       this.name = author.text;
16     }
17   }
18 }
```

XMLEvent È una struttura concreta che estende *EventPrototype*.

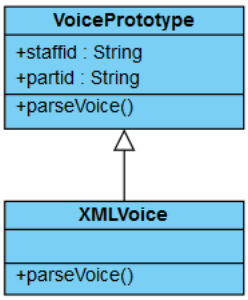


Fattorizza il codice comune alle implementazioni concrete trattate (MusicXML + IEEE 1599): il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseEvent* che li popola con i valori passati come parametri.

Listing 4.17: Definizione di XMLEvent in JS

```
1  class XMLEvent extends EventPrototype {
2    constructor(abstiming, reltiming) {
3      super();
4      this.parseEvent(abstiming, reltiming);
5    }
6    parseEvent(abstiming, reltiming) {
7      try {
8        this.absoluteTime = parseInt(abstiming);
9        this.relativeTime = parseInt(reltiming);
10     }
11     catch (error) {
12       console.log(error);
13     }
14   }
15 }
```

XMLVoice È una struttura concreta che estende *VoicePrototype*.

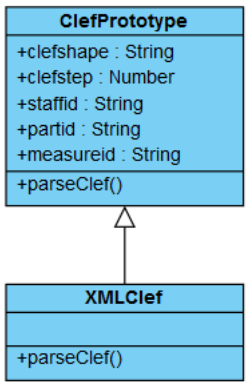


Fattorizza il codice comune alle implementazioni concrete trattate (MusicXML + IEEE 1599): il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseVoice* che li popola con i valori passati come parametri.

Listing 4.18: Definizione di XMLVoice in JS

```
1  class XMLVoice extends VoicePrototype {
2    constructor(staffid, partid) {
3      super();
4      this.parseVoice(staffid, partid);
5    }
6    parseVoice(staffid, partid) {
7      try {
8        this.staffid = staffid;
9        this.partid = partid;
10     }
11     catch (error) {
12       console.log(error);
13     }
14   }
15 }
```

XMLClef È una struttura concreta che estende *ClefPrototype*.

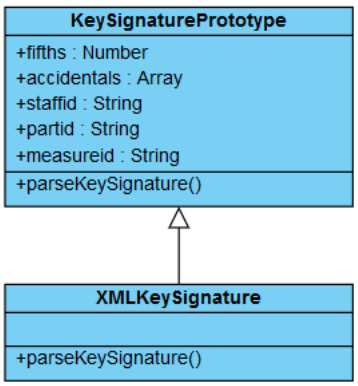


Fattorizza il codice comune alle implementazioni concrete trattate (MusicXML + IEEE 1599): il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseClef* che li popola con i valori passati come parametri.

Listing 4.19: Definizione di XMLClef in JS

```
1  class XMLClef extends ClefPrototype {
2    constructor(clefshape, clefstep, staffid, partid, measureid)
       {
3      super();
4      this.parseClef(clefshape, clefstep, staffid, partid,
         measureid);
5    }
6    parseClef(clefshape, clefstep, staffid, partid, measureid) {
7      try {
8        this.clefshape = clefshape;
9        this.clefstep = parseInt(clefstep);
10       if (staffid !== undefined) {
11         this.staffid = staffid;
12       }
13       if (partid !== undefined) {
14         this.partid = partid;
15       }
16       if (measureid !== undefined) {
17         this.measureid = measureid;
18       }
19     }
20     catch (error) {
21       console.log(error);
22     }
23   }
24 }
```

XMLKeySignature È una struttura concreta che estende *KeySignature-Prototype*.



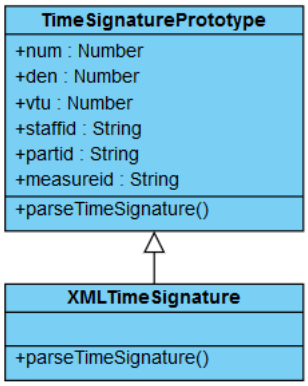
Fattorizza il codice comune alle implementazioni concrete trattate (MusicXML + IEEE 1599): il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseKeySignature* che li popola con i valori passati come parametri.

Listing 4.20: Definizione di XMLKeySignature in JS

```
class XMLKeySignature extends KeySignaturePrototype {
  constructor(fifths, accidentals, staffid, partid, measureid)
     {
    super();
    this.parseKeySignature(fifths, accidentals, staffid, partid
       , measureid);
  }
  parseKeySignature(fifths, accidentals, staffid, partid,
     measureid) {
    try {
      if ((fifths !== undefined) && (fifths !== null)) {
        this.fifths = parseInt(fifths);
      }
      if (accidentals !== undefined) {
        this.accidentals = accidentals;
      }
      if (staffid !== undefined) {
        this.staffid = staffid;
      }
      if (partid !== undefined) {
        this.partid = partid;
      }
      if (measureid !== undefined) {
        this.measureid = measureid;
      }
    }
    catch (error) {
      console.log(error);
    }
  }
}
```

XMLTimeSignature È una struttura concreta che estende *TimeSignature-Prototype*.

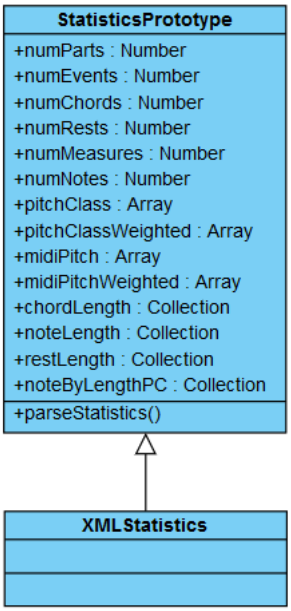
Fattorizza il codice comune alle implementazioni concrete trattate (MusicXML + IEEE 1599): il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseTimeSignature* che li popola con i valori passati come parametri.



Listing 4.21: Definizione di XMLTimeSignature in JS

```
1  class XMLTimeSignature extends TimeSignaturePrototype {
2    constructor(num, den, vtu, staffid, partid, measureid) {
3      super();
4      this.parseTimeSignature(num, den, vtu, staffid, partid,
           measureid);
5    }
6    parseTimeSignature(num, den, vtu, staffid, partid, measureid)
         {
7      try {
8        this.num = parseInt(num);
9        if ((den !== undefined) && (den !== null)) {
10         this.den = parseInt(den);
11       }
12       if ((vtu !== undefined) && (vtu !== null)) {
13         this.vtu = parseInt(vtu);
14       }
15       if (staffid !== undefined) {
16         this.staffid = staffid;
17       }
18       if (partid !== undefined) {
19         this.partid = partid;
20       }
21       if (measureid !== undefined) {
22         this.measureid = measureid;
23       }
24     }
25     catch (error) {
26       console.log(error);
27     }
28   }
29 }
```

XMLStatistics È una struttura concreta che estende *StatisticsPrototype*.

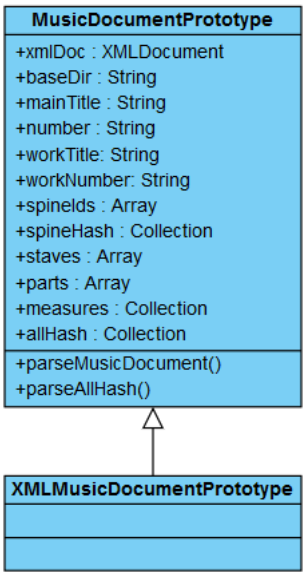


Il costruttore richiama l'omologa funzione del prototipo, con la sintassi *super*, passando come argomento un oggetto basato sul prototipo XMLMusicDocumentPrototype: gli attributi di questo oggetto vengono popolati dal metodo *parseStatistics*.

Listing 4.22: Definizione di XMLStatistics in JS

```
1 cclass XMLStatistics extends StatisticsPrototype {
2   constructor(xmlMusicDocument) {
3     super();
4     this.parseStatistics(xmlMusicDocument);
5   }
6 }
```

XMLMusicDocumentPrototype È una struttura astratta che estende le funzionalità di *MusicDocumentPrototype*.
Il costruttore istanzia gli attributi definiti nella struttura originale, assegna agli attributi *xmlDoc* (Document ricavato dal file XML) e *basedir* (percorso di caricamento del file) i valori passati come parametro e invoca il metodo *parseMusicDocument*. Quest'ultimo sarà oggetto di override nelle strutture specialistiche: a runtime, grazie al dinamic-binding, verrà eseguito il codice specifico dell'effettiva struttura concreta.



Listing 4.23: Definizione di XMLMusicDocumentPrototype in JS

```
1 cclass XMLMusicDocumentPrototype extends MusicDocumentPrototype
      {
2   constructor(doc, url) {
3     super();
4     this.xmlDoc = doc;
5     this.baseDir = splitPath(url).dirPart;
6     this.parseMusicDocument();
7   }
8 }
```

4.4 File IEEE 1599

4.4.1 Strutture dati specialistiche

Questa sezione contiene la descrizione delle strutture di specializzazione utilizzate per rappresentare documenti musicali ottenuti a partire da file XML in formato IEEE 1599 (.xml).

In Figura 18 sono rappresentati gli elementi che costituiscono questa implementazione: per compattezza non sono stati elencati per esteso gli attributi e i metodi delle strutture astratte già descritte in precedenza. Nel prosieguo della sezione vengono illustrati in dettaglio gli elementi costitutivi. Il codice che popola gli attributi degli oggetti è un adattamento di quello presente nell'*ieee1599-framework* (*IEEE1599.js*).

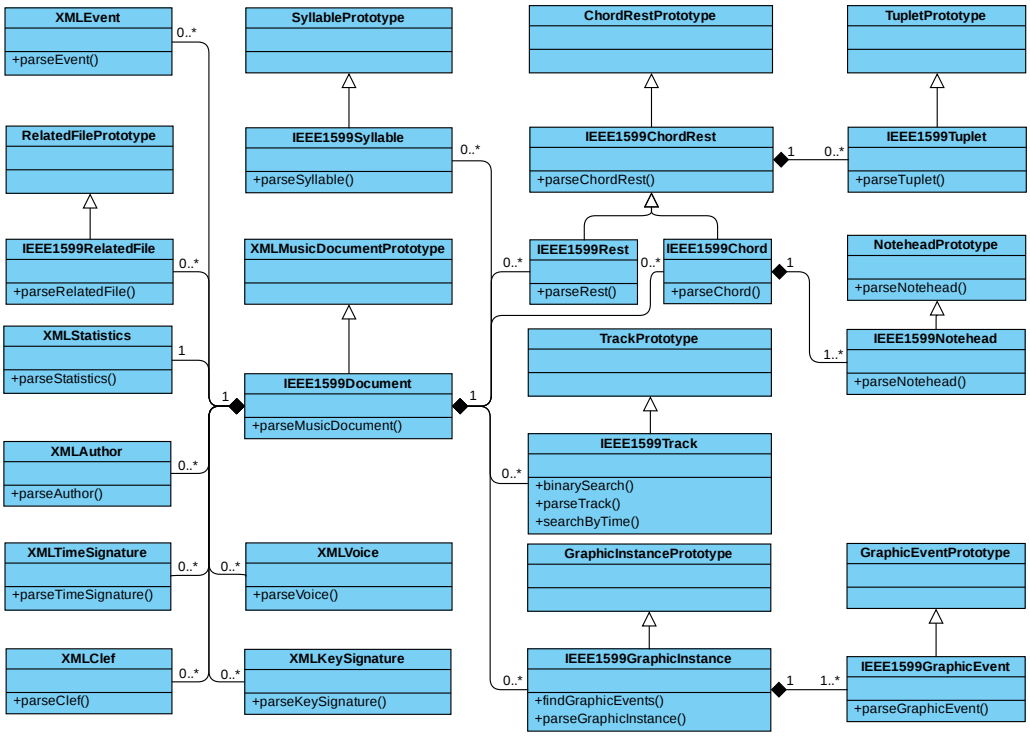
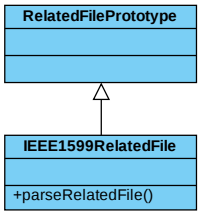


Figura 18: Specializzazione IEEE 1599 - rappresentazione UML

IEEE1599RelatedFile È una struttura concreta che estende *RelatedFile-Prototype*.

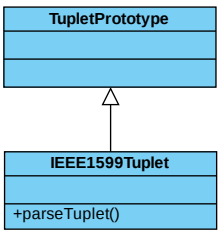


Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseRelatedFile* che li popola estraendo le informazioni necessarie dall'oggetto passato come parametro.

Listing 4.24: Definizione di IEEE1599RelatedFile in JS

```
1  class IEEE1599RelatedFile extends RelatedFilePrototype {
2    constructor(file) {
3      super();
4      this.parseRelatedFile(file);
5    }
6    parseRelatedFile(file) {
7      try {
8        if (file === undefined) {
9          return null;
10       }
11       this.fileName = makePath(file.getAttribute("file_name"));
12       this.description = file.getAttribute("description");
13       if (file.getAttribute("spine_start_ref") !== null) {
14         this.spineStartRef = file.getAttribute("spine_start_ref
               ");
15       }
16       if (file.getAttribute("spine_end_ref") !== null) {
17         this.spineEndRef = file.getAttribute("spine_end_ref");
18       }
19     }
20     catch (error) {
21       console.log(error);
22     }
23   }
24 }
```

IEEE1599Tuplet È una struttura concreta che estende *TupletPrototype*, per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.

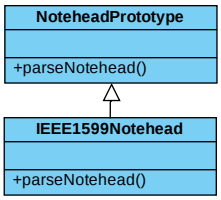


Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseTuplet* che li popola con i valori estratti dall'oggetto passato come parametro.

Listing 4.25: Definizione di IEEE1599Tuplet in JS

```
1  class IEEE1599Tuplet extends TupletPrototype {
2    constructor(tuplet_ratio) {
3      super();
4      this.parseTuplet(tuplet_ratio);
5    }
6    parseTuplet(tuplet_ratio) {
7      try {
8        this.enterNum = parseInt(tuplet_ratio.getAttribute("
             enter_num"));
9        this.enterDen = parseInt(tuplet_ratio.getAttribute("
             enter_den"));
10       if (tuplet_ratio.getAttribute("enter_dots") !== null) {
11         this.enterDots = parseInt(tuplet_ratio.getAttribute("
               enter_dots"));
12       }
13       this.inNum = parseInt(tuplet_ratio.getAttribute("in_num")
             );;
14       this.inDen = parseInt(tuplet_ratio.getAttribute("in_den")
             );
15       if (tuplet_ratio.getAttribute("in_dots") !== null) {
16         this.inDots = parseInt(tuplet_ratio.getAttribute("
               in_dots"));
17       }
18     }
19     catch (error) {
20       console.log(error);
21     }
22   }
23 }
```

IEEE1599Notehead È una struttura concreta che estende *NoteheadPrototype*, per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.



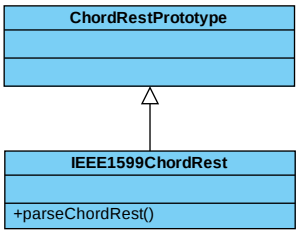
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseNotehead* che li popola con i valori estratti dall'oggetto passato come parametro.

Listing 4.26: Definizione di IEEE1599Notehead in JS

```
class IEEE1599Notehead extends NoteheadPrototype {
  constructor(notehead) {
    super();
    this.parseNotehead(notehead);
  }
  parseNotehead(notehead) {
    try {
      let pitch = notehead.getElementsByTagName("pitch")[0];
      this.step = pitch.getAttribute("step");
      this.octave = parseInt(pitch.getAttribute("octave"));
      this.printedAccidentals = [];
      let printedAccs = notehead.getElementsByTagName("
          printed_accidentals")[0];
      if (printedAccs !== undefined) {
        let accidentals = printedAccs.getElementsByTagName("*")
            ;
        for (let i = 0; i < accidentals.length; i++) {
          this.printedAccidentals.push(accidentals[i].nodeName)
              ;
        }
      }
      this.actualAccidental = notehead.getElementsByTagName("
          pitch")[0].getAttribute("actual_accidental");
      if (this.actualAccidental === null) {
        this.actualAccidental = "natural";
      }
      this.tie = (notehead.getElementsByTagName("tie")[0] ===
          undefined) ? false : true;
    }
    catch (error) {
      console.log(error);
    }
  }
}
```

IEEE1599ChordRest È una struttura astratta che estende *ChordRestPrototype*.

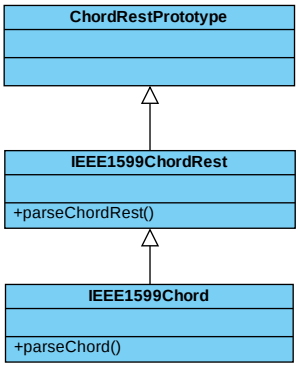
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseChordRest* che li popola estraendo le informazioni dall'oggetto passato come parametro, eventualmente istanziando oggetti per la rappresentazione di gruppi di note irregolari.



Listing 4.27: Definizione di IEEE1599ChordRest in JS

```
class IEEE1599ChordRest extends ChordRestPrototype {
  constructor(chord) {
    super();
    this.parseChordRest(chord);
  }
  parseChordRest(chord) {
    try {
      if (chord === undefined) {
        return null;
      }
      this.voice = chord.parentNode.getAttribute("
          voice_item_ref");
      this.measure = chord.parentNode.parentNode.getAttribute("
          number");
      this.durationNum = parseInt(chord.getElementsByTagName("
          duration")[0].getAttribute("num"));
      this.durationDen = parseInt(chord.getElementsByTagName("
          duration")[0].getAttribute("den"));
      let tuplet_ratio = chord.getElementsByTagName("
          tuplet_ratio")[0];
      if (tuplet_ratio !== undefined) {
        this.tuplet = new IEEE1599Tuplet(tuplet_ratio);
      }
      let augmentationDots = chord.getElementsByTagName("
          augmentation_dots")[0];
      if ((augmentationDots !== undefined) && (augmentationDots
           !== null)) {
        this.augmentationDots = parseInt(
            augmentationDots.getAttribute("number"));
      }
    }
    catch (error) {
      console.log(error);
    }
  }
}
```

IEEE1599Chord È una struttura concreta che estende *IEEE1599ChordRest*.

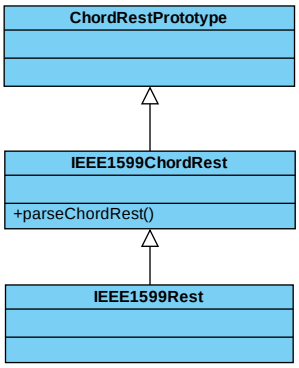


Il costruttore richiama quello del prototipo ed istanzia gli attributi ereditati, popolandoli con i valori estratti dall'oggetto passato come parametro.

Listing 4.28: Definizione di IEEE1599Chord in JS

```
1  class IEEE1599Chord extends IEEE1599ChordRest {
2    constructor(chord) {
3      super(chord);
4      this.parseChord(chord);
5    }
6    parseChord(chord) {
7      try {
8        this.noteheads = [];
9        var noteheads = chord.getElementsByTagName("notehead");
10       for (var i = 0; i < noteheads.length; i++) {
11         this.noteheads.push(new IEEE1599Notehead(noteheads[i]))
               ;
12       }
13     }
14     catch (error) {
15       console.log(error);
16     }
17   }
18 }
```

IEEE1599Rest È una struttura concreta che estende *IEEE1599ChordRest*.

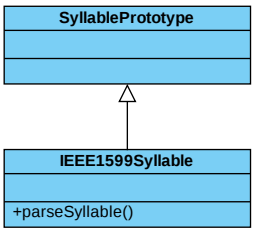


Il costruttore si limita a richiamare l'omologa funzione del prototipo passando i parametri per il popolamento degli attributi.

Listing 4.29: Definizione di IEEE1599Rest in JS

```
1 class IEEE1599Rest extends IEEE1599ChordRest {
2   constructor(rest) {
3     super(rest);
4     this.parseRest(rest);
5   }
6   parseRest(rest) { }
7 }
```

IEEE1599Syllable È una struttura concreta che estende il prototipo *SyllablePrototype*.



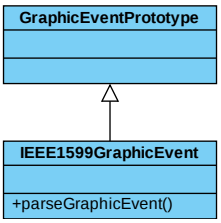
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseSyllable* che li popola con i valori ricavati dall'oggetto passato come parametro.

Listing 4.30: Definizione di IEEE1599Syllable in JS

```
1  class IEEE1599Syllable extends SyllablePrototype {
2    constructor(syllable) {
3      super();
4      this.parseSyllable(syllable);
5    }
6    parseSyllable(syllable) {
7      try {
8        this.startEventRef = syllable.getAttribute("
             start_event_ref");
9        if (syllable.getAttribute("end_event_ref") !== null) {
10         this.endEventRef = syllable.getAttribute("end_event_ref
               ");
11       }
12       if (syllable.getAttribute("hyphen") === "yes") {
13         this.hyphen = true;
14       }
15       else {
16         this.hyphen = false;
17       }
18       if (syllable.textContent !== undefined) {
19         this.text = syllable.textContent;
20       }
21       else if (syllable.text !== undefined) {
22         this.text = syllable.text;
23       }
24     }
25     catch (error) {
26       console.log(error);
27     }
28   }
29 }
```

IEEE1599GraphicEvent È una struttura concreta che estende il prototipo *GraphicEventPrototype*.

Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseGraphicEvent* che li popola con i valori ricavati dall'oggetto passato come parametro.



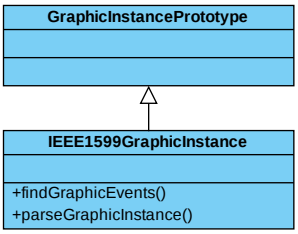
Listing 4.31: Definizione di IEEE1599GraphicEvent in JS

```
1  class IEEE1599GraphicEvent extends GraphicEventPrototype {
2    constructor(graphicEvent) {
3      super();
4      this.parseGraphicEvent(graphicEvent);
5    }
6    parseGraphicEvent(graphicEvent) {
7      try {
8        this.x0 = parseFloat(graphicEvent.getAttribute("
             upper_left_x"));
9        this.y0 = parseFloat(graphicEvent.getAttribute("
             upper_left_y"));
10       this.x1 = parseFloat(graphicEvent.getAttribute("
             lower_right_x"));
11       this.y1 = parseFloat(graphicEvent.getAttribute("
             lower_right_y"));
12       if (graphicEvent.hasAttribute("highlight_color")) {
13         this.highlightColor = graphicEvent.getAttribute("
               highlight_color");
14       }
15       if (graphicEvent.hasAttribute("description")) {
16         this.description = graphicEvent.getAttribute("
               description");
17       }
18     }
19     catch (error) {
20       console.log(error);
21     }
22   }
23 }
```

IEEE1599GraphicInstance È una struttura concreta che estende il prototipo *GraphicInstancePrototype*.

Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseGraphicInstance* che li popola con i valori ricavati dall'oggetto passato come

parametro. Viene implementato il metodo *findGraphicEvents*, per utilizzo da parte del Player Multimediale, che restituisce tutti gli eventi grafici per i quali le relative porzioni d'immagine includono un punto le cui coordinate sono definite mediante i parametri.

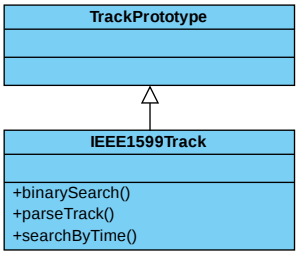


Listing 4.32: Definizione di IEEE1599GraphicInstance in JS

```
1  class IEEE1599GraphicInstance extends GraphicInstancePrototype
       {
2    constructor(page) {
3      super();
4      this.parseGraphicInstance(page);
5    }
6    findGraphicEvents(posX, posY) {
7      try {
8        let result = [];
9        let keys = Object.keys(this.graphicEvents);
10       for (let i in keys)
11         for (let j in this.graphicEvents[keys[i]]) {
12           let e = this.graphicEvents[keys[i]][j];
13           if (posX >= e["x0"] && posX <= e["x1"] && posY >= e["
                 y0"] && posY <= e["y1"]) {
14             result.push(keys[i]);
15           }
16         }
17       return result;
18     }
19     catch (error) {
20       console.log(error);
21     }
22   }
23   parseGraphicInstance(page) {
24     try {
25       if (page === undefined) {
26         return undefined;
27       }
```

```
28          this.positionInGroup = parseInt(page.getAttribute("
                position_in_group"));
29          this.fileName = makePath(page.getAttribute("file_name"));
30          if (page.hasAttribute("description")) {
31            this.description = page.getAttribute("description");
32          }
33          this.graphicEvents = {};
34          let graphicEvents = page.getElementsByTagName("
                graphic_event");
35          for (let i = 0; i < graphicEvents.length; i++) {
36            let key = graphicEvents[i].getAttribute("event_ref");
37            if (this.graphicEvents[key] === undefined) {
38              this.graphicEvents[key] = [];
39            }
40            this.graphicEvents[key].push(new IEEE1599GraphicEvent(
                  graphicEvents[i]));
41          }
42        }
43        catch (error) {
44          console.log(error);
45        }
46      }
47  }
```

IEEE1599Track È una struttura concreta che estende il prototipo *TrackPrototype*.



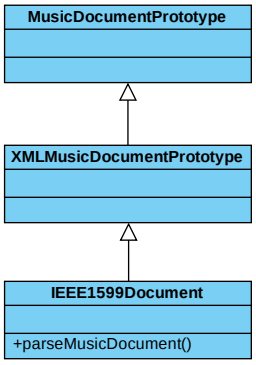
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseTrack* che li popola con i valori ricavati dall'oggetto passato come parametro. Vengono implementati i metodi *binarySearch* (ricerca binaria ricorsiva) e *searchByTime*, utilizzati dal Player Multimediale per la ricerca di eventi che occorrono in un determinato istante temporale.

Listing 4.33: Definizione di IEEE1599Track in JS

```
class IEEE1599Track extends TrackPrototype {
  constructor(tracks) {
    super(tracks);
    this.parseTrack(tracks);
  }
  binarySearch(list, value) {
    try {
      function _binarySearch(list, value, start, stop) {
        let index = Math.floor((start + stop) / 2);
        if (start >= stop) {
          if (index <= 0) {
            return list[0];
          }
          let diff = list[index] - value;
          if (diff > 0) {
            return list[index - 1];
          }
          return list[index];
        }
        if (value < list[index]) {
          return _binarySearch(list, value, start, index - 1);
        }
        if (value > list[index]) {
          return _binarySearch(list, value, index + 1, stop);
        }
        return list[index];
      }
      return _binarySearch(list, value, 0, list.length - 1);
    }
    catch (error) {
      console.log(error);
    }
  }
  parseTrack(tracks) {
    try {
      if (tracks === undefined) {
        return null;
      }
      this.notes = getChildTextContent(tracks, "notes") ?
          getChildTextContent(tracks, "notes") : undefined;
      this.performers = [];
      let performers = tracks.getElementsByTagName("performer")
          ;
      for (let i = 0; i < performers.length; i++) {
        let performer = performers[i];
        if (performer.getAttribute("name") !== null) {
          let name = performer.getAttribute("name");
          let type = performer.getAttribute("type");
```

```
47            if (type !== undefined) {
48              name += " (" + type + ")";
49            }
50            this.performers.push(name);
51          }
52        }
53        this.fileFormat = tracks.getAttribute("file_format");
54        this.trackEventsByRef = {};
55        this.trackEventsByTime = {};
56        let trackEvents = tracks.getElementsByTagName("
            track_event");
57        for (let i = 0; i < trackEvents.length; i++) {
58          let keyRef = trackEvents[i].getAttribute("event_ref");
59          let startTime = trackEvents[i].getAttribute("start_time
              ");
60          let keyTime = parseFloat(startTime);
61          if (keyTime % 1 === 0) {
62            keyTime = keyTime - 0.001;
63          }
64          this.trackEventsByRef[keyRef] = keyTime;
65          if (this.trackEventsByTime[keyTime] === undefined) {
66            this.trackEventsByTime[keyTime] = [];
67          }
68          this.trackEventsByTime[keyTime].push(trackEvents[i].
              getAttribute("event_ref"));
69        }
70      }
71      catch (error) {
72        console.log(error);
73      }
74    }
75    searchByTime(timeToFind) {
76      try {
77        let keysSorted = Object.keys(this.trackEventsByTime).sort
              ((a, b) => a - b);
78        let time = this.binarySearch(keysSorted, timeToFind);
79        return this.trackEventsByTime[time];
80      }
81      catch (error) {
82        console.log(error);
83      }
84    }
85  }
```

IEEE1599Document È una struttura di specializzazione che estende le funzionalità di *XMLDocumentPrototype*.



Il costruttore richiama l'omologa funzione di *XMLMusicDocumentPrototype*, con la sintassi *super*: vengono istanziati gli attributi, ereditati dalla catena di prototipi, i quali vengono popolati dall'esecuzione del metodo di parsing *parseMusicDocument* (descritto nella seguente Sezione 4.4.2).

Listing 4.34: Definizione di IEEE1599Document in JS

```
1  class IEEE1599Document extends XMLMusicDocumentPrototype {
2    constructor(doc, url) {
3      super(doc, url);
4    }
5    parseMusicDocument() {
6      try {
7        [...]
8      }
9      catch (error) {
10       console.log(error);
11     }
12   }
13 }
```

4.4.2 Algoritmo di parsing file

In questa sezione viene analizzata l'implementazione del metodo *parseMusicDocument* per la struttura dati *IEEE1599Document*: il codice sorgente è consultabile in Appendice A.2.

Le strutture dati proposte nel presente lavoro sono state ideate per essere compatibili con il Player Multimediale basato sullo standard IEEE 1599: di conseguenza il parsing di file in questo formato risulta essere molto lineare. Grazie al supporto per l'XML DOM, nativo in JavaScript, risulta semplice ricavare elementi ed attributi dalla rappresentazione del file originale contenuta in *xmlDoc*: nella maggior parte dei casi, dopo aver ricercato un particolare nodo per tag-name, le informazioni vengono estratte direttamente dal nodo oppure sfogliando l'alberatura dei nodi vicini spostandosi per un numero limitato di livelli (nodo padre/nodi figli).

I primi step del metodo *parseMusicDocument* vertono sull'estrazione dei metadati afferenti al layer General e sul popolamento dei relativi attributo dell'oggetto *IEEE1599Document*, come indicato nella Tabella 4:

Tabella 4: parseMusicDocument: estrazione informazioni da Layer General IEEE 1599

Attributo	**Origine informazione**
mainTitle	testo del tag *main_title*
number	testo del tag *number*
workTitle	testo del tag *work_title*
workNumber	testo del tag *work_number*
authors	ricerca di tutti i tag *author* da usare come parametri per la costruzione di oggetti *XMLAuthor*
relatedFiles	ricerca di tutti i tag *related_file* da usare come parametri per la costruzione di oggetti *IEEE1599RelatedFile*

In seguito si passa al layer Logic, con l'estrazione di tutti gli eventi memorizzati nei nodi figli del tag *spine*: l'*id* di questi elementi viene utilizzato per popolare gli attributi *spineIds*, *spineHash* e *spine*. Quest'ultimo attributo, in particolare, viene valorizzato con oggetti di tipo *XMLEvent* per i quali viene calcolato anche il progressivo temporale cumulato (basandosi sul valore di *timing*).

Terminata l'elaborazione dello spine, avviene l'estrazione delle informazioni dal LOS secondo le modalità indicate in Tabella 5.

Tabella 5: estrazione informazioni dal LOS IEEE 1599

Attributo	**Origine informazione**
staves	attributo *id* dei nodi con tag *staff*
parts	attributo *id* dei nodi con tag *part*

clefs	attributi *shape*, *staff_step* dei nodi con tag *clef* + attributo *id* del nodo *staff* padre usati per costruzione di oggetti *XMLClef*; attributo *event_ref* dei nodi con tag *clef* usato come chiave
keys	attributi *event_ref*, *sharp_num*, *flat_num* dei nodi con tag *key_signature* + attributo *id* del nodo *staff* padre usati per costruzione di oggetti *XMLKeySignature*. In caso di definizione di chiavi custom (tag *custom_key_signature*), gli attributi *step* dei nodi figli *key_accidental* vengono utilizzati per indicare le alterazioni per esteso. Attributo *event_ref* dei nodi con tag *key_signature* usato come chiave.
times	per ogni nodo con tag *time_signature*, attributi *num*, *den*, *vtu_amount* dei rispettivi nodi figli con tag *time_indication* + attributo *id* del nodo *staff* padre usati per costruzione di oggetti *XMLTimeSignature*. Attributo *event_ref* dei nodi con tag *time_signature* usato come chiave.
voices	*staff_ref* dei nodi con tag *voice_item* + attributo *id* del nodo *part* di appartenenza per costruzione di oggetti *XMLVoice*; attributo *id* dei nodi con tag *voice_item* usato come chiave
measures	attributo *number* dei nodi con tag *measure* usato come chiave, attributo *id* del nodo *part* di appartenenza come valore
chords	per ogni nodo con tag *chord* viene istanziato un oggetto *IEEE1599Chord*; attributo *event_ref* usato come chiave
rests	per ogni nodo con tag *rest* viene istanziato un oggetto *IEEE1599Rest*; attributo *event_ref* usato come chiave
lyrics	per ogni nodo con tag *lyrics*, attributo *voice_ref* usato come chiave e, come valori, oggetti *KernSyllable* istanziati a partire dai nodi figli con tag *syllable*

Successivamente si passa al parsing delle informazioni relative alle istanze grafiche del layer Notational: per ogni gruppo di istanze (nodi con tag *graphic_instance_group*), l'attributo *description* viene utilizzato come chiave. I valori vengono ricercati negli elementi figli con tag *graphic_instance*: ogni nodo viene utilizzato per la creazione di un oggetto *IEEE1599GraphicInstance*; i relativi attributi *description* vengono impiegati come chiavi. Ai singoli oggetti

IEEE1599GraphicInstance viene associata la funzione *firstEventInPage*, mutuata dall'*ieee1599-framework*, che restituisce il primo evento grafico presente nella rappresentazione.

Infine vengono estratte le informazioni relative alle tracce del layer Audio, ricercando tutti i nodi con tag *track* che vengono impiegati per istanziare oggetti di tipo *IEEE1599Track*: i relativi attributi *file_name* vengono utilizzati come chiavi.

Il metodo di parsing termina l'esecuzione con la costruzione delle statistiche *stats* (oggetto *XMLStatistics*).

4.5 File MusicXML

4.5.1 Strutture dati specialistiche

Questa sezione contiene la descrizione delle strutture di specializzazione utilizzate per rappresentare documenti musicali ottenuti a partire da file XML in formato IEEE 1599 (.musicxml).

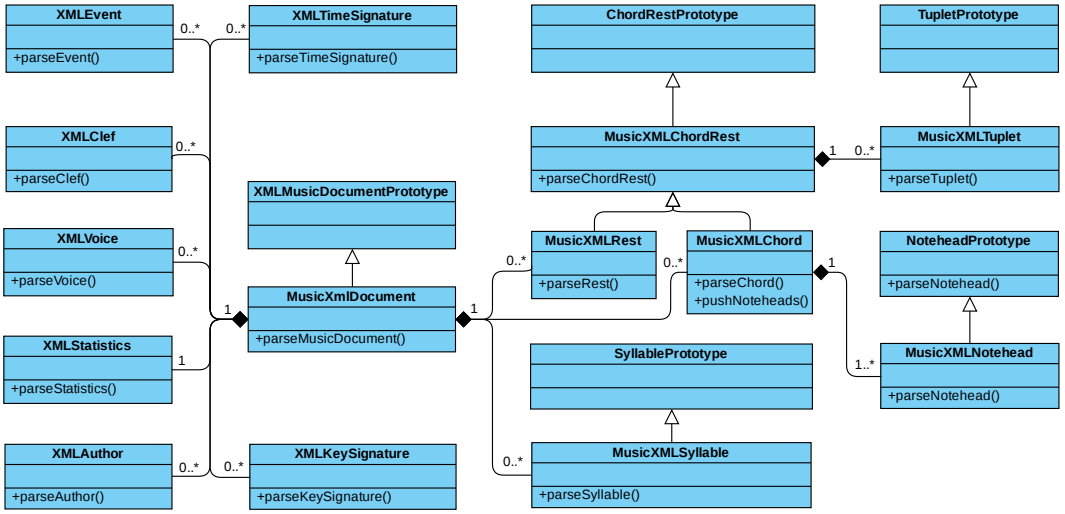
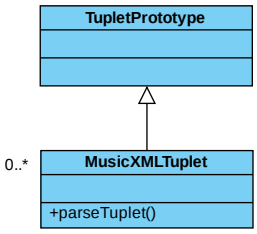


Figura 19: Specializzazione MusicXML - rappresentazione UML

In Figura 19 sono rappresentati gli elementi che costituiscono questa implementazione: per compattezza non sono stati elencati per esteso gli attributi e i metodi delle strutture astratte già descritte in precedenza. Nel prosieguo della sezione vengono illustrati in dettaglio gli elementi costitutivi.

MusicXMLTuplet È una struttura concreta che estende *TupletPrototype*, per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.



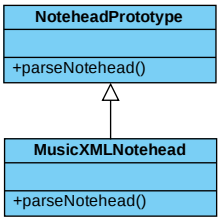
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseTuplet* che li popola con i valori estratti dagli oggetti passati come parametri e convertiti per essere compatibili con i tipi del modello (ad esempio l'indicazione del valore della nota del gruppo irregolare viene convertita nella rispettiva durata come frazione).

Listing 4.35: Definizione di MusicXMLTuplet in JS

```
1  class MusicXMLTuplet extends TupletPrototype {
2    constructor(note, den) {
3      super();
4      this.parseTuplet(note, den);
5    }
6    parseTuplet(note, den) {
7      try {
8        let timeModification = note.getElementsByTagName("
             time-modification")[0];
9        this.enterNum = getChildIntTextContent(timeModification,
             "actual-notes");
10       this.inNum = getChildIntTextContent(timeModification, "
             normal-notes");
11       let denType = getChildTextContent(timeModification, "
             normal-type");
12       if (denType !== "") {
13         switch (denType) {
14           case "1024th": den = 1024; break;
15           case "512th": den = 512; break;
16           case "256th": den = 256; break;
17           case "128th": den = 128; break;
18           case "64th": den = 64; break;
19           case "32nd": den = 32; break;
20           case "16th": den = 16; break;
21           case "eighth": den = 8; break;
22           case "quarter": den = 4; break;
23           case "half": den = 4; break;
24           case "whole": den = 4; break;
25           case "breve": den = 4; break;
```

```
26              case "long": den = 4; break;
27              case "maxima": den = 4; break;
28            }
29          }
30          this.enterDen = den;
31          this.inDen = den;
32          let dots = note.getElementsByTagName("dot").length;
33          if (dots > 0) {
34            this.enterDots = dots;
35          }
36          let normalDots = timeModification.getElementsByTagName("
              normal-dot").length;
37          if (normalDots > 0) {
38            this.inDots = normalDots;
39          }
40        }
41        catch (error) {
42          console.log(error);
43        }
44      }
45  }
```

MusicXMLNotehead È una struttura concreta che estende *NoteheadPrototype*, per l'utilizzo in strutture dati derivate da *ChordRestPrototype*.



Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseNotehead* che li popola con i valori estratti dall'oggetto passato come parametro, eseguendo le conversioni di rappresentazione del caso (ad esempio i valori di alterazione che vengono espressi per esteso).

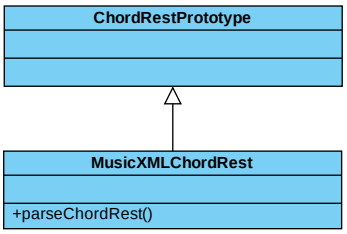
Listing 4.36: Definizione di MusicXMLNotehead in JS

```
1  class MusicXMLNotehead extends NoteheadPrototype {
2    constructor(note) {
3      super(note);
4      this.parseNotehead(note);
5    }
```

```
6     parseNotehead(note) {
7       try {
8         this.step;
9         this.octave;
10        this.printedAccidentals = [];
11        let acc = getChildTextContent(note, "accidental");
12        if (acc !== "") {
13          this.printedAccidentals.push(acc);
14        }
15        this.actualAccidental = "natural";
16        let pitches = note.getElementsByTagName("pitch");
17        let pitch;
18        if (pitches.length === 0) {
19          pitches = note.getElementsByTagName("unpitched");
20          if (pitches.length > 0) {
21            pitch = pitches[0];
22            this.step = getChildTextContent(pitch, "display-step"
                );
23            this.octave = getChildIntTextContent(pitch, "
                display-octave");
24          }
25        }
26        else {
27          pitch = pitches[0];
28          this.step = getChildTextContent(pitch, "step");
29          this.octave = getChildIntTextContent(pitch, "octave");
30          switch (getChildTextContent(pitch, "alter")) {
31            case "-2": this.actualAccidental = "double_flat";
                  break;
32            case "-1.5": this.actualAccidental = "flat_and_a_half
                  "; break;
33            case "-1": this.actualAccidental = "flat"; break;
34            case "-0.5": this.actualAccidental = "demiflat";
                  break;
35            case "0": this.actualAccidental = "natural"; break;
36            case "0.5": this.actualAccidental = "demisharp";
                  break;
37            case "1": this.actualAccidental = "sharp"; break;
38            case "1.5": this.actualAccidental = "sharp_and_a_half
                  "; break;
39            case "2": this.actualAccidental = "double_sharp";
                  break;
40          }
41        }
42        this.tie = false;
43        let ties = note.getElementsByTagName("tie");
44        for (let i = 0; i < ties.length; i++) {
45          if (ties[i].getAttribute("type") === "start") {
46            this.tie = true;
```

```
47          }
48        }
49      }
50      catch (error) {
51        console.log(error);
52      }
53    }
54  }
```

MusicXMLChordRest È una struttura astratta che estende *ChordRestPrototype*.



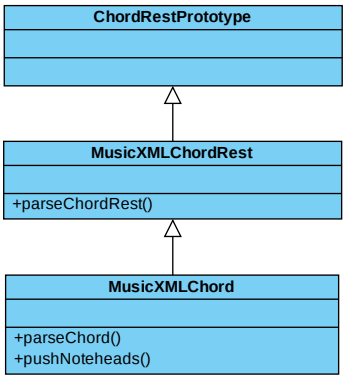
Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseChordRest* che li popola estraendo le informazioni dagli oggetti passati come parametri, eventualmente istanziando oggetti per rappresentare gruppi di note irregolari.

Listing 4.37: Definizione di MusicXMLChordRest in JS

```
1  class MusicXMLChordRest extends ChordRestPrototype {
2    constructor(note, voice, measure, num, den) {
3      super();
4      this.parseChordRest(note, voice, measure, num, den);
5    }
6    parseChordRest(note, voice, measure, num, den) {
7      try {
8        if (voice !== undefined) {
9          this.voice = voice;
10       }
11       this.measure = measure;
12       this.durationNum = parseInt(num);
13       this.durationDen = parseInt(den);
14       if (note.getElementsByTagName("time-modification").length
             > 0) {
15         this.tuplet = new MusicXMLTuplet(note, den);
16       }
```

```
17          let dots = note.getElementsByTagName("dot").length;
18          if (dots > 0) {
19            this.augmentationDots = dots;
20          }
21        }
22        catch (error) {
23          console.log(error);
24        }
25      }
26  }
```

MusicXMLChord È una struttura concreta che estende *MusicXMLChordRest*.



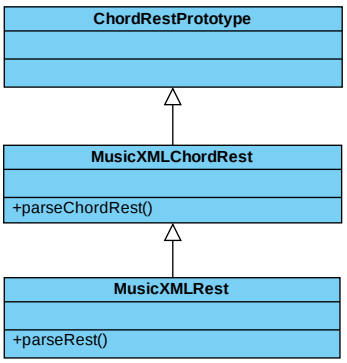
Il costruttore richiama quello del prototipo ed istanzia gli attributi ereditati, popolandoli con i valori passati come parametri, quindi invoca gli altri metodi definiti in questa struttura: *parseChord* (inizializzazione lista delle note) e *pushNoteheads* (aggiunta della nota passata come parametro alla lista). Il metodo *pushNoteheads* può essere richiamato, in istanti successivi alla costruzione dell'oggetto *MusicXMLchord*, per l'aggiunta di note all'accordo.

Listing 4.38: Definizione di MusicXMLChord in JS

```
1  class MusicXMLChord extends MusicXMLChordRest {
2    constructor(note, voice, measure, num, den) {
3      super(note, voice, measure, num, den);
4      this.parseChord();
5      this.pushNoteheads(note);
6    }
```

```
7     parseChord() {
8       try {
9         this.noteheads = [];
10      }
11      catch (error) {
12        console.log(error);
13      }
14    }
15    pushNoteheads(note) {
16      try {
17        this.noteheads.push(new MusicXMLNotehead(note));
18      }
19      catch (error) {
20        console.log(error);
21      }
22    }
23  }
```

MusicXMLRest È una struttura concreta che estende *MusicXMLChordRest*.



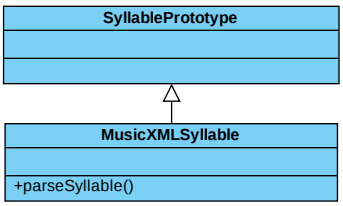
Il costruttore si limita a richiamare l'omologa funzione del prototipo passando i parametri per il popolamento degli attributi (in MusicXML le pause sono trattate come note degeneri).

Listing 4.39: Definizione di MusicXMLRest in JS

```
1  class MusicXMLRest extends MusicXMLChordRest {
2    constructor(note, voice, measure, num, den) {
3      super(note, voice, measure, num, den);
```

```
4     }
5 }
```

MusicXMLSyllable È una struttura concreta che estende il prototipo *SyllablePrototype*.



Il costruttore istanzia gli attributi, definiti nel prototipo, e richiama il metodo *parseSyllable* che li popola con i valori passati come parametri.

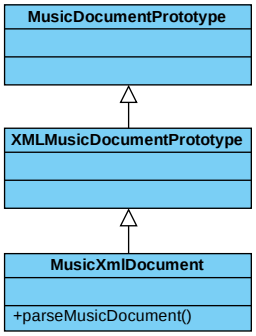
Listing 4.40: Definizione di MusicXMLSyllable in JS

```
1  class MusicXMLSyllable extends SyllablePrototype {
2      constructor(startEvent, endEvent, hyphen, text) {
3          super();
4          this.parseSyllable(startEvent, endEvent, hyphen, text);
5      }
6      parseSyllable(startEvent, endEvent, hyphen, text) {
7          try {
8              this.startEventRef = startEvent;
9              if (endEvent !== undefined) {
10                 this.endEventRef = endEvent;
11             }
12             this.hyphen = hyphen;
13             this.text = text;
14         }
15         catch (error) {
16             console.log(error);
17         }
18     }
19 }
```

MusicXMLDocument È una struttura di specializzazione che estende le funzionalità di *XMLDocumentPrototype*.

Il costruttore richiama l'omologa funzione di *XMLMusicDocumentPrototype*, con la sintassi *super*: vengono istanziati gli attributi, ereditati dalla catena di

prototipi, i quali vengono popolati dall'esecuzione del metodo di parsing *parseMusicDocument* descritto nella seguente Sezione 4.5.2.



Listing 4.41: Definizione di MusicXMLDocument in JS

```
1  class MusicXmlDocument extends XMLMusicDocumentPrototype {
2    constructor(doc, url) {
3      super(doc, url);
4    }
5    parseMusicDocument() {
6      try {
7        [...]
8      }
9      catch (error) {
10       console.log(error);
11     }
12   }
13 }
```

4.5.2 Algoritmo di parsing file

In questa sezione viene analizzata l'implementazione del metodo *parseMusicDocument* per la struttura dati *MusicXMLDocument*: il codice sorgente è consultabile in Appendice A.3.

Il parsing di file MusicXML in formato *score-partwise*, analogamente a quanto descritto nella Sezione 4.4.2 per i file IEEE 1599, è semplificato dall'utilizzo dell'XML DOM per sfogliare la rappresentazione del documento contenuta nell'attributo *xmlDoc* degli oggetti *MusicXMLDocument*.

Le prime istruzioni del metodo *parseMusicDocument* si occupano dell'estrazione dei metadati relativi al brano, secondo quanto riportato nella Tabella 6.

Tabella 6: estrazione metadati da documento MusicXML

Attributo	**Origine informazione**
mainTitle	testo del nodo con tag *movement-title*
number	testo del nodo con tag *movement-number*
workTitle	testo del nodo con tag *work_title*
workNumber	testo del nodo con tag *work_number*
authors	ricerca di tutti i tag *creator* da usare come parametri per la costruzione di oggetti *XMLAuthor*
relatedFiles	non presente in MusicXML (popolato con oggetto vuoto)

Come anticipato nella Sezione 2.4, che illustra le caratteristiche di base di un documento in questo formato, la durata delle note (e delle pause), viene descritta con l'elemento *duration* che è in rapporto con il valore associato alle divisioni della nota da $\frac{1}{4}$. Quest'ultimo valore viene descritto con il nodo *divisions* che compare tra gli attributi delle misure (solitamente nella prima misura di ogni parte). In un file MusicXML composto da più parti, può capitare di riscontrare misure con differenti valori di *divisions*: questo comporta che note con il medesimo valore nominale di durata possano essere espresse con differenti rapporti tra *duration* e *divisions*.

L'algoritmo di parsing si occupa quindi di determinare il minor rapporto $\frac{duration}{divisions}$ da utilizzare come moltiplicatore base per codificare la durata degli eventi in VTU: una volta individuato questo valore, lo si confronta con quelli espressi in Tabella 7 e si ricava il valore di duration immediatamente inferiore (per gestire valori frazionari intermedi risultanti dalla presenza di punti di valore).

Tabella 7: mapping durate per parsing file MusicXML

name	**duration**	**num**	**den**
maxima	8	32	4
long	4	16	4
breve	2	8	4
whole	1	4	4
half	0,5	2	4
quarter	0,25	1	4
eighth	0,125	1	8
16th	0,0625	1	16
32nd	0,03125	1	32
64th	0,015625	1	64
128th	0,0078125	1	128

256th	0,00390625	1	256
512th	0,001953125	1	512
1024th	0,0009765625	1	1024

Questo valore verrà ulteriormente rettificato a fronte della presenza di gruppi irregolari (nodi *"time-modification"*), scalandone l'inverso con il rapporto $\frac{normal\text{-}notes}{actual\text{-}notes}$ di entità massima.

A questo punto si procede con il parsing delle parti per estrarre le informazioni utili: per ogni nodo con tag *part* viene ricavato l'attributo *id*, per popolare l'elenco delle parti (*parts*) dell'oggetto *MusicXMLDocument*, e l'insieme dei nodi figli di tipo *measure*. Per ogni misura si estrae l'attributo *number* e lo si utilizza come chiave per popolare l'elenco delle misure *measure* del *MusicXMLDocument* con il part-id ricavato in precedenza. Si controlla se l'attributo divisions è stato modificato ed eventualmente si recepisce il valore aggiornato da utilizzare per il calcolo della durata delle note.

Si procede quindi a verificare la presenza di informazioni utili per popolare gli attributi *clefs*, *keys* e *times*, come indicato nella Tabella 8.

Tabella 8: origini dati per *clefs*, *keys*, *times* documento MusicXML

Attributo	**Origine informazione**
clefs	contenuto dei nodi *sign* e *line* figli di ogni elemento *clef* + part-id + measure-id utilizzati per costruzione di oggetti *XMLClef*; attributo *number* dei nodi con tag *clef* usato per composizione event-id (in combinazione con part-id e measure-id)
keys	contenuto dei nodi *fifths* figli degli elementi con tag *key* + part-id + measure-id utilizzati per costruzione di oggetti *XMLKeySignature*. Event-id generato come combinazione di part-id e measure-id
times	contenuto dei nodi *beats* e *beat-type* figli degli elementi con tag *time* + part-id + measure-id utilizzati per costruzione di oggetti *XMLTimeSignature*. Event-id generato come combinazione di part-id e measure-id

Ogni volta che viene riscontrata la presenza di uno di questi tre elementi, viene contestualmente aggiornata una struttura temporanea di supporto, denominata *spine*, con il relativo id evento e il valore assunto, in quell'istante, dal contatore globale che traccia la posizione dell'evento lungo l'asse temporale. Si rende necessario ricorrere a delle strutture di supporto, come quella appena descritta, a

causa della presenza dei tag *backup* e *forward* che alterano la sequenza temporale: come già affrontato nella Sezione 2.4, l'ordine cronologico degli eventi di un file MusicXML non segue necessariamente quello con il quale i relativi nodi compaiono nell'alberatura.

Terminata questa fase, si istanzia un contatore per gli eventi delle singole voci e si collezionano tutti i nodi figlio che vengono processati a seconda della tipologia:

- nodo *note*;
- nodo *backup*;
- nodo *forward*.

note A fronte di nodename coincidente con il tag *note*, si procede all'estrazione delle informazioni afferenti alla voce (nodo *voice*, valore di default v1) e al pentagramma di appartenenza (nodo *staff*, valore di default 1): queste vengono impiegate per generare gli id evento utilizzati per aggiornare, a fronte di necessità, gli attributi *staves* e *voices* dell'oggetto *MusicXMLDocument*. Nel caso in cui si riscontri una nuova voce, viene creato un oggetto *XMLVoice* e il valore del relativo contatore eventi viene impostato a 0. Se la nota contiene un nodo con tag *chord*, il contatore eventi viene incrementato di una unità. L'id evento viene costruito concatenando voice-id, measure-id e il valore del contatore.

Viene quindi estratta l'informazione di durata codificata nel nodo *duration*: ai fini di ricavarne il corrispondente in VTU, questa viene rapportata al valore delle divisioni e moltiplicata per il valore base individuato durante i primi step di esecuzione del metodo; il risultato viene salvato in una variabile locale denominata *durationWeighted*.

Il valore di *duration*, estratto dal relativo nodo, comprende anche le alterazioni dovute all'appartenenza a gruppi irregolari e/o alla presenza di punti di valore: al fine di determinare l'originale valore simbolico di durata dell'evento, si rende necessario individuare la presenza di questi elementi. Si ricercano quindi, tra i nodi figli di *note*, quelli con tag *time-modification* e si estraggono i valori degli elementi *actual-notes* e *normal-notes* con i quali viene rideterminato il valore di duration ($duration = duration \cdot \frac{actual\text{-}notes}{normal\text{-}notes}$). A questa operazione segue la ricerca dei nodi con tag *dot*, la cui eventuale presenza viene utilizzata per diminuire ulteriormente il valore di durata della nota (congruentemente alla quantità di elementi presenti). Il risultato ottenuto viene utilizzato per estrarre, da una lookup-table modellata sul contenuto della Tabella 7, i valori frazionari (*num* e *den*) che rappresentano la durata simbolica dell'evento.

In assenza di un nodo *chord*, figlio di *note*, si aggiunge un nuovo evento allo *spine* temporaneo e si aggiorna il contatore globale sommando il valore di durata in VTU salvato nella variabile *durationWeighted* (il valore del contatore globale,

precedente ad ogni operazione di modifica, viene salvato in una variabile temporanea denominata *prevCounter*). Se il nodo *note* contiene un figlio con tag rest, si istanzia una nuova pausa (*MusicXMLRest*), specificando la durata con i valori di *num* e *den*, e la si aggiunge all'attributo *rest* dell'oggetto *MusicXMLDocument*; in caso contrario si istanzia un nuovo accordo (*MusicXMLChord*) da aggiungere a *chords*.

Se invece è presente un nodo *chord*, l'elemento descritto da *note* è una nota che appartiene ad un accordo precedentemente istanziato: viene quindi invocato il metodo *pushNoteheads* dell'oggetto *MusicXMLChord* afferente all'id-evento corrente (popolato con il valore del contatore globale precedente l'ultimo aggiornamento).

Infine vengono ricercati i nodi relativi al testo cantato identificati con il tag *lyric*: l'id-evento viene composto leggendo l'attributo *number*, il testo viene letto dal nodo figlio *text* e la presenza del trattino separatore viene dedotta dal contenuto del nodo figlio *syllabic*. Le informazioni eventualmente ottenute vengono utilizzate per creare un nuovo oggetto *MusicXMLSyllable* da aggiungere alla rispettiva rappresentazione del testo nell'attributo *lyrics* di *MusicXMLDocument*.

backup Se la misura contiene un nodo con il tag *backup*, viene letto il contenuto dell'elemento figlio *duration*: il valore ottenuto viene rapportato al valore di *division* e moltiplicato per il valore di VTU di base. Il risultato viene utilizzato per decrementare il contatore globale.

forward In maniera analoga al caso precedente, se la misura contiene un nodo con il tag *forward*, viene il letto contenuto dell'elemento figlio *duration*: il valore ottenuto viene rapportato al valore di *division* e moltiplicato per il valore di VTU di base. Il risultato viene utilizzato per incrementare il contatore globale.

Terminato il parsing dei contenuti, viene avviato il consolidamento degli eventi: gli elementi dello *spine* temporaneo vengono riordinati ed utilizzati per creare nuovi *XMLEvent* da memorizzare nell'attributo *spine* del *MusicXMLDocument*; gli attributi *spineIds* e *spineHash* vengono popolati con i relativi event-id.

Lo step finale preve l'invocazione del metodo *parseAllHash*, per popolare l'attributo *allHash* unendo i contenuti di *rest* e *chords*, e la creazione di una nuova istanza di *XMLStatistics* per valorizzare le statistiche del brano (attributo *stats*).

Capitolo 5

Risultati

Questo capitolo illustra i risultati ottenuti processando alcuni file musicali, codificati nei formati di partenza descritti nel Capitolo 2, con le procedure descritte nel Capitoli 3 e 4.

5.1 Verifiche preliminari

Ai fini dei test funzionali, è stata predisposta una libreria di documenti musicali così composta:

- 20 file musicali in formato Humdrum **kern, scaricati dall'archivio online *KernScores* (`http://kern.humdrum.org/`);
- 30 file musicali in formato IEEE 1599, scaricati dalla collezione online del LIM (`https://ieee1599.lim.di.unimi.it/music_collection.php`);
- 18 file musicali in formato MusicXML score-partwise, costituenti il set di esempio scaricabile dal portale MusicXML (`https://www.musicxml.com/music-in-musicxml/example-set/`)
  Da questi files sono state ricavate, mediante trasformazione XSLT, le relative rappresentazione score-timewise (18 file aggiuntivi).

La suite di test, descritta nella Sezione 3.5, è stata applicata ai singoli file per verificare che gli algoritmi di parsing producessero gli esiti desiderati: gli oggetti ottenuti non hanno presentato anomalie. È stata implementata una pagina HTML, il cui codice sorgente è consultabile in Appendice D, che istanzia gli oggetti per l'intera libreria di file e stampa i risultati a console, come mostrato in Figura 20.

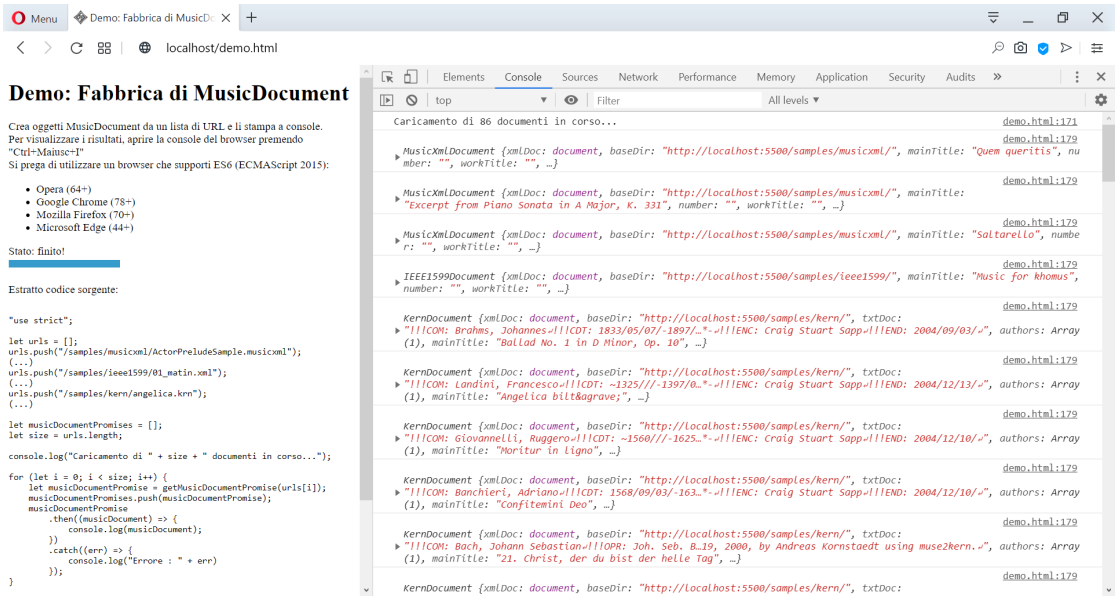


Figura 20: Pagina demo.html

5.2 Humdrum **kern

5.2.1 Xiao baicai (han0436.krn)

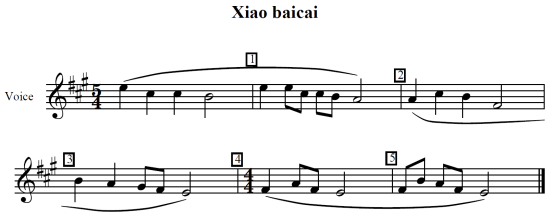


Figura 21: Xiao baicai - partitura

Questo brano presenta un tema molto semplice che dura poche battute: è ideale per verificare il funzionamento di base dei modelli implementati per i documenti Humdrum **kern.

Ad esempio, confrontando le informazioni statistiche mostrate in Figura 22 con quanto presente nella partitura in Figura 21, è semplice verificare che:

- *numParts* e *numMeasures* - è presente una parte con 5 misure (il primo gruppetto di note non afferisce ad una misura specifica);
- *numChords* e *numNotes* - sono presenti 28 accordi e 28 note;

```
▾ stats: TXTStatistics
    numParts: 1
    numEvents: 31
    numChords: 28
    numRests: 0
    numMeasures: 5
    numNotes: 28
  ▸ pitchClass: (12) [0, 5, 0, 0, 6, 0, 6, 0, 1, 5, 0, 5]
  ▸ pitchClassWeighted: (12) [0, 4, 0, 0, 8.5, 0, 5, 0, 0.5, 5, 0, 5]
  ▸ midiPitch: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 5, 0, 0, 3, 0, 0, 0, 0, 0,…
  ▸ midiPitchWeighted: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 0, 0, 2.5, 0, 0…
  ▸ chordLength: {1: 10, 2: 6, 0.5: 12}
  ▸ noteLength: {1: 10, 2: 6, 0.5: 12}
  ▸ restLength: {}
  ▾ noteByLengthPC:
    ▸ 1: {0: 0, 1: 3, 2: 0, 3: 0, 4: 2, 5: 0, 6: 1, 7: 0, 8: 0, 9: 2, 10: 0, 11: 2}
    ▸ 2: {0: 0, 1: 0, 2: 0, 3: 0, 4: 3, 5: 0, 6: 1, 7: 0, 8: 0, 9: 1, 10: 0, 11: 1}
    ▸ 0.5: {0: 0, 1: 2, 2: 0, 3: 0, 4: 1, 5: 0, 6: 4, 7: 0, 8: 1, 9: 2, 10: 0, 11: 2}
    ▸ __proto__: Object
  ▸ __proto__: StatisticsPrototype
```


Figura 22: Xiao baicai - Statistics

- *numEvents* - gli eventi totali dello spine sono 31 (alterazioni in chiave + 2 indicazioni di tempo + 28 accordi);
- *pitchClass* - ci sono 5 note C# (pitch 1), 6 note E (pitch 4), 6 note F# (pitch 6), 1 nota G# (pitch 8), 5 note A (pitch 9) e 5 note B (pitch 11);
- *noteLength* e *chordLength* - ci sono 10 accordi/note da $\frac{1}{4}$ (length 1), 6 da $\frac{2}{4}$ (length 2) e 12 da $\frac{1}{8}$ (length 0.5);
- *noteByLengthPC* - ci sono 3 note C# da $\frac{1}{4}$, 3 note E da $\frac{1}{4}$, 1 nota E da $\frac{1}{8}$, e così via.
- etc.

Come mostrato in Figura 23, gli eventi codificati nello spine risultano ordinati in maniera corretta: l'algoritmo ha associato l'unità di VTU alla durata della nota da $\frac{1}{4}$.


```
▼ spine:
  ▶ TimeSignature_P1_0_0: KernEvent {absoluteTime: 0, relativeTime: 0}
  ▶ KeySignature_P1_0_0: KernEvent {absoluteTime: 0, relativeTime: 0}
  ▶ P1_1_0_1: KernEvent {absoluteTime: 0, relativeTime: 0}
  ▶ P1_1_0_2: KernEvent {absoluteTime: 2, relativeTime: 2}
  ▶ P1_1_0_3: KernEvent {absoluteTime: 4, relativeTime: 2}
  ▶ P1_1_0_4: KernEvent {absoluteTime: 6, relativeTime: 2}
  ▶ P1_1_1_1: KernEvent {absoluteTime: 10, relativeTime: 4}
  ▶ P1_1_1_2: KernEvent {absoluteTime: 12, relativeTime: 2}
  ▶ P1_1_1_3: KernEvent {absoluteTime: 13, relativeTime: 1}
  ▶ P1_1_1_4: KernEvent {absoluteTime: 14, relativeTime: 1}
  ▶ P1_1_1_5: KernEvent {absoluteTime: 15, relativeTime: 1}
  ▶ P1_1_1_6: KernEvent {absoluteTime: 16, relativeTime: 1}
  ▶ P1_1_2_1: KernEvent {absoluteTime: 20, relativeTime: 4}
  ▶ P1_1_2_2: KernEvent {absoluteTime: 22, relativeTime: 2}
  ▶ P1_1_2_3: KernEvent {absoluteTime: 24, relativeTime: 2}
  ▶ P1_1_2_4: KernEvent {absoluteTime: 26, relativeTime: 2}
  ▶ P1_1_3_1: KernEvent {absoluteTime: 30, relativeTime: 4}
  ▶ P1_1_3_2: KernEvent {absoluteTime: 32, relativeTime: 2}
  ▶ P1_1_3_3: KernEvent {absoluteTime: 34, relativeTime: 2}
  ▶ P1_1_3_4: KernEvent {absoluteTime: 35, relativeTime: 1}
  ▶ P1_1_3_5: KernEvent {absoluteTime: 36, relativeTime: 1}
  ▶ TimeSignature_P1_4_0: KernEvent {absoluteTime: 40, relativeTime: 4}
  ▶ P1_1_4_1: KernEvent {absoluteTime: 40, relativeTime: 0}
  ▶ P1_1_4_2: KernEvent {absoluteTime: 42, relativeTime: 2}
  ▶ P1_1_4_3: KernEvent {absoluteTime: 43, relativeTime: 1}
  ▶ P1_1_4_4: KernEvent {absoluteTime: 44, relativeTime: 1}
  ▶ P1_1_5_1: KernEvent {absoluteTime: 48, relativeTime: 4}
  ▶ P1_1_5_2: KernEvent {absoluteTime: 49, relativeTime: 1}
  ▶ P1_1_5_3: KernEvent {absoluteTime: 50, relativeTime: 1}
  ▶ P1_1_5_4: KernEvent {absoluteTime: 51, relativeTime: 1}
  ▶ P1_1_5_5: KernEvent {absoluteTime: 52, relativeTime: 1}
  ▶ __proto__: Object
```


Figura 23: Xiao baicai - spine

Anche le note sono rappresentate correttamente: a titolo d'esempio viene riportata, in Figura 24, la codifica della seconda nota del gruppetto di apertura del brano.

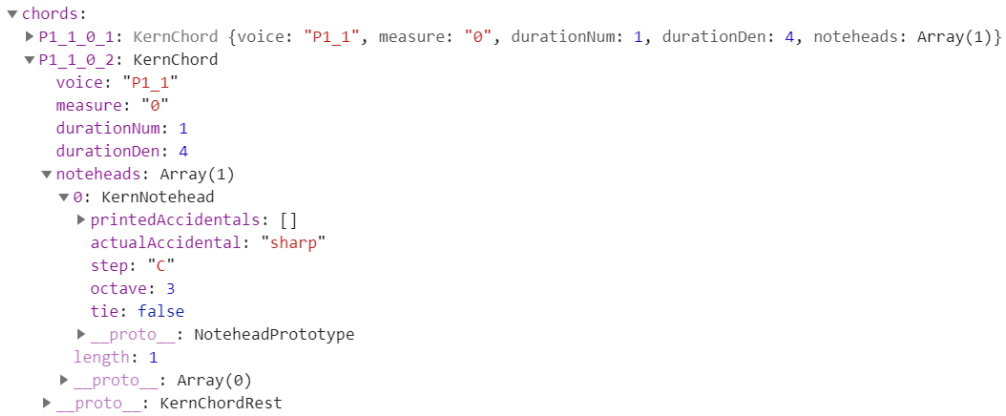


Figura 24: Xiao baicai - chords

La Figura 25 mostra l'elenco completo degli attributi dell'oggetto KernDocument ottenuto dal parsing del file originale: verificando i singoli elementi si giunge agli esiti descritti in Tabella 9.

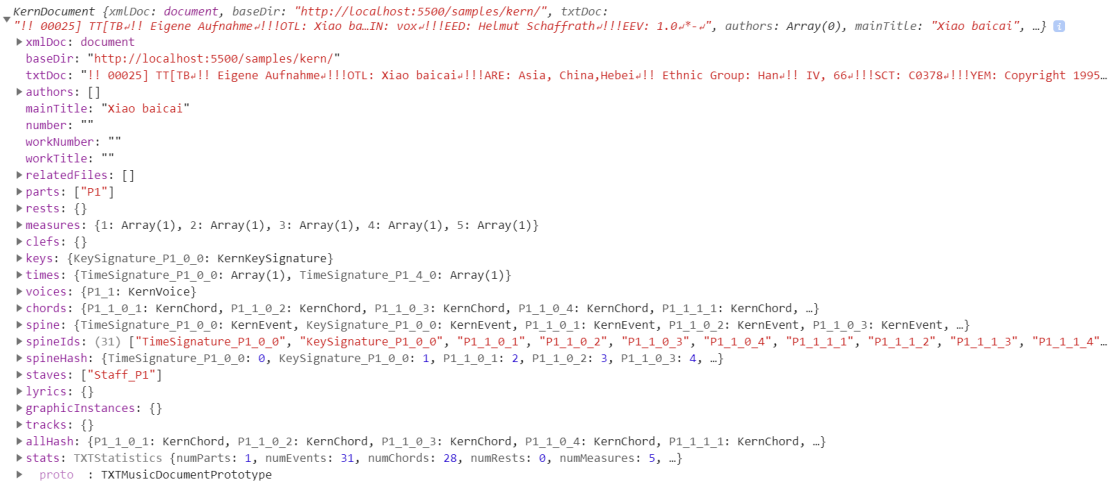


Figura 25: Xiao baicai - KernDocument

Tabella 9: Analisi attributi KernDocument Xiao baicai

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document vuoto
txtDoc	ok - popolato con il contenuto testuale del file
baseDir	ok - popolato con la radice di caricamento del file
mainTitle	ok - popolato correttamente

number	ok - vuoto (non presente nel file originale)
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - vuoto (non presente nel file originale)
relatedFiles	ok - vuoto (non trattato per i documenti Kern)
spine	ok - popolato correttamente con gli eventi e il timing
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - individuato singolo pentagramma *Staff_P1*
parts	ok - individuata singola parte *P1*
measures	ok - individuate cinque misure afferenti alla parte P1
clefs	ok - vuoto (nessuna chiave definita nel file originale)
keys	ok - individuate alterazioni in chiave ad inizio brano afferenti alla parte *P1* nel pentagramma *Staff_P1*
times	ok - individuate indicazioni di tempo ad inizio brano e nella misura 4, entrambe afferenti alla parte *P1* nel pentagramma *Staff_P1*
voices	ok - individuata singola voce *P1_1* afferente alla parte *P1* nel pentagramma *Staff_P1*
chords	ok - individuati 28 accordi composti da singole note, correttamente suddivisi per parte, misura e voce. I valori di altezza e durata delle singole note sono corretti, non sono presenti legature o punti di valore.
rests	ok - non sono presenti pause
allHash	ok - gli accordi sono riportati correttamente
stats	ok - i conteggi risultano corretti
lyrics	ok - non è presente testo cantato
graphicInstances	ok - vuoto (non trattato per i documenti Kern)
tracks	ok - vuoto (non trattato per i documenti Kern)

Di seguito viene riportato il contenuto completo del file orginale.

Listing 5.1: han0436.krn

```
!! 00025] TT[TB
!! Eigene Aufnahme
!!!OTL: Xiao baicai
!!!ARE: Asia, China,Hebei
!! Ethnic Group: Han
!! IV, 66
!!!SCT: C0378
```

```
!!!YEM: Copyright 1995, estate of Helmut Schaffrath.
**kern
*ICvox
*Ivox
!! 3+2 metric grouping
!! Mixed meters:  5/4; 4/4
*M5/4
*k[f#c#g#]
*A:
{4ee
4cc#
4cc#
2b
=1
4ee
8ee
8cc#
8cc#
8b
2a}
=2
{4a
4cc#
4b
2f#
=3
4b
4a
8g#
8f#
2e}
=4
*M4/4
{4f#
8a
8f#
2e
=5
8f#
8b
```

```
8a
8f#
2e}
==
!!!AGN: Xiaodiao; eigentlich war es eine Ballade; Kinderlied
!!      Lebenslauf, Klage
!! Als Tonbeispiel vorhanden; Signatur: 00151, Cut Nr.: 24
!!      sehr bekannt. Hauptsstueck der Oper: Das weisshaarige Maedchen.
!!      Inhalt: Ein Kind beklagt sich, dass der Vater nach dem Tod der
!!      Mutter wieder heiratet.
!!      Saengerin Huang Bai am 28.4.87 (wieder erkaeltet)
!!      WEITERE VERSIONEN: C1211, C0755, C0365
!!      Pinyin-Text der ersten Strophe vorhanden!
!!!ONB: ESAC (Essen Associative Code) Database: CHINA_032
!!!AMT: irregular
!!!AIN: vox
!!!EED: Helmut Schaffrath
!!!EEV: 1.0
*-
```

5.2.2 Angelica biltà (angelica.krn)

Questo brano è più complesso del precedente: contiene due parti musicali di pianoforte, per ogni parte sono indicate due linee di testo cantato. La Figura 26 illustra le misure 31-37 della partitura, dove si può notare la presenza di un gruppo irregolare di note (terzina).

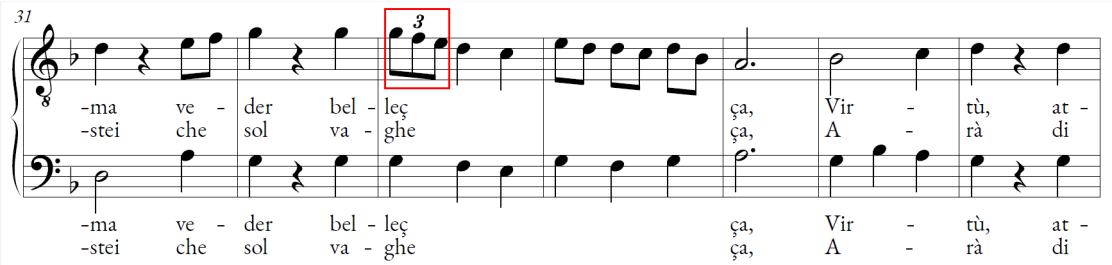


Figura 26: Angelica biltà - partitura

La Figura 27 mostra la rappresentazione KernChord delle note appartenenti alla terzina.

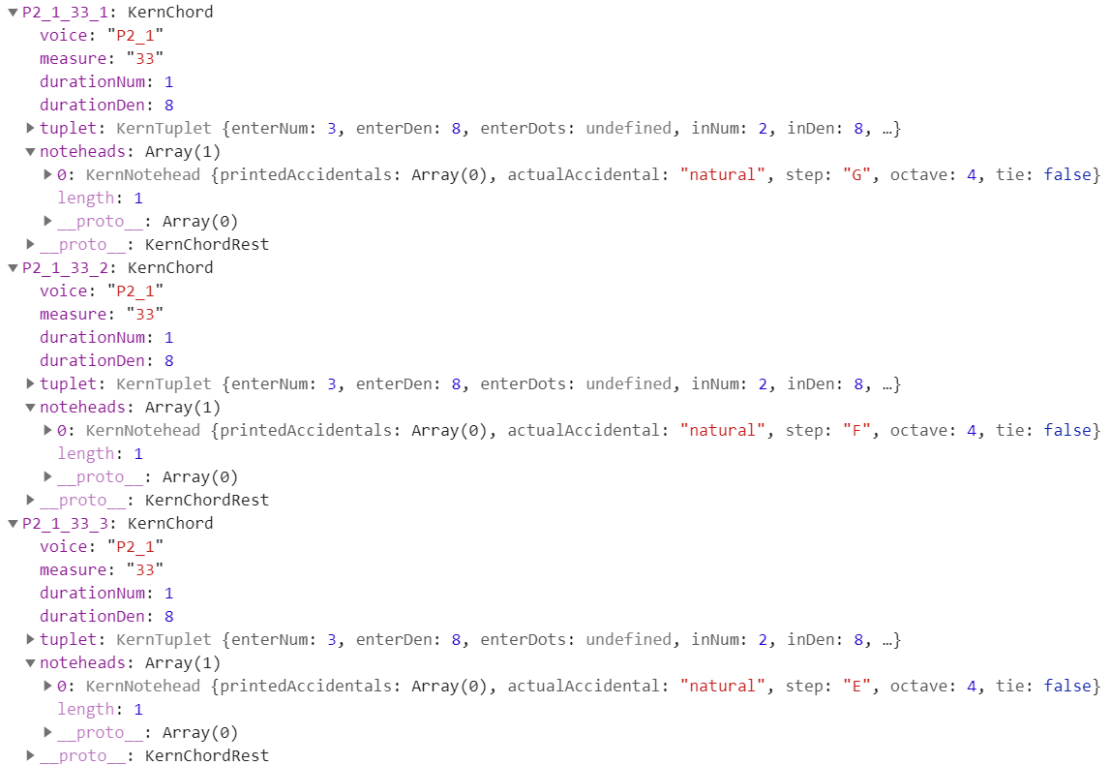


Figura 27: Angelica biltà - Terzina in misura 33

Risultano codificate correttamente anche le informazioni relative alle chiavi dei pentagrammi, come risulta dalla Figura 28 che mostra anche le relative alterazioni / indicazioni di tempo.

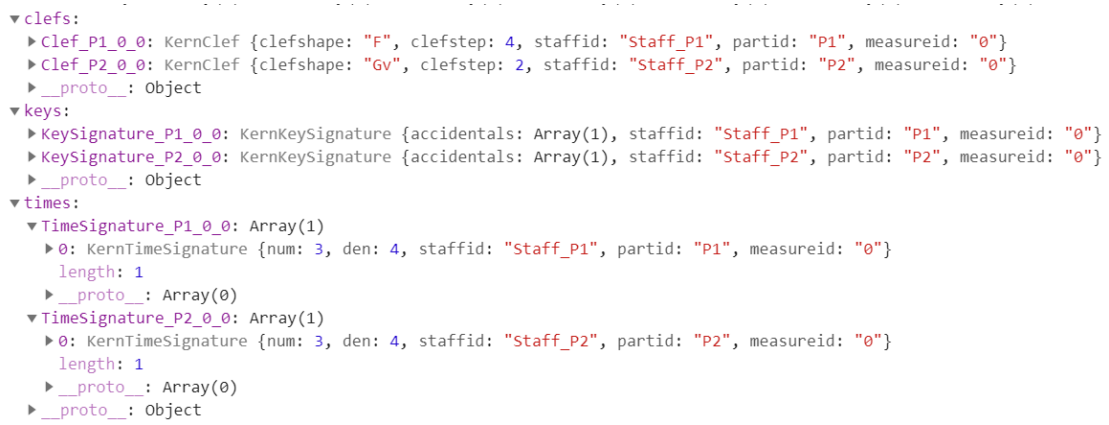


Figura 28: Angelica biltà - Chiavi, armature in chiave, tempo

In Figura 29 sono visibili gli oggetti che rappresentano le 4 linee di testo cantato, con il dettaglio delle KernSyllable afferenti alla prima linea della parte di basso.

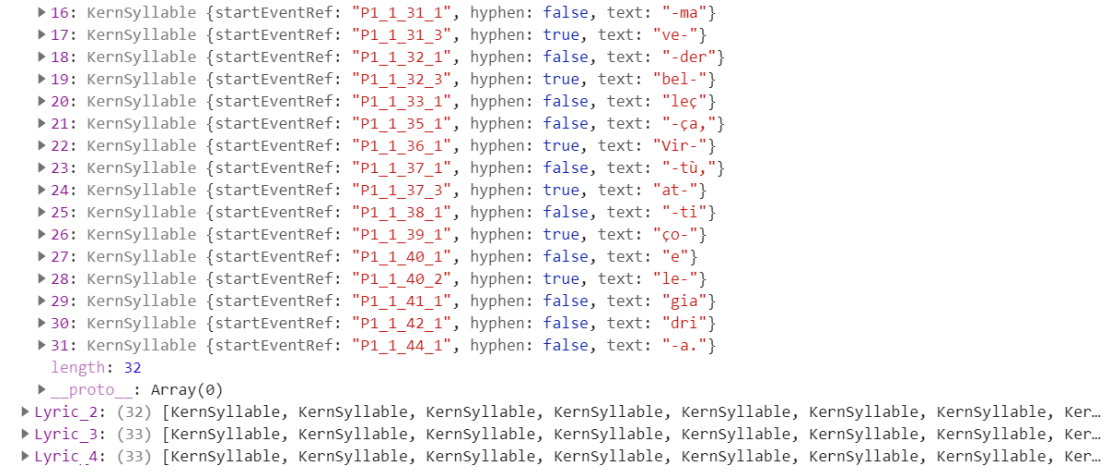


Figura 29: Angelica biltà - Sillabe misure 31-37 linea 1 parte di basso

Gli attributi completi dell'oggetto KernDocument generato dal file sono rappresentati in Figura 30: anche in questo caso risultano popolati in maniera coerente, come descritto in Tabella 10.

Figura 30: Angelica biltà - KernDocument

Tabella 10: Analisi attributi KernDocument Angelica Biltà

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document vuoto
txtDoc	ok - popolato con il contenuto testuale del file
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente
number	ok - vuoto (non presente nel file originale)
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - popolato con informazioni del compositore
relatedFiles	ok - vuoto (non trattato per i documenti Kern)
spine	ok - popolato correttamente con gli eventi e il timing
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - popolato con i due pentagrammi *Staff_P1* e *Staff_P2*
parts	ok - popolato con le due parti *P1* e *P2*

measures	ok - popolate misure dalla numero 19 alla numero 44. Le note della misura 18 sono presenti, ma nel file originale la misura non è contrassegnata da id esplicito; le note sono state assegnate alla misura 0 / gruppetto iniziale (come previsto dall'algoritmo di parsing)
clefs	ok - popolato con le chiavi presenti ad inizio brano per entrambe le parti
keys	ok - popolato con alterazioni in chiave ad inizio brano per entrambe le parti
times	ok - popolato con le indicazioni di tempo ad inizio brano per entrambe le parti
voices	ok - popolato con le informazioni delle singole voci presenti nelle due parti
chords	ok - gli accordi presenti sono ben formati, sono presenti alcuni punti di valore.
rests	ok - le pause presenti sono state trattate correttamente
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato, possono essere introdotti dei correttivi per i gruppi irregolari
lyrics	ok - popolato con 4 liste che codificano le sillabe delle linee cantate
graphicInstances	ok - vuoto (non trattato per i documenti Kern)
tracks	ok - vuoto (non trattato per i documenti Kern)

Segue l'estratto del file originale che descrive il contenuto musicale delle misure presenti in Figura 27.

Listing 5.2: angelica.krn

```
[...]
**kern  **text  **text  **kern  **text  **text
[...]
=31     =31     =31     =31     =31     =31
2D\     -ma     -stei   4d\     -ma     -stei
.       .       .       4r      .       .
4A\     ve-     che     8e\L    ve-     che
.       .       .       8f\J    .       .
=32     =32     =32     =32     =32     =32
4G\     -der    sol     4g\     -der    sol
4r      .       .       4r      .       .
4G\     bel-    va-     4g\     bel-    va-
```

=33	=33	=33	=33	=33	=33
4G\	l e c	g h e	12g\L	l e c	g h e
.	.	.	12f\	.	.
.	.	.	12e\J	.	.
4F\	.	.	4d\	.	.
4E\	.	.	4c\	.	.
=34	=34	=34	=34	=34	=34
4G\	.	.	8e\L	.	.
.	.	.	8d\J	.	.
4F\	.	.	8d\L	.	.
.	.	.	8c\J	.	.
4G\	.	.	8d\L	.	.
.	.	.	8B–\J	.	.
=35	=35	=35	=35	=35	=35
2.A\	−ca ,	−ca ,	2.A/	−ca ,	−ca ,
=36	=36	=36	=36	=36	=36
4G\	Vir−	A–	2B–\	Vir−	A–
4B–\	.	.	.	.	.
4A\	.	.	4c\	.	.
=37	=37	=37	=37	=37	=37
4G\	−tu ,	−r a	4d\	−tu ,	−r a
4r	.	.	4r	.	.
4G\	at−	d i	4d\	at−	d i

5.2.3 Piano Sonata no. 2 in A major (sonata02-3.krn)

Il file di testo che descrive questo brano contiene informazioni aggiuntive, codificate in uno spine ad hoc, che non sono di interesse ai fini del processo di parsing (indicazioni di dinamica): l'analisi dell'oggetto KernDocument, risultante in output, permette di verificare che i dati non rilevanti vengano correttamente scartati.

In questo brano, inoltre, viene fatto uso degli spine path indicators per rappresentare note eseguite da voci aggiuntive che si presentano, sporadicamente, in alcune misure: la Figura 31 rappresenta questo tipo di casistica (note evidenziate in rosso).

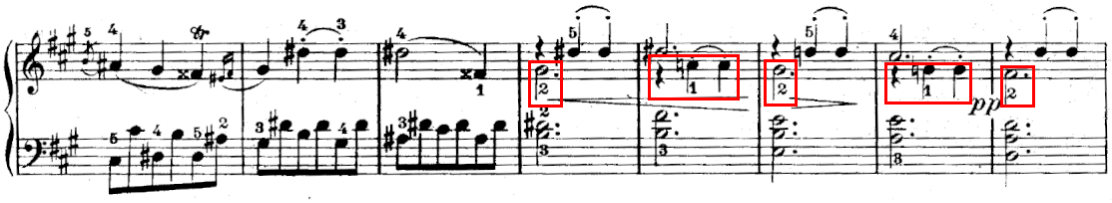


Figura 31: Sonata - partitura misure 22-29

La Figura 32 rappresenta gli oggetti KornChord e KornRest afferenti alla misura 25: si può notare come siano state correttamente create 2 voci (*P2_1* e *P2_2*) per la parte della mano destra (*P2*) e una voce per la parte della mano sinistra (*P1_1*).


```
▼P1_1_25_1: KernChord
    voice: "P1_1"
    measure: "25"
    durationNum: 2
    durationDen: 4
    augmentationDots: 1
  ▼noteheads: Array(2)
    ▶0: KernNotehead {printedAccidentals: Array(0), actualAccidental: "natural", step: "B", octave: 5, tie: false}
    ▶1: KernNotehead {printedAccidentals: Array(0), actualAccidental: "sharp", step: "D", octave: 4, tie: false}
      length: 2
    ▶__proto__: Array(0)
  ▶__proto__: KernChordRest
▶P2_1_25_1: KernRest {voice: "P2_1", measure: "25", durationNum: 1, durationDen: 4}
▶P2_1_25_2: KernChord {voice: "P2_1", measure: "25", durationNum: 1, durationDen: 4, noteheads: Array(1)}
▶P2_1_25_3: KernChord {voice: "P2_1", measure: "25", durationNum: 1, durationDen: 4, noteheads: Array(1)}
▼P2_2_25_1: KernChord
    voice: "P2_2"
    measure: "25"
    durationNum: 2
    durationDen: 4
    augmentationDots: 1
  ▼noteheads: Array(1)
    ▶0: KernNotehead {printedAccidentals: Array(0), actualAccidental: "sharp", step: "G", octave: 4, tie: false}
      length: 1
    ▶__proto__: Array(0)
  ▶__proto__: KernChordRest
```


Figura 32: KernDocument - eventi misura 25

L'immagine mostra anche il dettaglio degli eventi che avvengono contemporaneamente all'inizio della misura:

- *P1_1_25_1* della voce *P1_1* - descrive un accordo composto da due note (B e D#) da $\frac{2}{4}$ la cui durata è aumentata dalla presenza del punto di valore;
- *P2_1_25_1* della voce *P2_1* - descrive una pausa da $\frac{1}{4}$;
- *P2_2_25_1* della voce *P2_2* - descrive una nota (G#) da $\frac{2}{4}$ la cui durata è aumentata dalla presenza del punto di valore.

La Figura 33 mostra la temporizzazione associata agli eventi di questa misura: l'unità di VTU corrisponde alla durata di $\frac{1}{16}$.

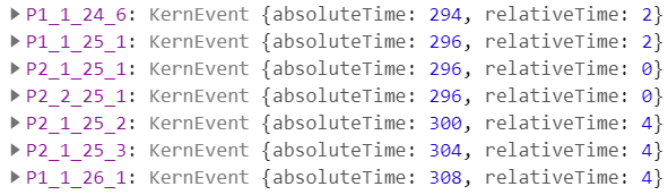


Figura 33: KernDocument - spine eventi misura 25

All'interno del brano sono presenti cambi di chiave (misure 52 e 57 parte *P1*) e alterazioni ad inizio battuta (misura 44 parti *P1* e *P2*): la Figura 34 illustra la codifica di questi eventi, con gli oggetti *KernClef* e *KernKeySignature*, che si aggiungono a quelli individuati all'inizio del brano (tra i quali figurano anche quelli relativi alle indicazioni di tempo).

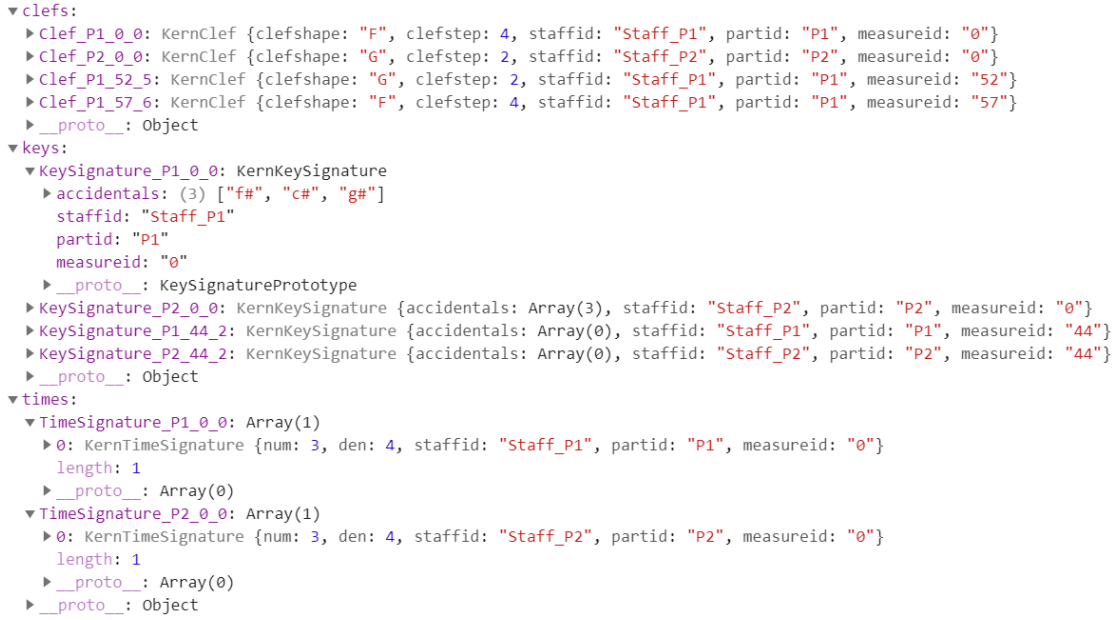


Figura 34: KernDocument - eventi *KernClef* e *KernKeySignature*

Gli attributi completi dell'oggetto KernDocument generato dal file sono rappresentati in Figura 35: anche in questo caso risultano popolati in maniera coerente, come descritto in Tabella 11.

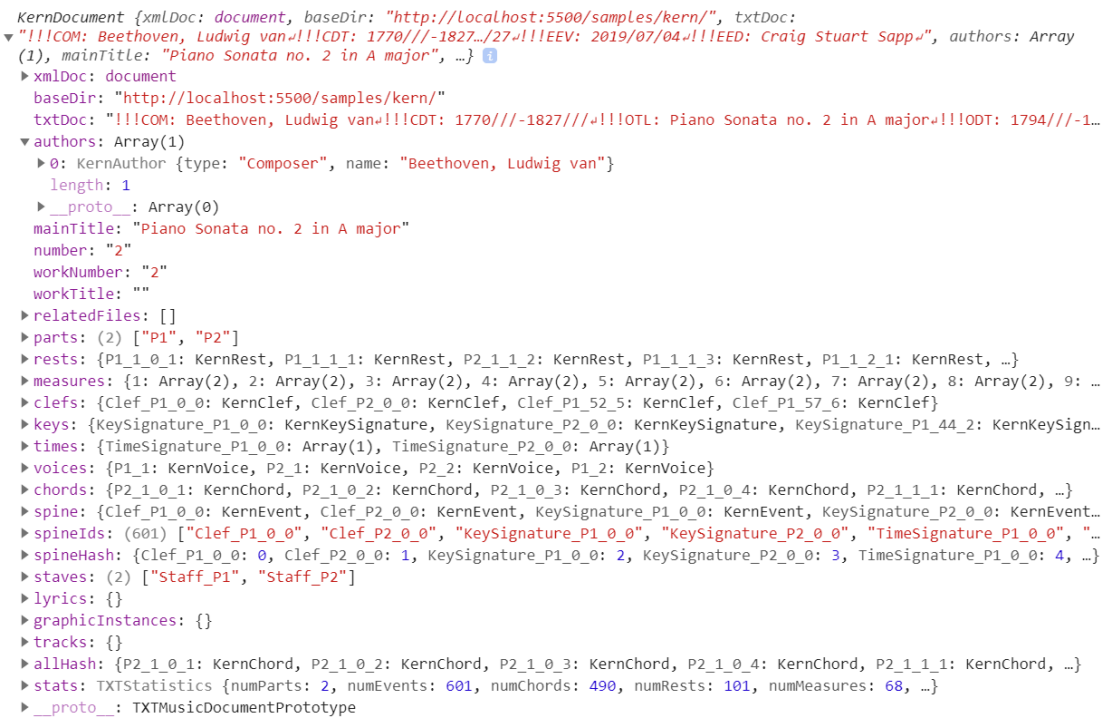


Figura 35: Piano Sonata no. 2 in A major - KernDocument

Tabella 11: Analisi attributi oggetto KernDocument

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document vuoto
txtDoc	ok - popolato con il contenuto testuale del file
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente
number	ok - popolato correttamente
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - popolato correttamente
authors	ok - popolato con informazioni del compositore
relatedFiles	ok - vuoto (non trattato per i documenti Kern)
spine	ok - popolato correttamente con gli eventi e il timing
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - popolato con i due pentagrammi
parts	ok - popolato con le due parti

measures	ok - popolate misure dalla numero 1 alla numero 68. Le note del gruppetto iniziale, per le quali non è specificata la misura di appartenenza, sono state assegnate alla misura 0
clefs	ok - popolato con le chiavi presenti nel brano per entrambe le parti
keys	ok - popolato con alterazioni in chiave presenti nel brano per entrambe le parti
times	ok - popolato con le indicazioni di tempo ad inizio brano per entrambe le parti
voices	ok - popolato con le informazioni delle quattro voci presenti (due per parte)
chords	ok - gli accordi presenti sono ben formati, sono presenti punti e legature di valore, non sono presenti gruppi irregolari
rests	ok - le pause presenti sono state trattate correttamente
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato
lyrics	ok - vuoto (non sono presenti sillabe)
graphicInstances	ok - vuoto (non trattato per i documenti Kern)
tracks	ok - vuoto (non trattato per i documenti Kern)

Segue un estratto del file originale che descrive il contenuto musicale delle misure presenti in Figura 31.

Listing 5.3: estratto sonata02-3.krn

```
[...]
!!!COM: Beethoven, Ludwig van
!!!CDT: 1770///-1827///
!!!OTL: Piano Sonata no. 2 in A major
!!!ODT: 1794///-1795///
!!!ODE: Joseph Haydn
!!!OMD: Scherzo: Allegretto
!!!OPS: 2
!!!ONM: 2
!!!OMV: 3
**kern  **kern  **dynam
[...]
=22     =22     =22
.       (8bq    .
```

```
8C#L        ()4 a#      .
8 c#        .           .
8D#         4 g#        .
8B          .           .
8D#         4 f##t )    .
8A#J        .           .
.           (16 e#qLL               .
.           16 f##qJJ               .
=23         =23         =23
8G#L        4 g#)       .
8 d#        .           .
8B          (4 dd#'     .
8 d#        .           .
8G#         4 dd #')    .
8 d#J       .           .
=24         =24         =24
8A#L        (2 dd#      .
8 d#        .           .
8 c#        .           .
8 d#        .           .
8A#         4 f##)      .
8 d#J       .           .
=25         =25         =25
*           *^          *
2 .B  2 .d#             4 r         2 .g#     <
.           (4 dd#'     .           .
.           4 dd #')    .           .
=26         =26         =26         =26
2 .B  2 .f#             2 .dd#      4 r       .
.           .           (>4an'>     .
.           .           4 a'>)      [ [
=27         =27         =27         =27
2 .E  2 .B  2 .e        4 r         2 .g#     >
.           (4 ddn'     .           .
.           4 dd ')     .           ] ]
=28         =28         =28         =28
2 .A  2 .e  2 .cc#      4 r         .
.           .           (>4gn'>     .
.           .           4 g'>)      .
=29         =29         =29         =29
```

```
!          !          !          !LO:DY: r j
2.D  2.A   2.d        4r         2. f#      pp
.          (4dd'      .          .
.          4dd')      .          .
[...]
```

5.3 IEEE 1599

5.3.1 Ave Maria (ave_maria.xml)

Per quanto riguarda gli oggetti *IEEE1599Document*, ottenuti a partire dai file in formato IEEE 1599, è possibile effettuare un primo confronto tra i risultati statistici prodotti dall'algoritmo di parsing e quelli generati dal *ieee1599-framework*: l'operazione è facilitata dalla disponibilità della scheda relativa nel Player Multimediale, mostrata in Figura 36

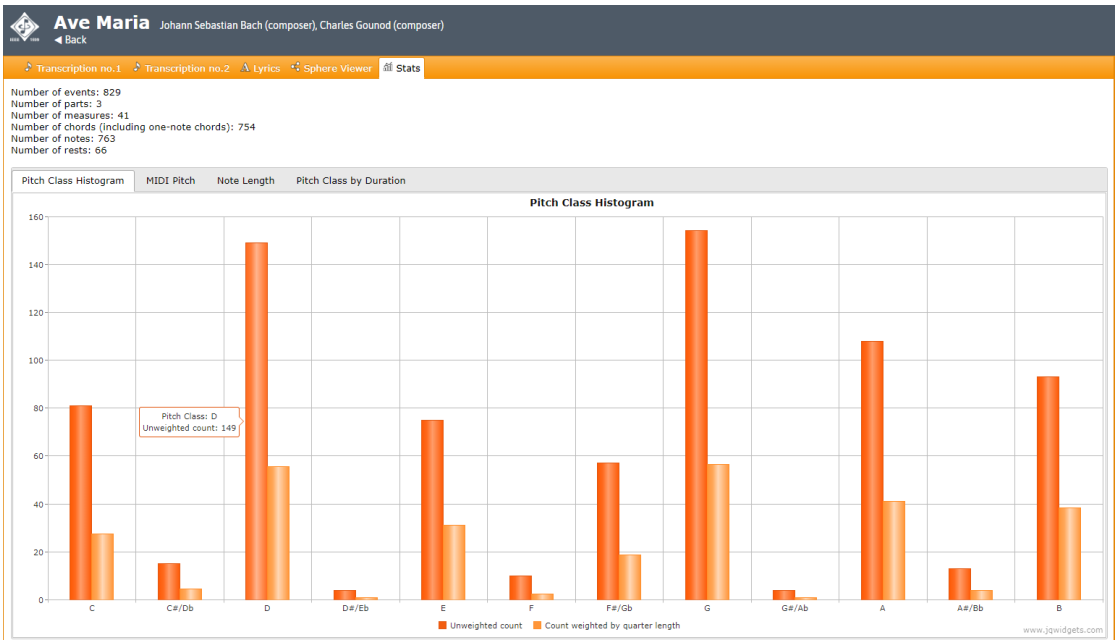


Figura 36: Ave Maria - statistiche Player Multimediale

I risultati combaciano con quelli mostrati in Figura 37: il valore evidenziato in rosso corrisponde a quello presente nel tooltip del grafico del Player Multimediale.

```
▼stats: XMLStatistics
   numParts: 3
   numEvents: 829
   numChords: 754
   numRests: 66
   numMeasures: 41
   numNotes: 763
 ▶pitchClass: (12) [81, 15, 149, 4, 75, 10, 57, 154, 4, 108, 13, 93]
 ▶pitchClassWeighted: (12) [27.5, 4.5, 55.75, 1, 31, 2.5, 18.75, 56.5, 1, 41, 4, 38.25]
 ▶midiPitch: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 7, 0,…
 ▶midiPitchWeighted: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, …
 ▶chordLength: {1: 26, 2: 25, 4: 3, 0.5: 64, 0.25: 628, 0.125: 8}
 ▶noteLength: {1: 29, 2: 25, 4: 3, 0.5: 64, 0.25: 628, 0.125: 14}
 ▶restLength: {1: 44, 0.125: 5, 0.25: 14, 0.5: 3}
 ▶noteByLengthPC: {1: {…}, 2: {…}, 4: {…}, 0.5: {…}, 0.25: {…}, 0.125: {…}}
 ▶__proto__: StatisticsPrototype
```


Figura 37: Ave Maria - *stats* IEEE1599Document

Anche le informazioni riconducibili a layer differenti dal Logic, utilizzate dal Player Multimediale, risultano importate correttamente. Per quanto riguarda il layer Notational, ad esempio, si può facilmente verificare che i contenuti del file originale, rappresentato in Figura 38, sono stati riportati fedelmente nell'attributo *graphicInstances* (Figura 39).


```
▼<notational>
  ▶<graphic_instance_group description="Transcription no.1">...</graphic_instance_group>
  ▼<graphic_instance_group description="Transcription no.2">
    ▶<graphic_instance encoding_format="image_jpeg" file_format="image_jpeg" file_name="score_files/transcription_no.2/ave_maria_t2_pag01.jpg"
     measurement_unit="pixels" position_in_group="1">...</graphic_instance>
    ▶<graphic_instance encoding_format="image_jpeg" file_format="image_jpeg" file_name="score_files/transcription_no.2/ave_maria_t2_pag02.jpg"
     measurement_unit="pixels" position_in_group="2">...</graphic_instance>
    ▶<graphic_instance encoding_format="image_jpeg" file_format="image_jpeg" file_name="score_files/transcription_no.2/ave_maria_t2_pag03.jpg"
     measurement_unit="pixels" position_in_group="3">...</graphic_instance>
    ▼<graphic_instance encoding_format="image_jpeg" file_format="image_jpeg" file_name="score_files/transcription_no.2/ave_maria_t2_pag04.jpg"
     measurement_unit="pixels" position_in_group="4">
       <graphic_event event_ref="Voix_1_voice0_measure31_ev0" upper_left_x="143" upper_left_y="112" lower_right_x="174" lower_right_y="164"/>
       <graphic_event event_ref="Voix_1_voice0_measure31_ev1" upper_left_x="301" upper_left_y="111" lower_right_x="328" lower_right_y="164"/>
```


Figura 38: Ave Maria - Layer Notational file IEEE 1599

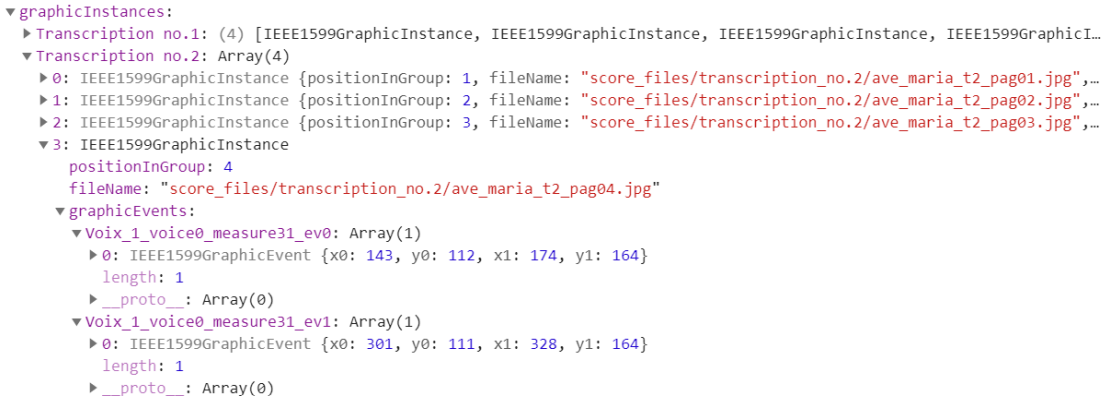


Figura 39: Ave Maria - *graphicInstances* IEEE1599Document

Confrontando le informazioni relative al layer Audio si giunge ad esito analogo: la Figura 40 riporta il contenuto dell'attributo *tracks*, che corrisponde alle tracce codificate nel file originale in Figura 41.

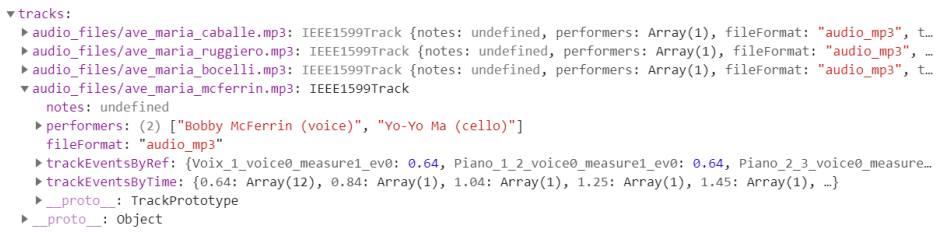


Figura 40: Ave Maria - *tracks* IEEE1599Document

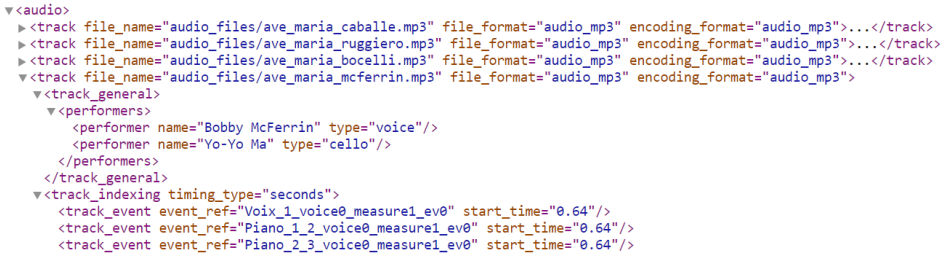


Figura 41: Ave Maria - layer Audio file IEEE 1599

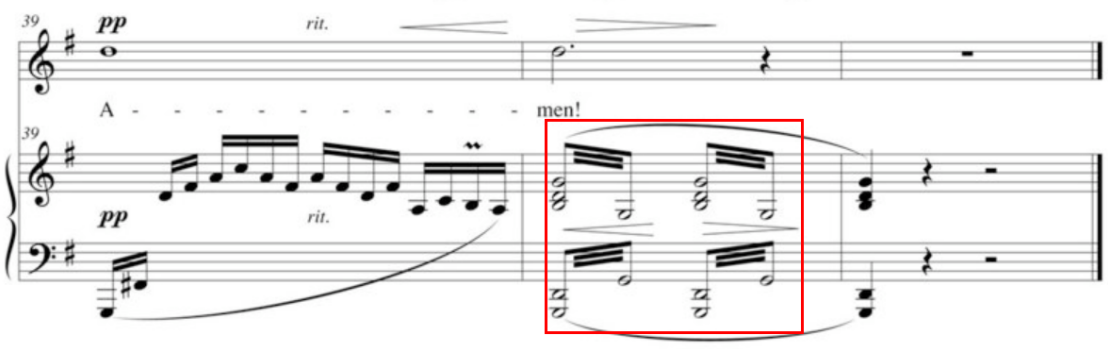


Figura 42: Ave Maria - partitura misure 39-40-41

La Figura 42 mostra le misure 39-41 della partitura (*Transcription no.1*).

Analizzando il file IEEE 1599 si può verificare che risultano codificate delle tuple, in corrispondenza dei tremolo in misura 40, per le quali non sono esplicitati i valori di tutti gli attributi: un esempio è riportato in Figura 43.

```
▼<measure number="40">
  ▼<voice voice_item_ref="Piano_1_2_0_voice">
    ▼<chord event_ref="Piano_1_2_voice0_measure40_ev0">
      ▼<duration den="32" num="1">
          <tuplet_ratio enter_den="" enter_num="" in_den="" in_num=""/>
        </duration>
      ▼<notehead>
          <pitch actual_accidental="natural" octave="4" step="B"/>
        </notehead>
      ▼<notehead>
          <pitch actual_accidental="natural" octave="5" step="D"/>
        </notehead>
      ▼<notehead>
          <pitch actual_accidental="natural" octave="5" step="G"/>
        </notehead>
    </chord>
```

Figura 43: Ave Maria - IEEE 1599 tuplet

Dato che il file originale risulta essere stato generato automaticamente con il plugin di esportazione dal software Finale, si presume che si tratti di un refuso. L'algoritmo di parsing gestisce comunque correttamente anche questa casistica, come mostrato in Figura 44, assegnando il valore *NaN* ai corrispondenti attributi obbligatori degli oggetti *IEEE1599Tuplet*: dato che *NaN* è di tipo Number, le convenzioni stabilite per il modello risultano rispettate.

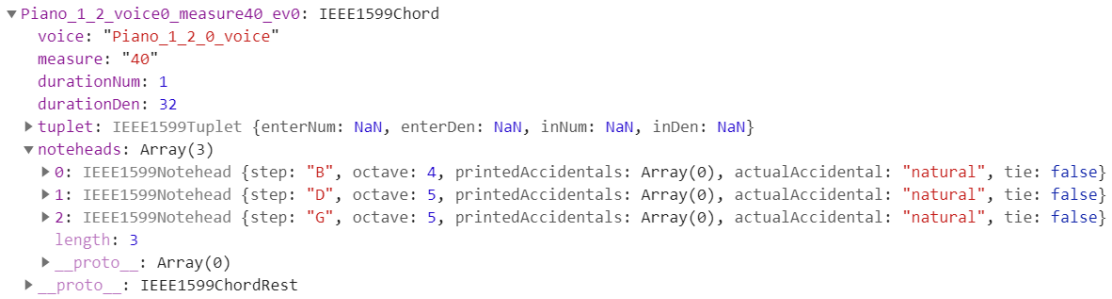


Figura 44: Ave Maria - *IEEE1599Tuplet* IEEE1599Document

Gli attributi completi dell'oggetto IEEE1599Document generato dal file sono rappresentati in Figura 45.

Figura 45: Ave Maria - IEEE1599Document

La Tabella 12 riassume gli esiti dell'analisi dei singoli attributi dell'oggetto IEEE1599Document.

Tabella 12: Analisi attributi IEEE1599Document

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document corrispondente al file XML originale
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente
number	ok - vuoto (non presente nel file originale)
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - popolato con informazioni dei compositori
relatedFiles	ok - vuoto (non presente nel file originale)
spine	ok - popolato correttamente con gli eventi e con aggiunta del timing assoluto
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - popolato con i tre pentagrammi presenti
parts	ok - popolato con le tre parti
measures	ok - popolate misure dalla numero 1 alla 41
clefs	ok - popolato con le chiavi presenti ad inizio brano per tutte le parti
keys	ok - popolato con alterazioni in chiave presenti ad inizio brano per tutte le parti
times	ok - popolato con le indicazioni di tempo presenti ad inizio brano per tutte le parti
voices	ok - popolato con le informazioni delle tre voci presenti (una per parte)
chords	ok - gli accordi presenti sono ben formati, i punti presenti vengono riportati, i valori assenti dai gruppi irregolari vengono gestiti correttamente
rests	ok - le pause presenti sono state trattate correttamente
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato coerentemente con le statistiche visualizzate dal Player Multimediale
lyrics	ok - popolato correttamente con le sillabe e gli eventuali trattini separatori
graphicInstances	ok - popolato con le due rappresentazioni grafiche

tracks	ok - popolato con le quattro tracce audio

5.3.2 Eleanor Rigby (eleanor_rigby.xml)

Le statistiche prodotte per il brano in questione, al completamento del parsing, sono riportate nella Figura 46.

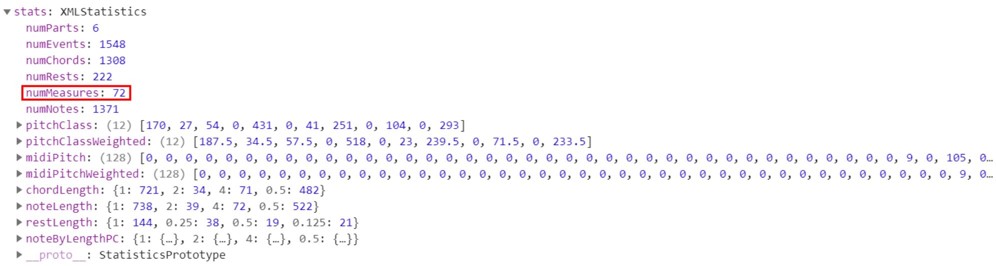


Figura 46: Eleanor Rigby - IEEE1599Document stats

Il risultato elaborato dall'*ieee1599-framework*, visualizzato dal Player Multimediale in Figura 47, presenta una differenza nel conteggio del numero di misure: il valore non è intero, come ci si attenderebbe.

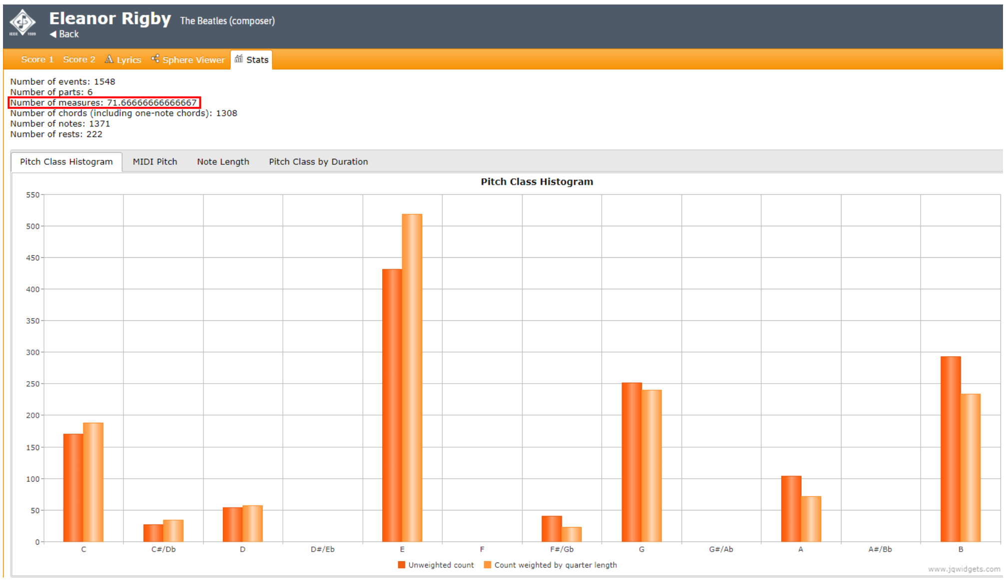


Figura 47: Eleanor Rigby - statistiche Player Multimediale

In casi come questo, il modello proposto può essere utilizzato come punto di partenza per effettuare del troubleshooting: ad esempio, verificando l'attributo *measures* dell'oggetto IEEE1599Document (Figura 48) risulta evidente che nella misura 35 non sono stati codificati eventi per due parti che invece risultano presenti in tutte le altre battute del brano (*chorus2* e *violin_i3*,); controllando il file XML di origine si ha conferma dell'anomalia (Figura 49) che causa l'errore nell'algoritmo di conteggio utilizzato dal Player Multimediale.

```
▶ 34: (6) ["vocal1", "chorus2", "violin_i3", "viola4", "violin_ii5", "cello6"]
▶ 35: (4) ["vocal1", "viola4", "violin_ii5", "cello6"]
▶ 36: (6) ["vocal1", "chorus2", "violin_i3", "viola4", "violin_ii5", "cello6"]
```

Figura 48: Eleanor Rigby - *measures* IEEE1599Document

```
▼<measure number="34">
  ▼<voice voice_item_ref="chorus2_voice1">
    ▼<rest event_ref="chorus2_meas34_voice1_ev1">
       <duration num="1" den="1"> </duration>
     </rest>
   </voice>
 </measure>
▼<measure number="36">
  ▼<voice voice_item_ref="chorus2_voice1">
    ▼<rest event_ref="chorus2_meas36_voice1_ev1">
       <duration num="1" den="1"> </duration>
     </rest>
   </voice>
 </measure>
```

Figura 49: Eleanor Rigby - assenza della misura 35 nella parte *chorus2* IEEE 1599

I risultati ottenuti dall'algoritmo di parsing, salvati nell'attributo *stats*, sono quindi quelli che corrispondono all'effettivo numero di misure codificate nel file.

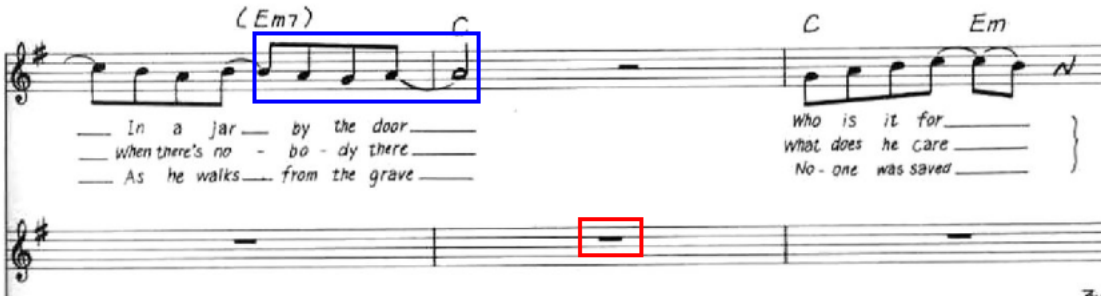


Figura 50: Eleanor Rigby - partitura misure 34-36

La pausa evidenziata in rosso nel pentagramma della parte *Chorus2* in Figura 50 dovrebbe essere codificata nella misura 35.

Approfondendo l'analisi delle sillabe presenti nell'attributo *lyrics*, mostrato in Figura 51, si nota come queste corrispondano in maniera fedele alla codifica XML originale in Figura 52.

```
▸ 120: IEEE1599Syllable {startEventRef: "vocal1_meas34_voice1_ev2", hyphen: false, text: "When"}
▸ 121: IEEE1599Syllable {startEventRef: "vocal1_meas34_voice1_ev3", hyphen: false, text: "there's"}
▸ 122: IEEE1599Syllable {startEventRef: "vocal1_meas34_voice1_ev4", hyphen: true, text: "no"}
▸ 123: IEEE1599Syllable {startEventRef: "vocal1_meas35_voice1_ev2", hyphen: true, text: "bo"}
▸ 124: IEEE1599Syllable {startEventRef: "vocal1_meas35_voice1_ev3", hyphen: false, text: "dy"}
▸ 125: IEEE1599Syllable {startEventRef: "vocal1_meas35_voice1_ev4", hyphen: false, text: "there"}
```

Figura 51: Eleanor Rigby - *IEEE1599Syllable* misure 34-35

```
<syllable start_event_ref="vocal1_meas34_voice1_ev2" hyphen="no">When</syllable>
<syllable start_event_ref="vocal1_meas34_voice1_ev3" hyphen="no">there's</syllable>
<syllable start_event_ref="vocal1_meas34_voice1_ev4" hyphen="yes">no</syllable>
<syllable start_event_ref="vocal1_meas35_voice1_ev2" hyphen="yes">bo</syllable>
<syllable start_event_ref="vocal1_meas35_voice1_ev3" hyphen="no">dy</syllable>
<syllable start_event_ref="vocal1_meas35_voice1_ev4" hyphen="no">there</syllable>
```

Figura 52: Eleanor Rigby - IEEE 1599 syllable misure 34-35

Tuttavia emergono indizi che sembrano indicare la presenza di altri errori nel file originale: le sillabe "bo-dy there" dovrebbero essere correlate alle ultime 3 note della misura 34 della parte *vocal1*, mentre l'id evento suggerisce la presenza nella misura 35. Analizzando il contenuto dell'attributo *chords* si trova conferma che le 4 note da $\frac{1}{8}$ che chiudono la misura 34, evidenziate in blu nella Figura 50, sono state erroneamente codificate come appartenenti alla misura 35: la Figura 53 conferma che l'errore è presente già nel file originale.

Ai fini dell'esperienza utente di fruizione contenuti con il Player Multimediale, difetti di questo tipo, che sembrano interessare anche altre voci, possono essere impercettibili (al netto dell'errore di conteggio già evidenziato): se nello spine è codificata la corretta sequenza temporale degli eventi, come avviene per questo pezzo, i vari tipi di rappresentazione risulteranno comunque correttamente sincronizzati tra di loro durante l'esecuzione. Chiaramente andrà rettificato il documento XML IEEE 1599, che si è appurato essere risultante da una procedura di conversione automatizzata con plugin *MuseScore*, per via delle implicazioni di carattere semantico e dei possibili impatti su strumenti terzi (ad esempio tools che effettuano analisi automatizzata di carattere musicologico o altro). La disamina completa della problematica originale, che non è oggetto del presente lavoro, potrà comportare anche una revisione del codice sorgente del plugin di esportazione (qualora non si individuassero errori nel file all'origine della procedura di conversione).

```
<measure number="35">
  <voice voice_item_ref="vocal1_voice1">
    <chord event_ref="vocal1_meas35_voice1_ev1">
      <duration num="1" den="8"> </duration>
      <notehead>
        <pitch step="B" octave="5" actual_accidental="natural"/>
      </notehead>
    </chord>
    <chord event_ref="vocal1_meas35_voice1_ev2">
      <duration num="1" den="8"> </duration>
      <notehead>
        <pitch step="A" octave="5" actual_accidental="natural"/>
      </notehead>
    </chord>
    <chord event_ref="vocal1_meas35_voice1_ev3">
      <duration num="1" den="8"> </duration>
      <notehead>
        <pitch step="G" octave="5" actual_accidental="natural"/>
      </notehead>
    </chord>
    <chord event_ref="vocal1_meas35_voice1_ev4">
      <duration num="1" den="8"> </duration>
      <notehead>
        <pitch step="A" octave="5" actual_accidental="natural"/>
        <tie/>
      </notehead>
    </chord>
  </voice>
</measure>
```

Figura 53: Eleanor Rigby - IEEE1599 syllable misure 34-35

Proseguendo con l'analisi degli altri attributi, si verifica che le informazioni relative a chiavi, alterazioni in chiave e indicazioni di tempo sono rappresentate correttamente, come rappresentato nelle Figura 54.

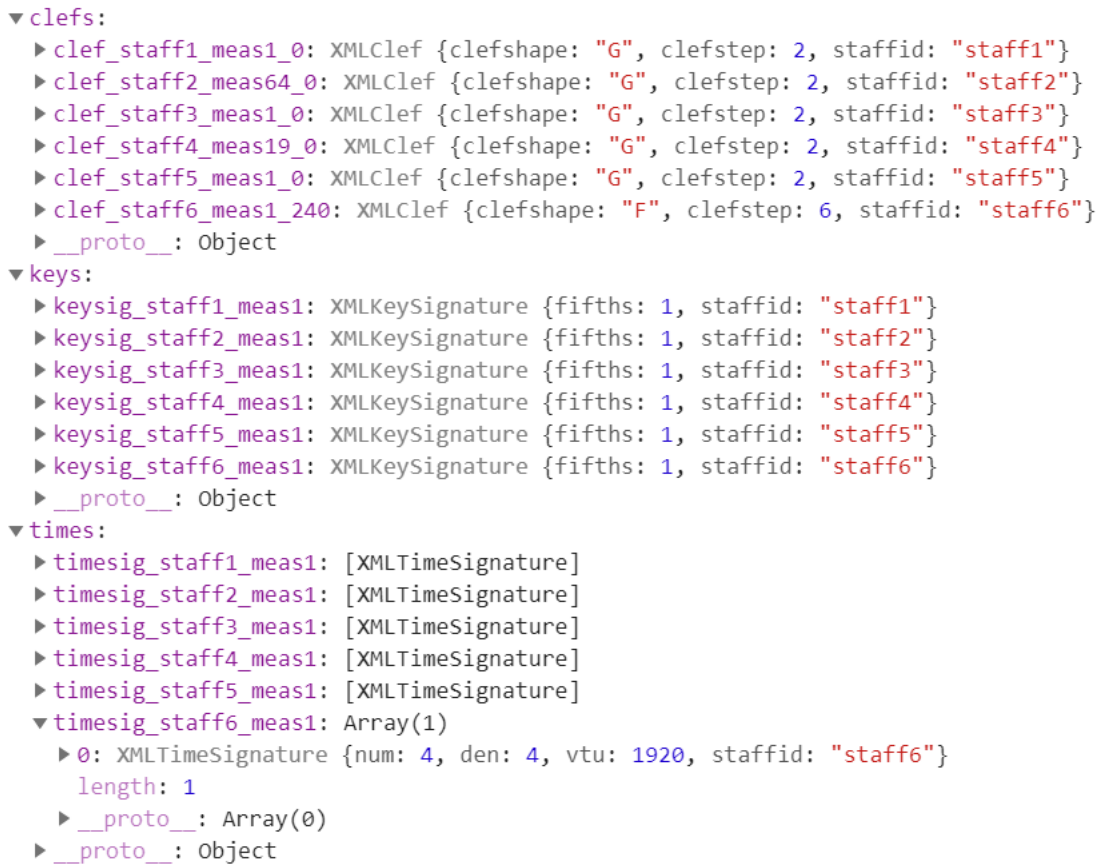


Figura 54: Eleanor Rigby - *Clefs*, *Keys*, *Times* IEEE1599Document

Gli attributi completi dell'oggetto IEEE1599Document generato dal file sono rappresentati in Figura 55.

Figura 55: Eleanor Rigby - IEEE1599Document

La Tabella 13 riassume gli esiti dell'analisi dei singoli attributi dell'oggetto IEEE1599Document.

Tabella 13: Analisi attributi IEEE1599Document

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document corrispondente al file XML originale
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente
number	ok - vuoto (non presente nel file originale)
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - popolato con informazioni del compositore
relatedFiles	ok - vuoto (non presente nel file originale)
spine	ok - popolato correttamente con gli eventi del file originale e con aggiunta del timing assoluto; alcuni id-evento hanno errori semantici (errata associazione tra misure)
spineIds	ok - popolato correttamente; stesse considerazioni di *spine* per id-evento

spineHash	ok - popolato correttamente; stesse considerazioni di *spine* per id-evento
staves	ok - popolato con sei pentagrammi
parts	ok - popolato con sei parti
measures	ok - popolate misure dalla numero 1 alla 72; una delle misure ha meno elementi delle altre (come nel file originale)
clefs	ok - popolato con le chiavi presenti ad inizio brano per tutte le parti
keys	ok - popolato con alterazioni in chiave presenti ad inizio brano per tutte le parti
times	ok - popolato con le indicazioni di tempo presenti ad inizio brano per tutte le parti
voices	ok - popolato con informazioni di sei voci che, stranamente, afferiscono solo a 5 delle sei parti codificate (come nel file originale)
chords	ok - gli accordi presenti sono ben formati, i punti presenti vengono riportati, non sono presenti gruppi irregolari
rests	ok - le pause presenti sono state trattate correttamente
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato correttamente, conteggio misure differisce da statistiche Player Multimediale
lyrics	ok - popolato correttamente con le sillabe e gli eventuali trattini separatori delle due parti vocali (solista e coro)
graphicInstances	ok - popolato con le due rappresentazioni grafiche (partitura orchestrale e di pianoforte)
tracks	ok - popolato con una traccia audio

5.3.3 Il mio ben quando verrà (il_mio_ben_quando_verra.xml)

Questo brano è tra i più corposi e ricchi di caratteristiche nella libreria IEEE 1599. L'analisi delle statistiche sui risultati ottenuti dall'algoritmo di parsing (Figura 56) e dall'*ieee1599-framework* (Figura 57) mostra valori identici.


```
▼stats: XMLStatistics
    numParts: 10
    numEvents: 5136
    numChords: 3394
    numRests: 1653
    numMeasures: 150
    numNotes: 3407
  ▶pitchClass: (12) [703, 6, 264, 45, 352, 672, 14, 372, 13, 459, 389, 118]
  ▶pitchClassWeighted: (12) [575.5, 5, 163.5, 25, 254.75, 493.5, 7, 275.5, 9.5, 314.25, 264, 80]
  ▶midiPitch: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 6, 0, 0, 0, 3, 32,…
  ▶midiPitchWeighted: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4.5, 0, 0,…
  ▶chordLength: {1: 605, 2: 315, 0.5: 2410, 0.25: 64}
  ▶noteLength: {1: 616, 2: 315, 0.5: 2410, 0.25: 66}
  ▶restLength: {0.25: 664, 0.125: 379, 0.5: 610}
  ▶noteByLengthPC: {1: {…}, 2: {…}, 0.5: {…}, 0.25: {…}}
  ▶__proto__: StatisticsPrototype
```


Figura 56: Il mio ben quando verrà - *stats* IEEE1599Document

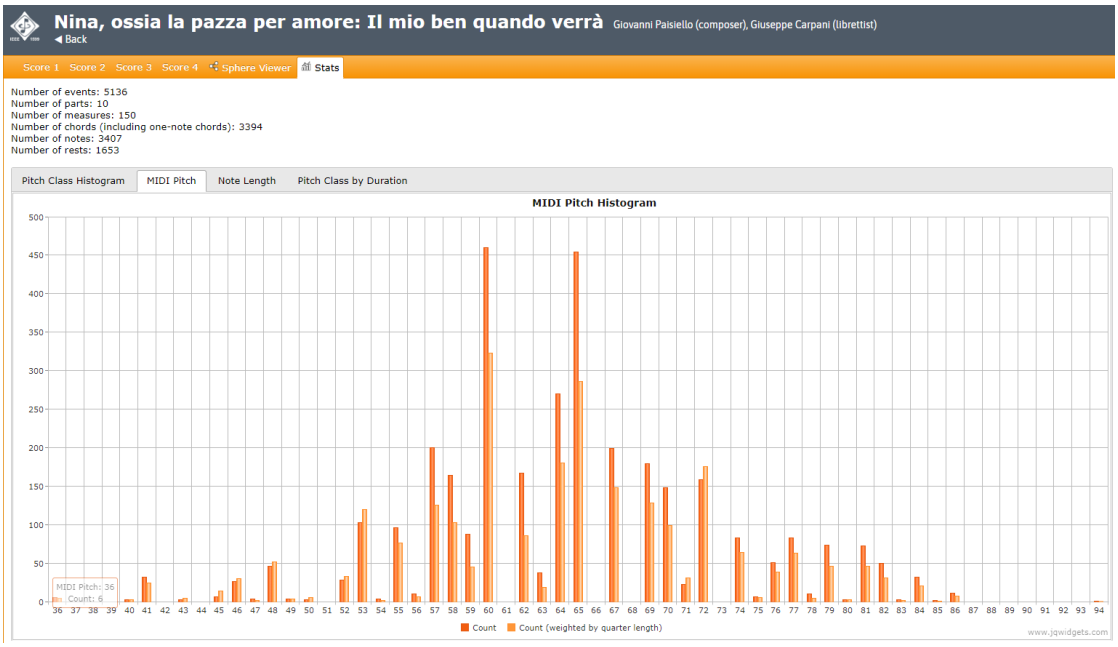


Figura 57: Il mio ben quando verrà - statistiche Player Multimediale

Come si evince dalla Figura 58, i collegamenti ai file esterni, correlati con il brano e presenti nell'XML originale (Figura 59), vengono correttamente riportati nell'attributo *relatedFiles*.


```
▼relatedFiles: Array(4)
  ▶0: IEEE1599RelatedFile {fileName: "related_files/muti.jpg", description: "Il Maestro Riccardo Muti", spineStartRef: "nina_001_01", spineEndRef: …
  ▶1: IEEE1599RelatedFile {fileName: "related_files/8700.jpg", description: "Immagine di scena", spineStartRef: "nina_003_01", spineEndRef: "nina_0…
  ▶2: IEEE1599RelatedFile {fileName: "related_filesantonacci.jpg", description: "Caterina Antonacci", spineStartRef: "nina_005_01", spineEndRef: "n…
  ▼3: IEEE1599RelatedFile
     fileName: "related_files/8699.jpg"
     description: "Immagine di scena"
     spineStartRef: "nina_007_01"
     spineEndRef: "nina_009_01"
    ▶__proto__: RelatedFileProtoype
    length: 4
  ▶__proto__: Array(0)
```


Figura 58: Il mio ben quando verrà - *relatedFiles* IEEE1599Document


```
▼<ieee1599 version="1.0">
  ▼<general>
    ▼<description>
       <main_title>Il mio ben quando verrà</main_title>
       <author type="composer">Paisiello, Giovanni</author>
       <author type="librettist">Carpani, Giuseppe</author>
       <work_title>Nina, ossia la pazza per amore</work_title>
     </description>
    ▼<related_files>
       <related_file file_name="related_files/muti.jpg" file_format="image_jpeg" encoding_format="image_jpeg" spine_start_ref="nina_001_01" spine_end_ref="nina_003_01"
       description="Il Maestro Riccardo Muti"/>
       <related_file file_name="related_files/8700.jpg" file_format="image_jpeg" encoding_format="image_jpeg" spine_start_ref="nina_003_01" spine_end_ref="nina_005_01"
       description="Immagine di scena"/>
       <related_file file_name="related_filesantonacci.jpg" file_format="image_jpeg" encoding_format="image_jpeg" spine_start_ref="nina_005_01" spine_end_ref="nina_007_01"
       description="Caterina Antonacci"/>
       <related_file file_name="related_files/8699.jpg" file_format="image_jpeg" encoding_format="image_jpeg" spine_start_ref="nina_007_01" spine_end_ref="nina_009_01"
       description="Immagine di scena"/>
     </related_files>
   </general>
```


Figura 59: Il mio ben quando verrà - IEEE 1599 layer General

La Figura 60 mostra le prime battute della partitura autografa, mappata in *graphicInstances* come '*Autograph manuscript*': le coordinate degli eventi grafici corrispondenti alla prima terzina di note di violino, evidenziate in rosso, risultano associate agli eventi descritti in Figura 61.

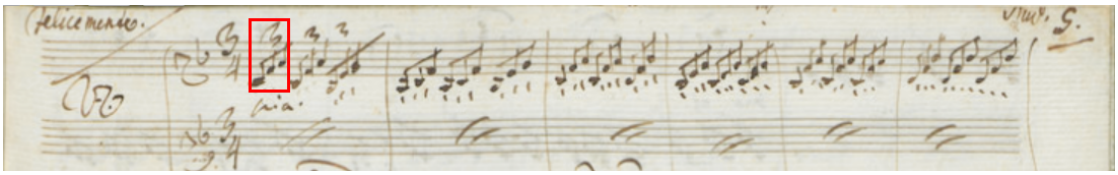


Figura 60: Il mio ben quando verrà - parti di violino, misure 1-6

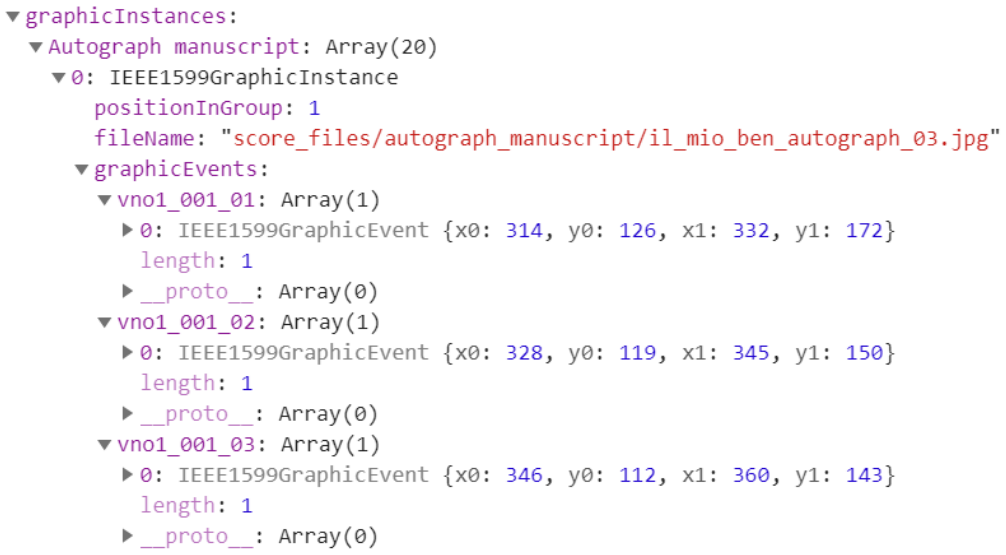


Figura 61: Il mio ben quando verrà - *graphicInstances* IEEE1599Document

La temporizzazione di questi eventi nell'attributo *spine* è corretta, come rappresentato in Figura 62.

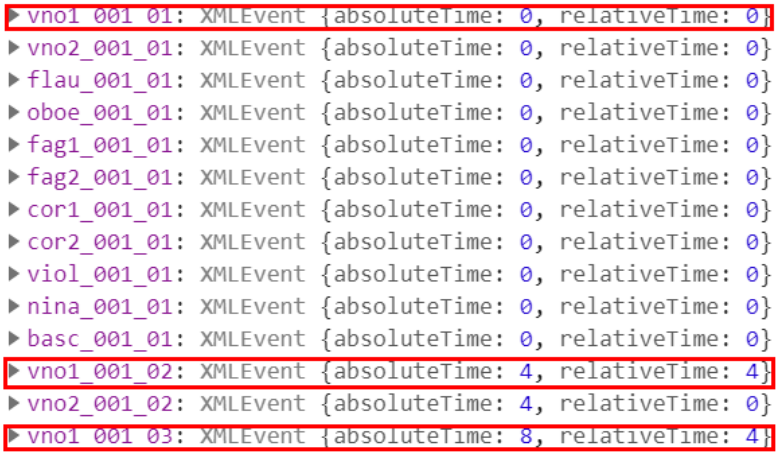


Figura 62: Il mio ben quando verrà - *spine* IEEE1599Document

Anche gli oggetti *IEEE1599Chords* corrispondenti, presenti nell'attributo *chords*, risultano codificati in maniera corretta, sia per quanto riguarda i valori delle tuple (*IEEE1599Tuplet*) sia per quanto riguarda l'altezza delle note (*IEEE1599Notehead*): la Figura 63 mostra la rappresentazione di questi elementi, con una vista di dettaglio sull'ultima nota della terzina.

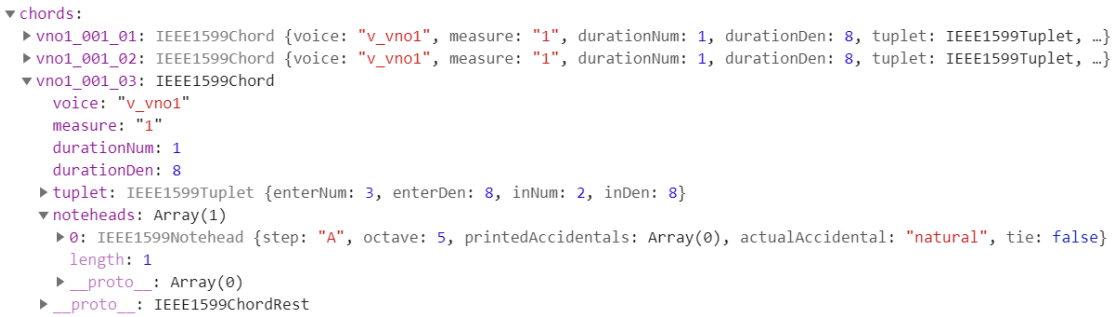


Figura 63: Il mio ben quando verrà - *chords* IEEE1599Document

Gli attributi completi dell'oggetto IEEE1599Document generato dal file sono rappresentati in Figura 64.

Figura 64: Il mio ben quando verrà - IEEE1599Document

La Tabella 14 riassume gli esiti dell'analisi dei singoli attributi dell'oggetto IEEE1599Document.

Tabella 14: Analisi attributi IEEE1599Document

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document corrispondente al file XML originale
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente

number	ok - vuoto (non presente nel file originale)
workTitle	ok - popolato correttamente
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - popolato con informazioni di compositore e librettista
relatedFiles	ok - popolato con i riferimenti di 4 file esterni
spine	ok - popolato correttamente con gli eventi del file originale e con aggiunta del timing assoluto
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - popolato con dieci pentagrammi
parts	ok - popolato con dieci parti
measures	ok - popolate misure dalla numero 1 alla 150
clefs	ok - popolato con le chiavi presenti ad inizio brano per tutte le parti
keys	ok - popolato con alterazioni in chiave presenti ad inizio brano per tutte le parti
times	ok - popolato con le indicazioni di tempo presenti ad inizio brano per tutte le parti
voices	ok - popolato con informazioni di dieci voci corrispondenti alle parti
chords	ok - gli accordi presenti sono ben formati, i punti e le legature presenti vengono riportati, così come i gruppi irregolari
rests	ok - le pause presenti sono state trattate correttamente, anche quando fanno parte di gruppi irregolari
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato correttamente
lyrics	ok - vuoto (non presente nel file originale)
graphicInstances	ok - popolato con le quattro rappresentazioni grafiche (manoscritti, partitura e libretto)
tracks	ok - popolato con una traccia audio e una traccia video

5.4 MusicXML

5.4.1 Echigo-Jishi (Echigo-Jishi.musicxml)

La Figura 65 rappresenta le prime battute di questo brano di musica popolare giapponese: il file originale, in formato *score-timewise*, viene processato e ricondotto alla rappresentazione *score-partwise*.

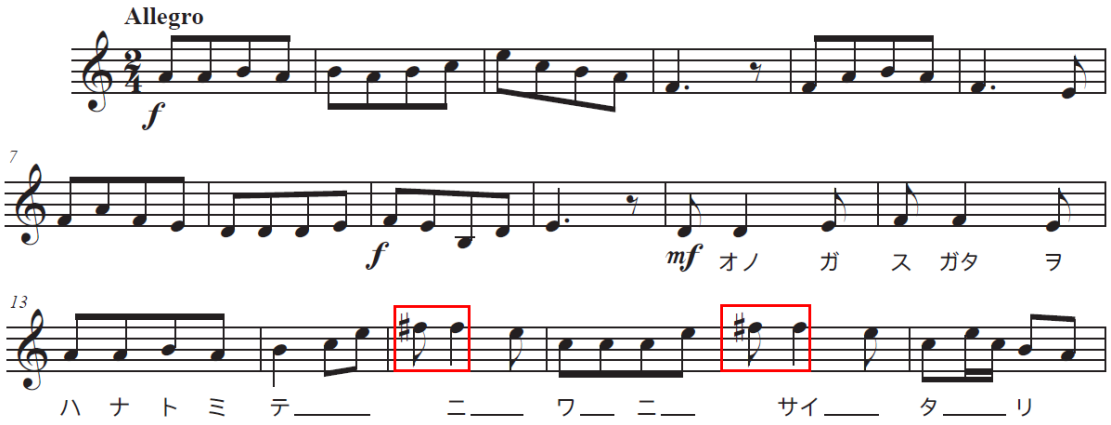


Figura 65: Echigo-Jishi - partitura misure 1-18

Lo spartito contiene un'unica voce (Figura 66), composta da singole note, con le sillabe del testo cantato in lingua originale: rappresenta un esempio ideale per verificare il corretto funzionamento di base dell'algoritmo di processing.


```
▼clefs:
  ▶Clef_P1_1_01: XMLClef {clefshape: "G", clefstep: 2, staffid: "Staff_P1_1", partid: "P1", measureid: "1"}
  ▶__proto__: Object
▼keys:
  ▶KeySignature_P1_01: XMLKeySignature {fifths: 0, partid: "P1", measureid: "1"}
  ▶__proto__: Object
▼times:
  ▼TimeSignature_P1_01: Array(1)
    ▶0: XMLTimeSignature {num: 2, den: 4, vtu: 256, partid: "P1", measureid: "1"}
     length: 1
    ▶__proto__: Array(0)
  ▶__proto__: Object
▼voices:
  ▶P1_1: XMLVoice {staffid: "Staff_P1_1", partid: "P1"}
  ▶__proto__: Object
```


Figura 66: Echigo-Jishi - *clefs*, *keys*, *times*, *voices* MusicXMLDocument

La Figura 67 documenta le statistiche memorizzate nell'attributo *stats*: risultano 172 eventi, totalizzati da 163 accordi, composti di altrettante note, 6 pause e 3 eventi ad inizio misura (chiave, alterazioni in chiave e tempo mostrati in Figura 66). Risulta anche un conteggio di 4 note con pitch *F#*, evidenziato in rosso in figura, che corrisponde al numero di note della stessa altezza contrassegnate, con il medesimo colore, in partitura.

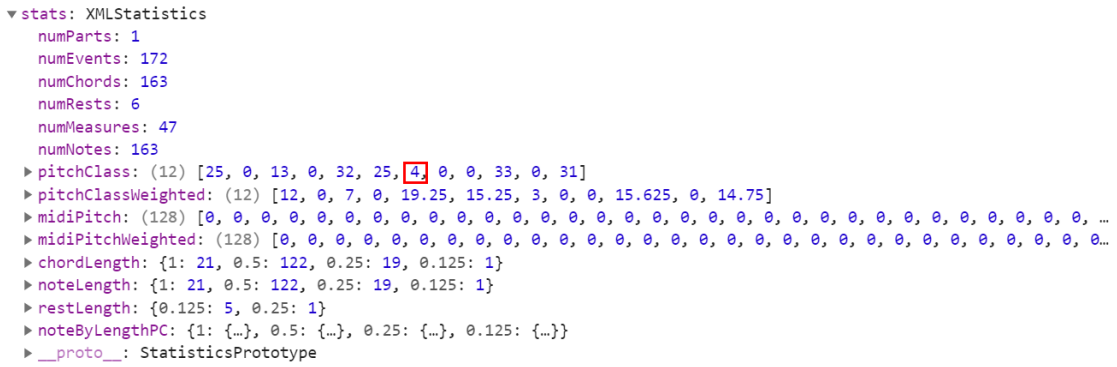


Figura 67: Echigo-Jishi - *stats* MusicXMLDocument

Dalla Figura 68, che mostra la codifica *MusicXMLChord* delle prime due note in misura 15, si può verificare come questi eventi siano stati correttamente codificati.

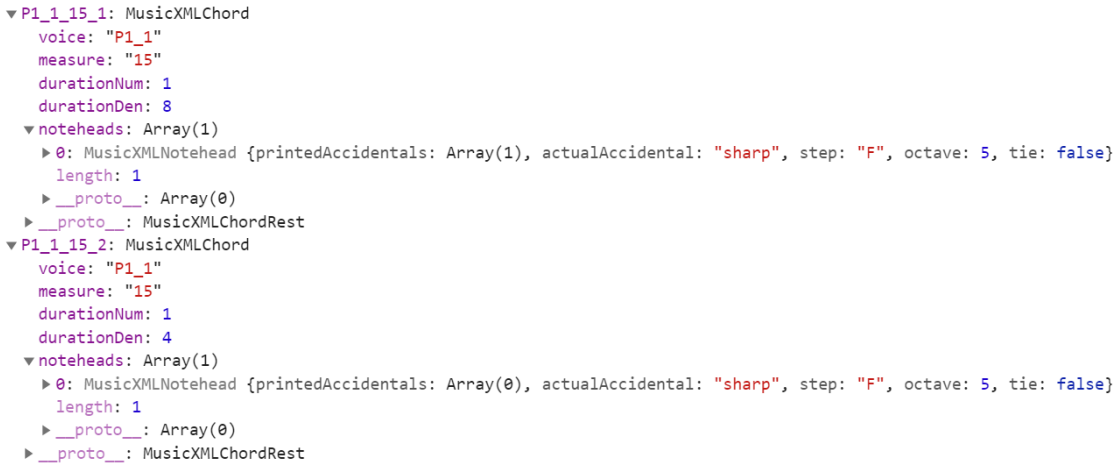


Figura 68: Echigo-Jishi - *chords* MusicXMLDocument

Anche l'attributo *spine* contiene la corretta sequenza degli eventi, come si può evincere dalla Figura 69 che rappresenta le prime 4 misure.

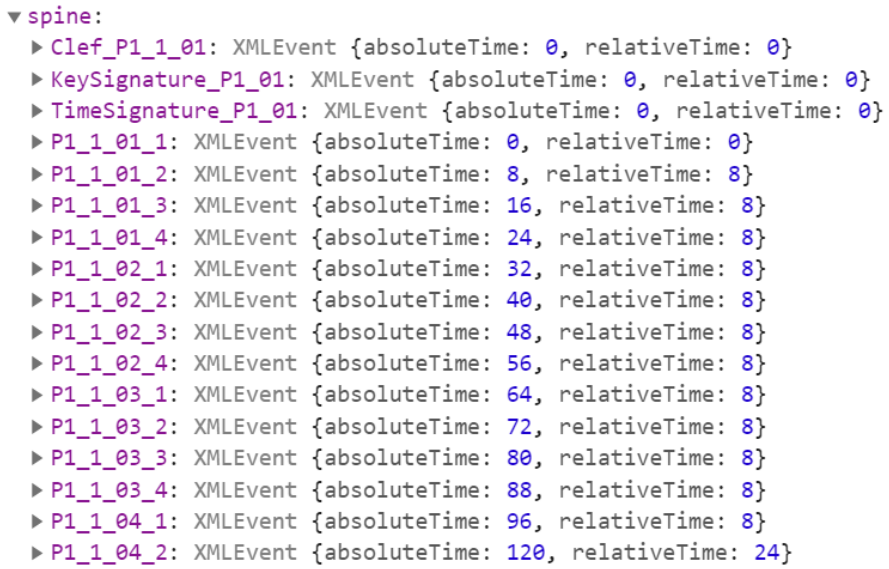


Figura 69: Echigo-Jishi - *spine* MusicXMLDocument

La Figura 70 illustra gli attributi contenenti i metadati frutto del processo di parsing.

Figura 70: Echigo-Jishi - metadati MusicXMLDocument

Gli oggetti *MusicXMLSyllable* afferenti alle misure 11, 12 e 13, salvati nella lista *Lyric_1* dell'attributo *lyrics*, sono rappresentati in Figura 71.


```
▶ 0: MusicXMLSyllable {startEventRef: "P1_1_11_2", hyphen: false, text: "オノ"}
▶ 1: MusicXMLSyllable {startEventRef: "P1_1_11_3", hyphen: false, text: "ガ"}
▶ 2: MusicXMLSyllable {startEventRef: "P1_1_12_1", hyphen: false, text: "ス"}
▶ 3: MusicXMLSyllable {startEventRef: "P1_1_12_2", hyphen: false, text: "ガタ"}
▶ 4: MusicXMLSyllable {startEventRef: "P1_1_12_3", hyphen: false, text: "ヲ"}
▶ 5: MusicXMLSyllable {startEventRef: "P1_1_13_1", hyphen: false, text: "ハ"}
▶ 6: MusicXMLSyllable {startEventRef: "P1_1_13_2", hyphen: false, text: "ナ"}
▶ 7: MusicXMLSyllable {startEventRef: "P1_1_13_3", hyphen: false, text: "ト"}
▶ 8: MusicXMLSyllable {startEventRef: "P1_1_13_4", hyphen: false, text: "ミ"}
```


Figura 71: Echigo-Jishi - *lyrics* MusicXMLDocument

Gli attributi completi dell'oggetto MusicXMLDocument generato dal file sono rappresentati in Figura 72.


```
▼ MusicXmlDocument
  ▶ xmlDoc: document
    baseDir: "http://127.0.0.1:5500/samples/musicxml/timewise/"
    mainTitle: "越後獅子"
    number: ""
    workTitle: ""
    workNumber: ""
  ▶ authors: (2) [XMLAuthor, XMLAuthor]
  ▶ relatedFiles: []
  ▶ spine: {Clef_P1_1_01: XMLEvent, KeySignature_P1_01: XMLEvent, TimeSignature_P1_01: XMLEvent, P1_1_01_1: XMLEvent, P1_1_…
  ▶ spineIds: (172) ["Clef_P1_1_01", "KeySignature_P1_01", "TimeSignature_P1_01", "P1_1_01_1", "P1_1_01_2", "P1_1_01_3", "P…
  ▶ spineHash: {Clef_P1_1_01: 0, KeySignature_P1_01: 1, TimeSignature_P1_01: 2, P1_1_01_1: 3, P1_1_01_2: 4, …}
  ▶ staves: ["Staff_P1_1"]
  ▶ parts: ["P1"]
  ▶ measures: {1: Array(1), 2: Array(1), 3: Array(1), 4: Array(1), 5: Array(1), 6: Array(1), 7: Array(1), 8: Array(1), 9: A…
  ▶ clefs: {Clef_P1_1_01: XMLClef}
  ▶ keys: {KeySignature_P1_01: XMLKeySignature}
  ▶ times: {TimeSignature_P1_01: Array(1)}
  ▶ voices: {P1_1: XMLVoice}
  ▶ chords: {P1_1_01_1: MusicXMLChord, P1_1_01_2: MusicXMLChord, P1_1_01_3: MusicXMLChord, P1_1_01_4: MusicXMLChord, P1_1_0…
  ▶ rests: {P1_1_04_2: MusicXMLRest, P1_1_10_2: MusicXMLRest, P1_1_36_1: MusicXMLRest, P1_1_39_1: MusicXMLRest, P1_1_45_3: …
  ▶ allHash: {P1_1_01_1: MusicXMLChord, P1_1_01_2: MusicXMLChord, P1_1_01_3: MusicXMLChord, P1_1_01_4: MusicXMLChord, P1_1_…
  ▶ stats: XMLStatistics {numParts: 1, numEvents: 172, numChords: 163, numRests: 6, numMeasures: 47, …}
  ▶ lyrics: {Lyric_1: Array(113)}
  ▶ graphicInstances: {}
  ▶ tracks: {}
  ▶ __proto__: XMLMusicDocumentPrototype
```


Figura 72: Echigo-Jishi - MusicXMLDocument

La Tabella 15 riassume gli esiti dell'analisi dei singoli attributi dell'oggetto MusicXMLDocument.

Tabella 15: Analisi attributi MusicXMLDocument

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document corrispondente al file XML originale
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente
number	ok - vuoto (non presente nel file originale)
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - popolato con informazioni arrangiatori
relatedFiles	ok - vuoto (non trattato per i documenti MusicXML)
spine	ok - popolato correttamente
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - popolato con un pentagramma
parts	ok - popolato con una parte
measures	ok - popolate misure dalla numero 1 alla 47
clefs	ok - popolato con la chiave presenti ad inizio brano
keys	ok - popolato con le alterazioni in chiave ad inizio brano
times	ok - popolato con l'indicazioni di tempo ad inizio brano
voices	ok - popolato con una voce
chords	ok - gli accordi presenti, composti da singole note, sono ben formati. I punti di valore sono correttamente gestiti, non sono presenti legature di valore o gruppi irregolari
rests	ok - le pause presenti sono state trattate correttamente
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato correttamente
lyrics	ok - contiene una singola voce con le sillabe del testo giapponese
graphicInstances	ok - vuoto (non trattato per i documenti MusicXML)
tracks	ok - vuoto (non trattato per i documenti MusicXML)

5.4.2 Ave Maria (SchbAvMaSample.musicxml)

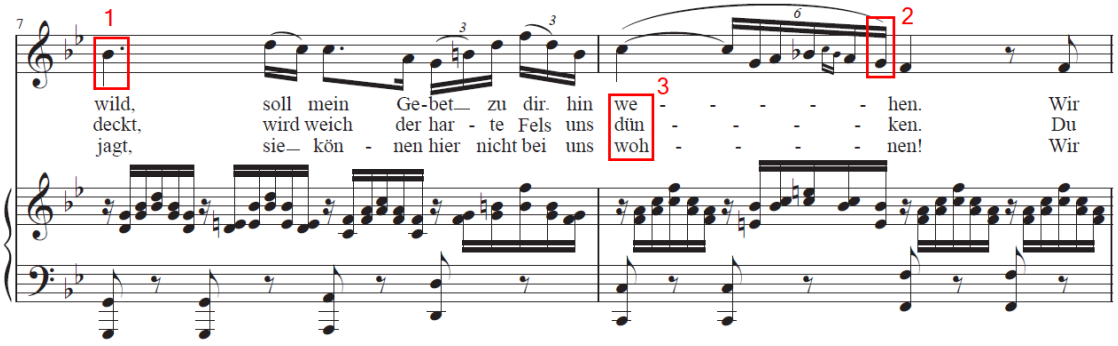


Figura 73: Ave Maria - misure 7-8

Questo brano è composto da una parte vocale (singola voce) e da una parte di pianoforte (doppia voce): la Figura 73 mostra alcune battute che contengono elementi di interesse per la verifica della funzionalità dell'algoritmo di parsing.

La nota contenuta nel primo rettangolo, appartenente alla parte vocale, è alterata in chiave e presenta un punto di durata: la Figura 74 mostra la codifica corrispondente salvata nell'attributo *chord*. Le informazioni corrispondono a quelle


```
▼P1_1_7_1: MusicXMLChord
    voice: "P1_1"
    measure: "7"
    durationNum: 1
    durationDen: 4
    augmentationDots: 1
  ▼noteheads: Array(1)
    ▶0: MusicXMLNotehead {printedAccidentals: Array(0), actualAccidental: "flat", step: "B", octave: 4, tie: false}
      length: 1
    ▶__proto__: Array(0)
  ▶__proto__: MusicXMLChordRest
```


Figura 74: Ave Maria - *MusicXMLChord*

codificate nel file originale in Figura 75: il valore della nota da $\frac{1}{4}$ ($divisions = 48$) coincide a quello di *duration* scalato del punto di valore ($\frac{72}{1.5} = 48$).

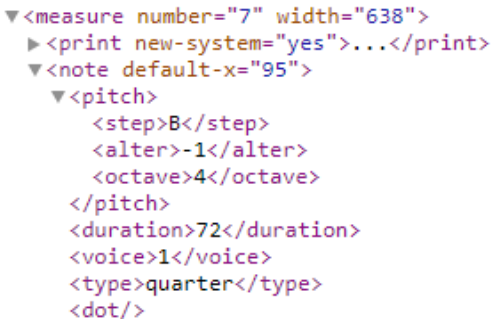


Figura 75: Ave Maria - nota nel file MusicXML

Il secondo rettangolo in Figura 73, afferente alla parte vocale, contiene una nota appartenente ad un gruppo irregolare (sestina di sedicesimi): anche questo elemento risulta correttamente memorizzato nell'oggetto corrispondente, come mostrato in Figura 76.

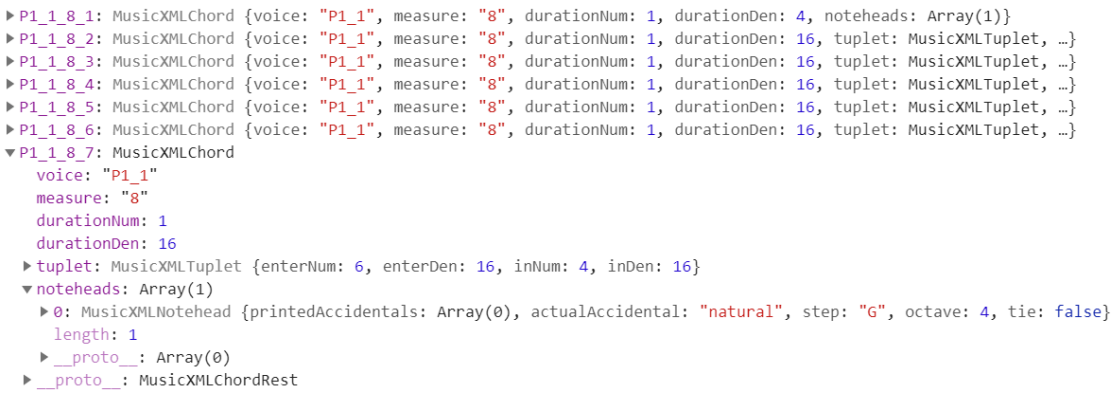


Figura 76: Ave Maria - *MusicXMLTuplet*

La Figura 77 mostra il frammento XML originale.

```
▼<note default-x="164">
  ▼<pitch>
      <step>G</step>
      <octave>4</octave>
    </pitch>
    <duration>8</duration>
    <voice>1</voice>
    <type>16th</type>
  ▼<time-modification>
      <actual-notes>6</actual-notes>
      <normal-notes>4</normal-notes>
    </time-modification>
```

Figura 77: Ave Maria - nota della sestina nel file MusicXML

Analizzando il contenuto dell'attributo *spine*, si può verificare che anche la temporizzazione è stata gestita correttamente (il sedicesimo in sestina dura 4 VTU): la Figura 78 ne mostra un estratto che contiene l'intero gruppo irregolare.

```
▶ P1_1_8_2: XMLEvent {absoluteTime: 696, relativeTime: 4}
▶ P2_1_8_7: XMLEvent {absoluteTime: 696, relativeTime: 0}
▶ P2_2_8_3: XMLEvent {absoluteTime: 696, relativeTime: 0}
▶ P1_1_8_3: XMLEvent {absoluteTime: 700, relativeTime: 4}
▶ P2_1_8_8: XMLEvent {absoluteTime: 700, relativeTime: 0}
▶ P1_1_8_4: XMLEvent {absoluteTime: 704, relativeTime: 4}
▶ P2_1_8_9: XMLEvent {absoluteTime: 704, relativeTime: 0}
▶ P1_1_8_5: XMLEvent {absoluteTime: 708, relativeTime: 4}
▶ P2_1_8_10: XMLEvent {absoluteTime: 708, relativeTime: 0}
▶ P2_2_8_4: XMLEvent {absoluteTime: 708, relativeTime: 0}
▶ P1_1_8_6: XMLEvent {absoluteTime: 712, relativeTime: 4}
▶ P2_1_8_11: XMLEvent {absoluteTime: 712, relativeTime: 0}
▶ P1_1_8_7: XMLEvent {absoluteTime: 716, relativeTime: 4}
▶ P2_1_8_12: XMLEvent {absoluteTime: 716, relativeTime: 0}
```

Figura 78: Ave Maria - *spine* MusicXMLDocument

Il terzo rettangolo di Figura 73 mostra le sillabe afferenti alla parte vocale: le parole di ogni riga risultano memorizzate in una differente lista nell'attributo *lyrics*. La Figura 79 mostra gli oggetti *MusicXMLSyllable* ricavati dai nodi originali rappresentati in Figura 80.

```
▶ 32: MusicXMLSyllable {startEventRef: "P1_1_8_1", hyphen: true, text: "we"}
▶ 32: MusicXMLSyllable {startEventRef: "P1_1_8_1", hyphen: true, text: "dün"}
▶ 32: MusicXMLSyllable {startEventRef: "P1_1_8_1", hyphen: true, text: "woh"}
```

Figura 79: Ave Maria - *MusicXMLSyllable* Lyric_1, Lyric_2 e Lyric_3

```
▼<lyric default-y="-82" justify="left" number="1">
   <syllabic>begin</syllabic>
   <text>we</text>
  </lyric>
▼<lyric default-y="-104" justify="left" number="2">
   <syllabic>begin</syllabic>
   <text>dün</text>
  </lyric>
▼<lyric default-y="-127" justify="left" number="3">
   <syllabic>begin</syllabic>
   <text>woh</text>
  </lyric>
```

Figura 80: Ave Maria - nodi *lyric* in MusicXML

Gli attributi completi dell'oggetto MusicXMLDocument generato dal file sono rappresentati in Figura 81.

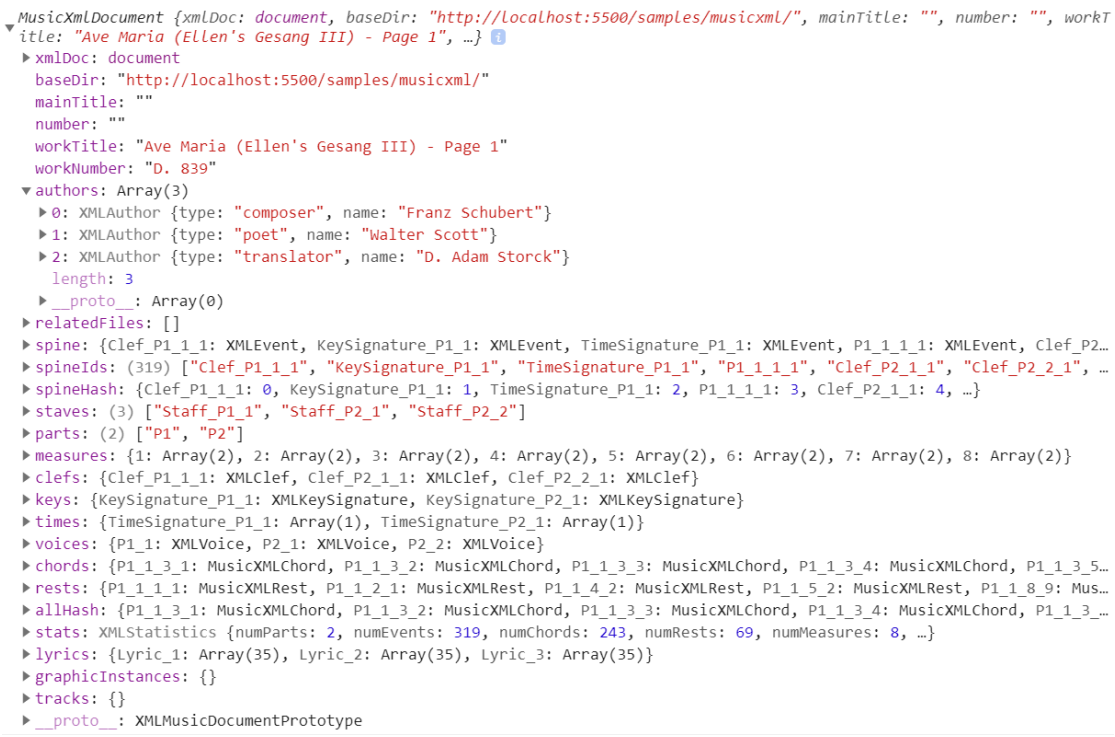


Figura 81: Ave Maria - MusicXMLDocument

La Tabella 16 riassume gli esiti dell'analisi dei singoli attributi dell'oggetto MusicXMLDocument.

Tabella 16: Analisi attributi MusicXMLDocument

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document corrispondente al file XML originale
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - vuoto (non presente nel file originale)
number	ok - vuoto (non presente nel file originale)
workTitle	ok - popolato correttamente
workNumber	ok - popolato correttamente
authors	ok - popolato con informazioni di compositore, poeta e traduttore
relatedFiles	ok - vuoto (non trattato per i documenti MusicXML)
spine	ok - popolato correttamente
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente

staves	ok - popolato con tre pentagrammi
parts	ok - popolato con due parti (voce e piano)
measures	ok - popolate misure dalla numero 1 alla 8
clefs	ok - popolato con le chiavi presenti ad inizio brano nei 3 pentagrammi
keys	ok - popolato con le alterazioni in chiave per la prima misura delle due parti (2 pentagrammi su 3, come nel file originale)
times	ok - popolato con le indicazioni di tempo per le due parti presenti (2 pentagrammi su 3, come nel file originale)
voices	ok - popolato con informazioni sulle 3 voci (una vocale, due di piano)
chords	ok - gli accordi presenti sono ben formati; i punti (anche doppi), le legature di valore e i gruppi irregolari sono correttamente gestiti
rests	ok - le pause presenti sono state trattate correttamente, anche quando fanno parte di gruppi irregolari
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato correttamente
lyrics	ok - popolato con le sillabe delle 3 linee di voce, trattini gestiti correttamente
graphicInstances	ok - vuoto (non trattato per i documenti MusicXML)
tracks	ok - vuoto (non trattato per i documenti MusicXML)

5.4.3 Prelude to a Tragedy (ActorPreludeSample.musicxml)

Questo brano è il più corposo tra quelli presenti nel set dimostrativo MusicXML: descrive le prime 41 misure di uno spartito per orchestra di 22 elementi. Le statistiche in Figura 82 mostrano che l'algoritmo di parsing ha codificato più di 3000 singoli eventi.


```
▼stats: XMLStatistics
    numParts: 22
    numEvents: 3033
    numChords: 1679
    numRests: 757
    numMeasures: 41
    numNotes: 1944
  ▶pitchClass: (12) [173, 82, 128, 119, 262, 179, 266, 218, 83, 134, 123, 177]
  ▶pitchClassWeighted: (12) [71.5, 49.25, 78.5, 45.5, 228.5, 125.5, 123.5, 86.75, 36.75, 46.5, 68.25, 136.25]
  ▶midiPitch: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 4, 4, 0, 4, 4, 30, 22, 14, 0…
  ▶midiPitchWeighted: (128) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2.5, 4.5, 0, 2.5, 4.…
  ▶chordLength: {1: 98, 2: 128, 4: 35, 0.25: 1410, 0.5: 8}
  ▶noteLength: {1: 127, 2: 182, 4: 52, 0.25: 1575, 0.5: 8}
  ▶restLength: {1: 141, 0.75: 347, 0.5: 113, 0.25: 122, 0.0625: 20, 0.125: 14}
  ▶noteByLengthPC: {1: {…}, 2: {…}, 4: {…}, 0.25: {…}, 0.5: {…}}
  ▶__proto__: StatisticsPrototype
```


Figura 82: Prelude to a Tragedy - *stats* MusicXMLDocument

Due parti del brano sono composte da doppia voce: quelle della parte *P15* afferiscono al medesimo pentagramma (*Staff_P15_1*), mentre la seconda voce della parte *P17* (*P17_2*) è rappresentata in un pentagramma ad hoc (*Staff_P17_2*). La Figura 83 illustra gli oggetti emphXMLVoice, contenuti in *voices*, che rappresentano le due casistiche descritte.


```
▶P15_1: XMLVoice {staffid: "Staff_P15_1", partid: "P15"}
▶P15_2: XMLVoice {staffid: "Staff_P15_1", partid: "P15"}
▶P16_1: XMLVoice {staffid: "Staff_P16_1", partid: "P16"}
▶P17_1: XMLVoice {staffid: "Staff_P17_1", partid: "P17"}
▶P17_2: XMLVoice {staffid: "Staff_P17_2", partid: "P17"}
```


Figura 83: Prelude to a Tragedy - *XMLVoice* MusicXMLDocument

La Figura 84 mostra alcune misure dello spartito.

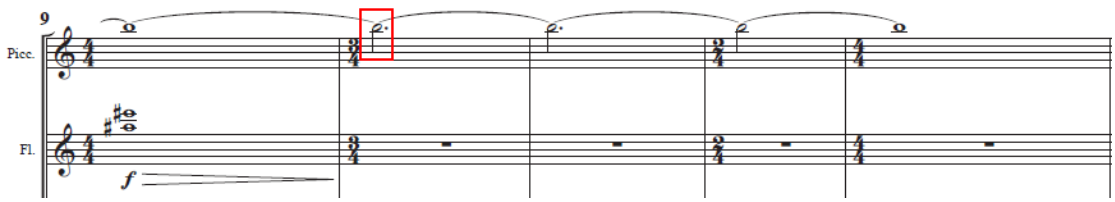


Figura 84: Prelude to a Tragedy - partitura misure 9-13 (parti *P1* e *P2*)

In partitura di può notare la presenza di frequenti cambi di tempo: questi eventi sono stati correttamente processati mediante la creazione di oggetti *XMLTimeSignature*. La Figura 85 mostra gli elementi creati per rappresentare le informazioni relative alla parte *P1*.

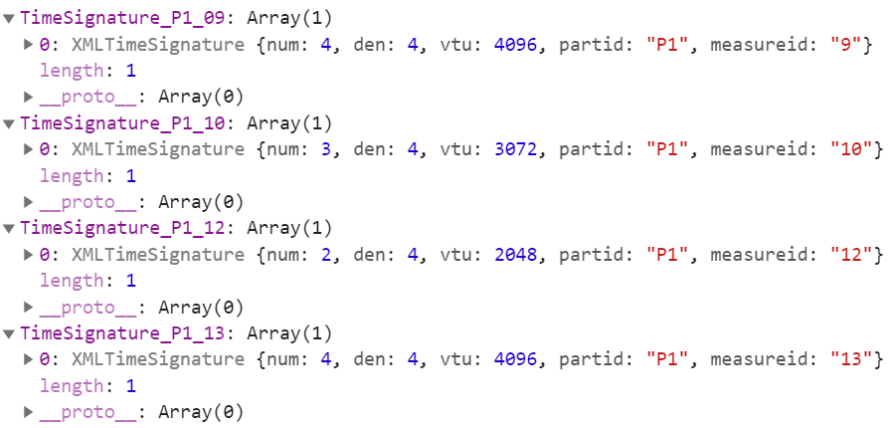


Figura 85: Prelude to a Tragedy - *XMLTimeSignature* misure 9-13 parte *P1*

La Figura 86 mostra la codifica della nota puntata da $\frac{2}{4}$ evidenziata in rosso in partitura (Figura 84): tutti gli attributi risultano popolati.

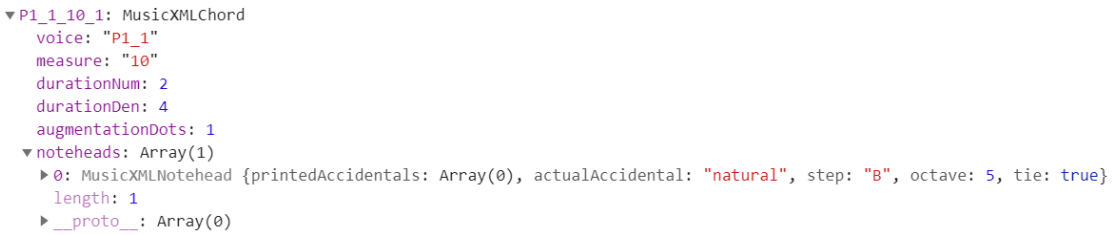


Figura 86: Prelude to a Tragedy - *chords* MusicXMLDocument

In Figura 87 è rappresentato un estratto del contenuto dell'attributo *spine* che mostra la corretta temporizzazione degli eventi afferenti alle parti *P1* e *P2* in misura 10.

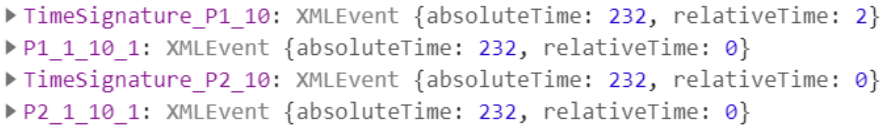


Figura 87: Prelude to a Tragedy - *spine* MusicXMLDocument

L'evento *P2_1_10_1* è una pausa, correttamente memorizzata nell'attributo *rests* con un oggetto *MusicXMLRest*: la Figura 88 mostra i dettagli dell'oggetto.

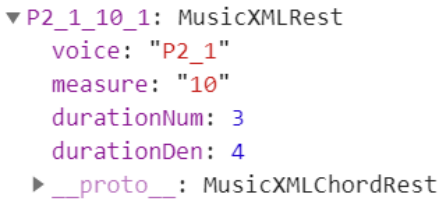


Figura 88: Prelude to a Tragedy - *rests* MusicXMLDocument

Anche le informazioni relative alle chiavi risultano correttamente codificate nell'attributo *clefs*: come si può appurare in Figura 89, risultano gestiti anche i cambi di chiave che avvengono nella misura 26 della parte *P17* (*Clef_P17_2_26*) e nella misura 38 della parte *P11* (*Clef_P11_1_38*).

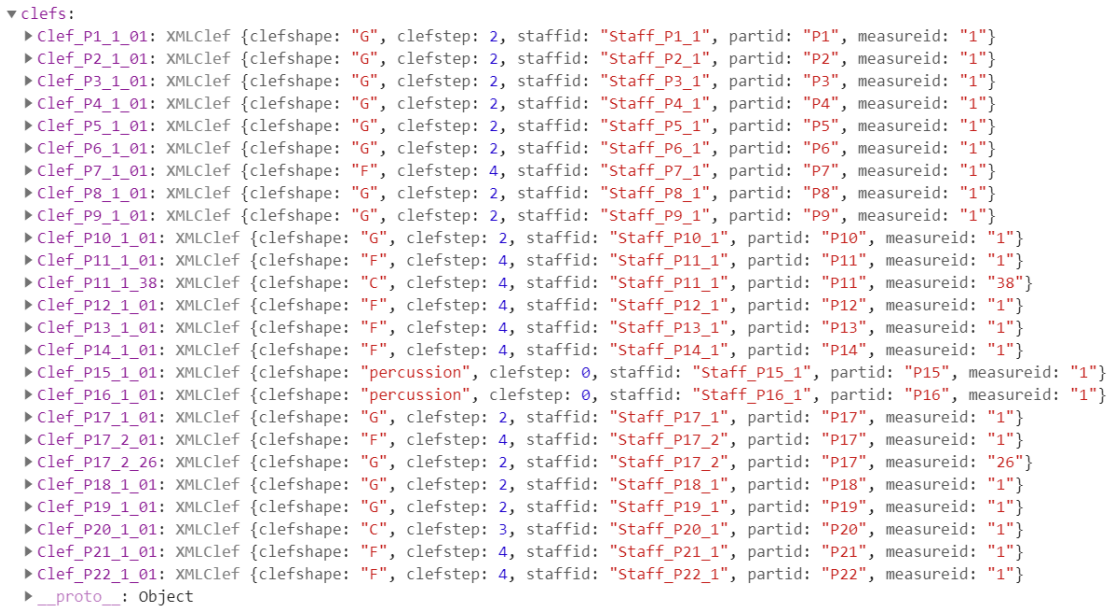


Figura 89: Prelude to a Tragedy - *XMLClef* MusicXMLDocument

Gli attributi completi dell'oggetto MusicXMLDocument generato dal file sono rappresentati in Figura 90: la Tabella 17 riassume gli esiti delle singole analisi condotte sull'oggetto MusicXMLDocument.

Figura 90: Prelude to a Tragedy - MusicXMLDocument

Tabella 17: Analisi attributi MusicXMLDocument

Attributo	**Esito verifica**
xmlDoc	ok - popolato con oggetto Document corrispondente al file XML originale
baseDir	ok - popolato con il percorso di caricamento del file
mainTitle	ok - popolato correttamente
number	ok - vuoto (non presente nel file originale)
workTitle	ok - vuoto (non presente nel file originale)
workNumber	ok - vuoto (non presente nel file originale)
authors	ok - popolato con informazioni compositore
relatedFiles	ok - vuoto (non trattato per i documenti MusicXML)
spine	ok - popolato correttamente
spineIds	ok - popolato correttamente
spineHash	ok - popolato correttamente
staves	ok - popolato con 23 pentagrammi; la seconda voce di arpa è rappresentata in un pentagramma dedicato
parts	ok - popolato con 22 parti
measures	ok - popolate misure dalla numero 1 alla 41

clefs	ok - popolato con le chiavi presenti ad inizio brano per i 23 pentagrammi e con i cambi di chiave nel corso del brano (voci P11_1 e P17_2
keys	ok - popolato con le 22 alterazioni in chiave (una per ogni parte)
times	ok - popolato con 550 indicazioni di tempo (25 cambi di tempo per ogni parte)
voices	ok - popolato con informazioni su 24 voci presenti; 2 parti hanno doppia voce (P15, P17). Il file originale conteneva anche due voci ausiliarie, non visibili in partitura, in corrispondenza dei tremolo delle parti di percussione P14 e P16: anche se correttamente processabili, sono state commentate per non introdurre rumore nella rappresentazione oggetto di analisi
chords	ok - gli accordi presenti sono ben formati. I punti e le legature di valore sono correttamente gestiti, non sono presenti gruppi irregolari
rests	ok - le pause presenti sono state trattate correttamente
allHash	ok - contiene tutti gli accordi e le pause
stats	ok - risulta popolato correttamente
lyrics	ok - vuoto (non presente nel file originale)
graphicInstances	ok - vuoto (non trattato per i documenti MusicXML)
tracks	ok - vuoto (non trattato per i documenti MusicXML)

Capitolo 6

Conclusioni e sviluppi futuri

Le verifiche condotte utilizzando la libreria di file di test, esemplificate nel precedente Capitolo 5, hanno prodotto esiti incoraggianti: nei casi analizzati, gli algoritmi di parsing hanno popolato le strutture dati specialistiche con le informazioni correttamente estrapolate dalle rappresentazioni originali. Resta inteso che queste procedure d'esempio potranno essere ulteriormente affinate nel tempo, in modo da trattare con modalità specifiche casistiche particolari già emerse (numerazione delle misure non standard per i file **kern), sviluppi evolutivi (come la gestione dei cambi multipli di chiave nella medesima misura per i file musicxml) e nuove necessità (ad esempio casistiche complesse di gruppi irregolari).

I test condotti hanno portato alla luce alcune criticità legate a rappresentazioni approssimative/non corrette delle informazioni nei formati di origine: la validazione dei file di input va chiaramente al di là del perimetro del presente lavoro, tuttavia emerge la necessità di introdurre dei meccanismi di controllo che potrebbero trovare attuazione sia a monte delle attività di processing (sviluppo di tool ad hoc per ogni formato) sia a valle delle stesse (messaggi di warning, ampliamento dei casi di test per verificare che la somma delle durate delle note corrisponda a quella complessiva prevista per la misura di appartenenza, etc.).

Il nucleo di informazioni che costituisce il modello di base include quanto necessario per ottenere una rappresentazione esaustiva dal punto di vista semantico: sulla base del processo evolutivo che nei prossimi mesi interesserà lo standard di riferimento, sarà possibile intervenire in maniera agevole per introdurre proprietà aggiuntive ritenute rilevanti (ad esempio si potrebbe prevedere la codifica delle informazioni relative alle acciaccature/appoggiature o integrare altri attributi derivati dal layer Logic IEEE 1599).

Le strutture dati astratte e le funzioni di creazione documenti si sono dimostrate sufficientemente malleabili da poter essere facilmente adattate, nella fase di sviluppo, agli utilizzi specifici implementati; gli oggetti istanziati sono stati controllati con la suite di test che non ha evidenziato difformità rispetto a quanto

previsto in fase di progettazione. Ci sono quindi buone probabilità che il modello possa essere impiegato con successo per codificare le informazioni per i tipi di file previsti ed esteso ad ulteriori formati / tipi di file d'origine: se il gruppo che lavorerà alla nuova release di IEEE 1599 lo riterrà opportuno, potrà essere d'ausilio per il processo di revisione dello standard.

Gli oggetti ottenuti si confermano compatibili con l'*ieee1599-framework*: è prefigurabile, a breve termine, un'evoluzione del Player Multimediale che consentirà, mediante l'adozione del modello proposto, di fruire contenuti musicali appartenenti a librerie multimediali codificate con standard terzi. Durate il processo di integrazione, se ritenuto opportuno, si potrà procedere ad una revisione del metodo di parsing delle statistiche in modo da potenziarne il funzionamento (ad esempio introducendo il riconoscimento dei gruppi irregolari o l'elaborazione di statistiche su intervalli armonici / melodici).

La gestione degli aspetti legati alle varie fasi del ciclo di vita del progetto ha permesso allo scrivente di approfondire e mettere in pratica diverse nozioni acquisite durante il percorso d'istruzione: dall'analisi dei requisiti fino al rilascio della versione finale, si è appurato come con un'organizzazione del lavoro basata su una formazione solida ed un approccio metodico, uniti a competenza e disponibilità da parte dei relatori, sia possibile ottenere risultati di qualità anche in situazioni di elevato stress. Non ci si poteva augurare una migliore conclusione del ciclo di studi!

I risultati ottenuti, in ultima analisi, sembrano rispondere a quanto atteso.

Appendice A

Metodi di parsing

Questa appendice contiene il codice sorgente sviluppato per il parsing dei file musicali: il funzionamento è descritto nel Capitolo 4.

A.1 Humdrum **kern

Listing A.1: codice JS per parsing file Humdrum **kern

```
parseMusicDocument() {
  try {
    this.authors = [];
    this.mainTitle = "";
    this.number = "";
    this.workNumber = "";
    this.workTitle = "";
    this.relatedFiles = [];
    this.parts = [];
    this.rests = {};
    this.measures = {};
    this.clefs = {};
    this.keys = {};
    this.times = {};
    this.voices = {};
    this.chords = {};
    this.spine = {};
    this.spineIds = [];
    this.spineHash = {};
    this.staves = [];
    this.lyrics = {};
    this.graphicInstances = {};
    this.tracks = {};
    let partsCounter = 1;
    let lyricsCounter = 1;
```

```
26      let eventsCounters = [];
27      let measureId = "0";
28      let parts = [];
29      let voices = [];
30      let spineArray = [];
31      let lines = this.txtDoc.split("\n");
32      let x = -1, v;
33      let staves = [];
34      let durations = new Set();
35      let lineCounter = 0;
36      for (let i = 0; i < lines.length; i++) {
37        let line = lines[i];
38        if (line.length === 0) {
39          continue;
40        }
41        let operations = [];
42        if (spineArray[lineCounter] === undefined) {
43          spineArray[lineCounter] = [];
44        }
45        v = -1;
46        let tokens = line.split("\t");
47        for (let j = 0; j < tokens.length; j++) {
48          if (tokens[j][0] === "!") { // a comment or reference
               record
49            if (tokens[j].match(/^!{3}/)) { // reference record (
                 string beginning with !!!)
50              let referenceRecord = tokens[j].slice(3).split(":")
                   ;
51              let referenceCode = referenceRecord[0];
52              let referenceValue = referenceRecord[1].trim();
53              // work identification
54              if (referenceCode[0] === "O") {
55                if (this.mainTitle === "") {
56                  if (referenceCode.match(/^OTL/)) {
57                    this.mainTitle = referenceValue;
58                  }
59                }
60                if (this.number === "") {
61                  if (referenceCode.match(/^ONM/)) {
62                    this.number = referenceValue;
63                  }
64                }
65                if (this.workTitle === "") {
66                  if (referenceCode.match(/^OPR/)) {
67                    this.workTitle = referenceValue;
68                  }
69                }
70                if (this.workNumber === "") {
71                  if (referenceCode.match(/^OMV/)) {
```

```
72                    this.workNumber = referenceValue;
73                  }
74                  else if (referenceCode.match(/^OPS/)) {
75                    this.workNumber = referenceValue;
76                  }
77                }
78              }
79              // authors
80              else if (referenceCode[0] === "C") {
81                if (referenceCode.match(/^COM/)) {
82                  this.authors.push(new KernAuthor(referenceValue
                      , "Composer"));
83                }
84                else if (referenceCode.match(/^COA/)) {
85                  this.authors.push(new KernAuthor(referenceValue
                      , "Attributed Composer"));
86                }
87                else if (referenceCode.match(/^COS/)) {
88                  this.authors.push(new KernAuthor(referenceValue
                      , "Suspected Composer"));
89                }
90                else if (referenceCode.match(/^COL/)) {
91                  this.authors.push(new KernAuthor(referenceValue
                      , "Composer abbreviated"));
92                }
93                else if (referenceCode.match(/^COC/)) {
94                  this.authors.push(new KernAuthor(referenceValue
                      , "Composer(s) corporate name"));
95                }
96              }
97              else if (referenceCode[0] === "L") {
98                if (referenceCode.match(/^LYR/)) {
99                  this.authors.push(new KernAuthor(referenceValue
                      , "Lyricist"));
100               }
101               else if (referenceCode.match(/^LIB/)) {
102                 this.authors.push(new KernAuthor(referenceValue
                      , "Librettist"));
103               }
104               else if (referenceCode.match(/^LAR/)) {
105                 this.authors.push(new KernAuthor(referenceValue
                      , "Arranger"));
106               }
107               else if (referenceCode.match(/^LOR/)) {
108                 this.authors.push(new KernAuthor(referenceValue
                      , "Orchestrator"));
109               }
110             }
111             else if (referenceCode.match(/^TRN/)) {
```

```
112                 this.authors.push(new KernAuthor(referenceValue,
                        "Translator"));
113               }
114             }
115           }
116           else if (tokens[j][0] === "*") { // interpretation
                  records
117             // column/part header
118             if (tokens[j] === "**kern") {
119               let partid = "P" + partsCounter;
120               this.parts.push(partid);
121               partsCounter++;
122               parts[j] = partid;
123               let staffid = "Staff_" + partid;
124               staves[partid] = staffid;
125               this.staves.push(staffid);
126               voices[j] = 1;
127               let voiceid = partid + "_" + voices[j];
128               this.voices[voiceid] = new KernVoice(staffid,
                      partid);
129               eventsCounters[j] = 0;
130             }
131             else if (tokens[j] === "**text" || tokens[j] === "**
                    silbe") {
132               // lyrics
133               let lyricId = "Lyric_" + lyricsCounter;
134               lyricsCounter++;
135               parts[j] = lyricId;
136               this.lyrics[lyricId] = [];
137             }
138             if (parts[j]) {
139               // clef
140               if (tokens[j].match(/^\*clef/)) {
141                 let clef = tokens[j].slice(5);
142                 let clefshape = clef.match(/\D+/g)[0];
143                 let clefstep = clef.match(/\d+/g)[0];
144                 let clefId = "Clef_" + parts[j] + "_" + measureId
                        + "_" + eventsCounters[j];
145                 spineArray[lineCounter].push([clefId, undefined,
                        undefined]);
146                 this.clefs[clefId] = new KernClef(clefshape,
                        clefstep, staves[parts[j]], parts[j],
                        measureId);
147               }
148               // key signature
149               else if (tokens[j].match(/^\*k\[.*\]$/)) {
150                 let key = tokens[j].slice(3, -1);
151                 let keyId = "KeySignature_" + parts[j] + "_" +
                        measureId + "_" + eventsCounters[j];
```

```
152                 let accidentals = key.match(/[a-g]#+|[a-g]-+|[a-g
                        ]n/g);
153                 spineArray[lineCounter].push([keyId, undefined,
                        undefined]);
154                 this.keys[keyId] = new KernKeySignature(undefined
                        , accidentals ? accidentals : [], staves[
                        parts[j]], parts[j], measureId);
155               }
156               // time signature
157               else if (tokens[j].match(/^\*M\d+\/\d+/)) {
158                 let time = tokens[j].slice(2).split("/");
159                 let timeId = "TimeSignature_" + parts[j] + "_" +
                        measureId + "_" + eventsCounters[j];
160                 spineArray[lineCounter].push([timeId, undefined,
                        undefined]);
161                 if (this.times[timeId] === undefined) {
162                   this.times[timeId] = [];
163                 }
164                 this.times[timeId].push(new KernTimeSignature(
                        time[0], time[1], undefined, staves[parts[j
                        ]], parts[j], measureId));
165               }
166               // spine paths
167               else if (tokens[j] === "*-") { // terminate spine
168                 operations[j] = "d";
169               }
170               else if (tokens[j] === "*v") { // join two or more
                      spines
171                 operations[j] = "v";
172                 let k = j + 1;
173                 while (k < tokens.length) { // find all
                        consecutuve spines to be joined and change
                        the column index
174                   if (tokens[k] === "*v") {
175                     operations[k] = "d";
176                     k++;
177                   }
178                   else {
179                     k--;
180                     break;
181                   }
182                 }
183                 j = k;
184                 continue;
185               }
186               else if (tokens[j] === "*+") { // new spine
187                 operations[j] = "a";
188               } // add spine
189               else if (tokens[j] === "*^") { // split spine
```

```
190                 operations[j] = "s";
191                 let voiceid = parts[j] + "_" + (voices[j] + 1);
192                 if (this.voices[voiceid] === undefined) {
193                   this.voices[voiceid] = new KernVoice(staves[
                          parts[j]], parts[j]);
194                 }
195               }
196               else if (tokens[j] === "*x") { // exchange two
                      spines
197                 if (x === -1) {       // first spine
198                   x = j;
199                 }
200                 else {            // second spine
201                   let temp = parts[x];
202                   parts[x] = parts[j];
203                   parts[j] = temp;
204                   temp = voices[x];
205                   voices[x] = voices[j];
206                   voices[j] = temp;
207                   temp = operations[x];
208                   operations[x] = operations[j];
209                   operations[j] = temp;
210                   x = -1;
211                 }
212               }
213             }
214           }
215           else if (parts[j]) {
216             // measures
217             if (tokens[j].match(/^=\w+/)) {
218               eventsCounters[j] = 0;
219               measureId = tokens[j].match(/\w+/)[0];
220               if (this.measures[measureId] === undefined) {
221                 this.measures[measureId] = [];
222               }
223               if (this.measures[measureId].indexOf(parts[j]) ===
                      -1) {
224                 this.measures[measureId].push(parts[j]);
225               }
226             }
227             else if (tokens[j][0] !== "=") { // data token
228               eventsCounters[j]++;
229               let stops = tokens[j].split(/\s+/);
230               for (let k = 0; k < stops.length; k++) {
231                 let stop = stops[k];
232                 if (stop === ".") {
233                   continue;
234                 }
235                 if (stop.match(/\d+/)) {
```

```
236                 let duration = stop.match(/\d+/)[0];
237                 let dots = stop.match(/\./g);
238                 if (dots !== null) {
239                   durations.add(duration * 2 ** dots.length)
240                 }
241                 else {
242                   durations.add(parseInt(duration));
243                 }
244                 let voiceId = parts[j] + "_" + voices[j];
245                 let eventId = voiceId + "_" + measureId + "_" +
                        eventsCounters[j];
246                 // if (stop.match(/P+|p+/)) { }     // Sviluppi
                        futuri: appoggiatura
247                 if (stop.match(/r+/)) {          // rest
248                   this.rests[eventId] = new KernRest(voiceId,
                          measureId, duration, dots);
249                   spineArray[lineCounter].push([eventId,
                          duration, dots ? dots.length : undefined
                          ]);
250                 }
251                 else if (stop.match(/[A-G]+|[a-g]+/)) { // note
252                   if (this.chords[eventId]) {     // existing
                          chord
253                     this.chords[eventId].pushNoteheads(stop);
254                   }
255                   else {                // new chord
256                     this.chords[eventId] = new KernChord(
                            voiceId, measureId, duration, dots,
                            stop);
257                     spineArray[lineCounter].push([eventId,
                            duration, dots ? dots.length :
                            undefined]);
258                   }
259                 }
260               }
261               else if (parts[j].match(/^Lyric_/)) {    //
                      lyrics
262                 let hyphen = stop.match(/-$/) ? true : false;
263                 this.lyrics[parts[j]].push(new KernSyllable(
                        spineArray[lineCounter][0][0], undefined,
                        hyphen, stop));
264               }
265             }
266           }
267         }
268       }
269       if (spineArray[lineCounter].length > 0) {
270         lineCounter++;
271       }
```

```
272         if (operations.length > 0) {
273           // parse spine paths
274           while (operations.indexOf("d") !== -1) {
275             let d = operations.indexOf("d");
276             parts.splice(d, 1);
277             voices.splice(d, 1);
278             eventsCounters.splice(d, 1);
279             operations.splice(d, 1);
280           }
281           while (operations.indexOf("s") !== -1) {
282             let s = operations.indexOf("s");
283             parts.splice(s + 1, 0, parts[s]);
284             voices.splice(s + 1, 0, voices[s] + 1);
285             eventsCounters.splice(s + 1, 0, eventsCounters[s]);
286             operations.splice(s + 1, 0, undefined);
287             operations[s] = undefined;
288           }
289           while (operations.indexOf("+") !== -1) {
290             let a = operations.indexOf("a");
291             let partid = "P" + partsCounter;
292             this.parts.push(partid);
293             partsCounter++;
294             parts.splice(a + 1, 0, partid);
295             let staffid = "Staff_" + partid;
296             staves[partid] = staffid;
297             this.staves.push(staffid);
298             voices.splice(a + 1, 0, 1);
299             let voiceid = partid + "_" + voices[a + 1];
300             this.voices[voiceid] = new KernVoice(staffid, partid)
                  ;
301             eventsCounters.splice(a + 1, 0, 0);
302             operations.splice(a + 1, 0, undefined);
303             operations[a] = undefined;
304           }
305         }
306       }
307       this.parseAllHash();
308       // this.spine
309       let lcm = lcmArray([...durations]);
310       let spineSize = spineArray.length;
311       let vtuMin = lcm;
312       let vtuCounter = 0;
313       let vtuDelta = 0;
314       let flag = true;
315       let lastEventId = "";
316       for (let i = 0; i < spineSize; i++) {
317         let line = spineArray[i];
318         for (let j = 0; j < line.length; j++) {
319           let eventId = line[j][0];
```

```
320           let duration = line[j][1];
321           let dots = line[j][2];
322           let vtu = 0;
323           if (duration !== undefined) {
324             vtu = dots ? dotEval(dots) * lcm / duration : lcm /
                  duration; // vtu duration
325           }
326           if (vtu < vtuMin) {
327             vtuMin = vtu;
328           }
329           if (lastEventId[0] != "P") {  // events following clefs
                , keys, time signatures
330             flag = false;
331           }
332           if (flag) {
333             this.spine[eventId] = new KernEvent(vtuCounter,
                  vtuDelta);
334             flag = false;
335           }
336           else {
337             this.spine[eventId] = new KernEvent(vtuCounter, 0);
338           }
339           this.spineIds.push(eventId);
340           lastEventId = eventId;
341         }
342         vtuCounter += vtuMin;
343         if (vtuMin !== 0) {
344           vtuDelta = vtuMin;
345         }
346         vtuMin = lcm;
347         flag = true;
348       }
349       for (let i = 0; i < this.spineIds.length; i++) {
350         this.spineHash[this.spineIds[i]] = i;
351       }
352       this.stats = new TXTStatistics(this);
353     }
354     catch (error) {
355       console.log(error);
356     }
357   }
```

A.2 IEEE 1599

Listing A.2: codice JS per parsing file IEEE 1599

```
parseMusicDocument() {
  try {
    this.mainTitle = getChildTextContent(this.xmlDoc, "main_title");
    this.number = getChildTextContent(this.xmlDoc, "number");
    this.workTitle = getChildTextContent(this.xmlDoc, "work_title");
    this.workNumber = getChildTextContent(this.xmlDoc, "work_number");
    this.authors = [];
    let auth = this.xmlDoc.getElementsByTagName("author");
    for (let i = 0; i < auth.length; i++) {
      this.authors.push(new XMLAuthor(auth[i], "type"));
    }
    this.relatedFiles = [];
    let relFiles = this.xmlDoc.getElementsByTagName("related_file");
    for (let i = 0; i < relFiles.length; i++) {
      this.relatedFiles.push(new IEEE1599RelatedFile(relFiles[i]));
    }
    let events = this.xmlDoc.getElementsByTagName("spine")[0].children;
    this.spineIds = attributeArray(events, "id");
    this.spineHash = makeHash(events, "id", function (i) { return i; }, true);
    let spine = {};
    let abstime = 0;
    for (let i = 0; i < events.length; i++) {
      let eventid = events[i].getAttribute("id");
      let reltime = events[i].getAttribute("timing");
      abstime += parseInt(reltime);
      spine[eventid] = new XMLEvent(abstime, reltime);
    }
    this.spine = spine;
    this.staves = [];
    let staffItems = this.xmlDoc.getElementsByTagName("staff");
    this.staves = attributeArray(staffItems, "id");
    this.parts = [];
    let partItems = this.xmlDoc.getElementsByTagName("part");
    this.parts = attributeArray(partItems, "id");
    this.measures = {};
    this.clefs = {};
    let clefs = this.xmlDoc.getElementsByTagName("clef");
    for (let i = 0; i < clefs.length; i++) {
```

```
39        let id = clefs[i].getAttribute("event_ref");
40        let staffid = clefs[i].parentNode.getAttribute("id");
41        let clefshape = clefs[i].getAttribute("shape");
42        let clefstep = clefs[i].getAttribute("staff_step");
43        this.clefs[id] = new XMLClef(clefshape, clefstep, staffid
              , undefined, undefined);
44      }
45      this.keys = {};
46      let keysign = this.xmlDoc.getElementsByTagName("
            key_signature");
47      if (keysign) {
48        for (let i = 0; i < keysign.length; i++) {
49          let id = keysign[i].getAttribute("event_ref");
50          let staffid = keysign[i].parentNode.getAttribute("id");
51          let sharps = keysign[i].getElementsByTagName("sharp_num
                ")[0]
52          let flats = keysign[i].getElementsByTagName("flat_num")
                [0];
53          let fifths = 0;
54          fifths += sharps ? parseInt(sharps.getAttribute("number
                ")) : (flats ? -parseInt(flats.getAttribute("number
                ")) : 0);
55          this.keys[id] = new XMLKeySignature(fifths, undefined,
                staffid, undefined, undefined);
56        }
57      }
58      let customkeys = this.xmlDoc.getElementsByTagName("
            custom_key_signature");
59      if (customkeys) {
60        for (let i = 0; i < customkeys.length; i++) {
61          let id = customkeys[i].getAttribute("event_ref");
62          let staffid = customkeys[i].parentNode.getAttribute("id
                ");
63          let accidentals = [];
64          let keyaccs = customkeys[i].getElementsByTagName("
                key_accidental")
65          for (let j = 0; j < keyaccs; j++) {
66            accidentals.push(keyaccs[j].getAttribute("step"));
67          }
68          this.keys[id] = new XMLKeySignature(undefined,
                accidentals, staffid, undefined, undefined);
69        }
70      }
71      this.times = {};
72      let timesigns = this.xmlDoc.getElementsByTagName("
            time_signature");
73      for (let i = 0; i < timesigns.length; i++) {
74        let id = timesigns[i].getAttribute("event_ref");
75        let staffid = timesigns[i].parentNode.getAttribute("id");
```

```
76        let timeIndications = timesigns[i].getElementsByTagName("
              time_indication")
77        this.times[id] = [];
78        for (let j = 0; j < timeIndications.length; j++) {
79          let num = timeIndications[j].getAttribute("num");
80          let den = timeIndications[j].getAttribute("den");
81          let vtu = timeIndications[j].getAttribute("vtu_amount")
                ;
82          this.times[id].push(new XMLTimeSignature(num, den, vtu,
                staffid, undefined, undefined));
83        }
84      }
85      this.voices = {};
86      for (let i = 0; i < partItems.length; i++) {
87        let partid = partItems[i].getAttribute("id");
88        let voiceItems = this.xmlDoc.getElementsByTagName("
              voice_item");
89        for (let j = 0; j < voiceItems.length; j++) {
90          let id = voiceItems[i].getAttribute("id");
91          let staffref = voiceItems[i].getAttribute("staff_ref");
92          this.voices[id] = new XMLVoice(staffref, partid);
93        }
94        let measures = partItems[i].getElementsByTagName("measure
              ");
95        for (let j = 0; j < measures.length; j++) {
96          let measurenum = measures[j].getAttribute("number");
97          if (!this.measures[measurenum]) {
98            this.measures[measurenum] = [];
99          }
100         this.measures[measurenum].push(partid);
101       }
102     }
103     let chordElements = this.xmlDoc.getElementsByTagName("chord
            ");
104     this.chords = makeHash(chordElements, "event_ref", function
            (i) { return new IEEE1599Chord(chordElements[i]); },
            true);;
105     let restElements = this.xmlDoc.getElementsByTagName("rest")
            ;
106     this.rests = makeHash(restElements, "event_ref", function (
            i) { return new IEEE1599Rest(restElements[i]); }, true)
            ;
107     this.parseAllHash();
108     let lyrics = this.xmlDoc.getElementsByTagName("lyrics");
109     this.lyrics = {};
110     this.lyrics = makeHash(lyrics, "voice_ref", function () {
            return []; }, true);
111     for (let i = 0; i < lyrics.length; i++) {
```

```
112         let syllables = lyrics[i].getElementsByTagName("syllable"
              );
113         for (let j = 0; j < syllables.length; j++) {
114           this.lyrics[lyrics[i].getAttribute("voice_ref")][j] =
                new IEEE1599Syllable(syllables[j]);
115         }
116       }
117       let graphicInstances = this.xmlDoc.getElementsByTagName("
            graphic_instance_group");
118       this.graphicInstances = [];
119       this.graphicInstances = makeHash(graphicInstances, "
            description", function () { return []; }, true);
120       for (let i = 0; i < graphicInstances.length; i++) {
121         let pages = graphicInstances[i].getElementsByTagName("
              graphic_instance");
122         let groupName = graphicInstances[i].getAttribute("
              description");
123         for (let j = 0; j < pages.length; j++) {
124           this.graphicInstances[groupName][j] = new
                IEEE1599GraphicInstance(pages[j]);
125           let doc = this;
126           this.graphicInstances[groupName][j].firstEventInPage =
                function () {
127             let curPageEvents = Object.keys(this.graphicEvents);
128             let minIndex = doc.spineIds.length;
129             for (let key in curPageEvents) {
130               var curIndex = doc.spineHash[curPageEvents[key]];
131               if (curIndex < minIndex) {
132                 minIndex = curIndex;
133               }
134             }
135             return doc.spineIds[minIndex];
136           };
137         }
138       }
139       var tracks = this.xmlDoc.getElementsByTagName("track");
140       this.tracks = {}
141       for (var i = 0; i < tracks.length; i++) {
142         var key = makePath(tracks[i].getAttribute("file_name"));
143         this.tracks[key] = new IEEE1599Track(tracks[i]);
144       }
145       this.stats = new XMLStatistics(this);
146     }
147     catch (error) {
148       console.log(error);
149     }
150   }
```

A.3 MusicXML

Listing A.3: codice JS per parsing file MusicXML

```
parseMusicDocument() {
  try {
    this.mainTitle = getChildTextContent(this.xmlDoc, "
       movement-title");
    this.number = getChildTextContent(this.xmlDoc, "
       movement-number");
    this.workTitle = getChildTextContent(this.xmlDoc, "
       work-title");
    this.workNumber = getChildTextContent(this.xmlDoc, "
       work-number");
    this.authors = [];
    let authors = this.xmlDoc.getElementsByTagName("creator");
    for (let i = 0; i < authors.length; i++) {
      this.authors.push(new XMLAuthor(authors[i]));
    }
    this.relatedFiles = [];
    this.spine = {};
    this.spineIds = [];
    this.spineHash = {};
    this.staves = [];
    let parts = this.xmlDoc.getElementsByTagName("part");
    this.parts = attributeArray(parts, "id");
    this.measures = {};
    this.clefs = {};
    this.keys = {};
    this.times = {};
    this.voices = {};
    this.chords = {};
    this.rests = {};
    this.allHash = {};
    this.stats = {};
    this.lyrics = {};
    this.graphicInstances = {};
    this.tracks = {};
    let notesMap = {};
    /*
      name   duration     num den
      maxima 8          32  4
      long   4          16 4
      breve   2          8 4
      whole   1          4 4
      half   0,5         2 4
      quarter 0,25         1 4
      eighth 0,125         1 8
      16th   0,0625      1   16
```

```
42          32nd  0,03125     1 32
43          64th  0,015625    1 64
44          128th  0,0078125    1 128
45          256th  0,00390625   1 256
46          512th  0,001953125   1  512
47          1024th 0.0009765625  1  1024
48        */
49        for (let i = 8; i >= (1 / 1024); i /= 2) {
50          if (i >= 1)
51            notesMap[i] = [4 * i, 4];
52          if (i === 0.5)
53            notesMap[i] = [2, 4];
54          if (i < 0.5)
55            notesMap[i] = [1, 1 / i];
56        }
57        let vtu = 8;
58        for (let i = 0; i < parts.length; i++) {
59          let measures = parts[i].getElementsByTagName("measure");
60          let divisions;
61          for (let j = 0; j < measures.length; j++) {
62            if (getChildIntTextContent(measures[j], "divisions") !
                == undefined) {
63              divisions = getChildIntTextContent(measures[j], "
                   divisions");
64            }
65            let durations = measures[j].getElementsByTagName("
                duration");
66            let duration;
67            for (let k = 0; k < durations.length; k++) {
68              if (durations[k].textContent !== undefined) {
69                duration = parseInt(durations[k].textContent);
70              }
71              else {
72                duration = parseInt(durations[k].text);
73              }
74              let ratio = duration / divisions;
75              vtu = ratio < vtu ? ratio : vtu;
76            }
77          }
78        }
79        for (let i = 8; i >= (1 / 1024); i /= 2) {
80          // cerco la prima nota che dura meno della durata minima
81          if (i < (vtu)) {
82            vtu = i;
83            break;
84          }
85        }
86        let maxTuplet = 0;
```

```
87      let timeModifications = this.xmlDoc.getElementsByTagName("
            time-modification");
88      for (let i = 0; i < timeModifications.length; i++) {
89        let timeModification = timeModifications[i];
90        let actualNote = getChildIntTextContent(timeModification,
              "actual-notes");
91        let normalNote = getChildIntTextContent(timeModification,
              "normal-notes");
92        if (actualNote !== undefined && normalNote !== undefined)
              {
93          maxTuplet = normalNote / actualNote > maxTuplet ?
                normalNote / actualNote : maxTuplet;
94        }
95      }
96      if (maxTuplet === 0) {
97        maxTuplet = 1;
98      }
99      vtu = 1 / (maxTuplet * vtu);
100     let spine = {};
101     for (let i = 0; i < parts.length; i++) {
102       // prendo le misure in ogni parte
103       let partId = parts[i].getAttribute("id");
104       let counter = 0, prevCounter = 0;
105       let measures = parts[i].getElementsByTagName("measure");
106       let whole, beats, beatType;
107       let divisions;
108       for (let j = 0; j < measures.length; j++) {
109         // se c'è una nuova definizione di division aggiorno il
                valore delle note da 1/4 e da 4/4
110         let div = getChildIntTextContent(measures[j], "
              divisions")
111         if (div !== undefined) {
112           divisions = div;
113           whole = 4 * divisions;
114         }
115         let measureNumber = measures[j].getAttribute("number");
116         if (!this.measures[measureNumber]) {
117           this.measures[measureNumber] = [];
118         }
119         this.measures[measureNumber].push(partId);
120         let clefs = measures[j].getElementsByTagName("clef");
121         for (let k = 0; k < clefs.length; k++) {
122           let number = clefs[k].getAttribute("number");
123           if (!number) {
124             number = 1;
125           }
126           let sign = getChildTextContent(clefs[k], "sign");
127           let line = getChildIntTextContent(clefs[k], "line");
128           if (!line) {
```

```
129             line = 0;
130           }
131           let clefId = "Clef_" + partId + "_" + number + "_" +
                  formatNumberLength(measureNumber,
                  measures.length.toString().length);
132           let staffId = "Staff_" + partId + "_" + number;
133           this.clefs[clefId] = new XMLClef(sign, line, staffId,
                   partId, measureNumber);
134           spine[clefId] = counter;
135           // sviluppi futuri : gestire più chiavi per la stessa
                   voce all'interno della stessa misura
136         }
137         let keys = measures[j].getElementsByTagName("key");
138         if (keys.length > 0) {
139           let fifths = getChildIntTextContent(keys[0], "fifths"
                  );
140           let keyId = "KeySignature_" + partId + "_" +
                  formatNumberLength(measureNumber,
                  measures.length.toString().length)
141           this.keys[keyId] = new XMLKeySignature(fifths,
                  undefined, undefined, partId, measureNumber);
142           spine[keyId] = counter;
143         }
144         let times = measures[j].getElementsByTagName("time");
145         if (times.length > 0) {
146           // aggiorno beats e beattype se vengono ridefiniti
                  nella misura
147           let time = times[0];
148           let b = getChildIntTextContent(time, "beats");
149           if (b !== undefined) {
150             beats = b;
151           }
152           let bt = getChildIntTextContent(time, "beat-type");
153           if (bt !== undefined) {
154             beatType = bt;
155           }
156           let timeId = "TimeSignature_" + partId + "_" +
                  formatNumberLength(measureNumber,
                  measures.length.toString().length);
157           if (this.times[timeId] === undefined) {
158             this.times[timeId] = [];
159           }
160           this.times[timeId].push(new XMLTimeSignature(beats,
                  beatType, vtu * whole * beats / beatType,
                  undefined, partId, measureNumber));
161           spine[timeId] = counter;
162         }
163         let measureChildren = measures[j].children;
```

```
164         let eventCounters = {} //contatore eventi per le varie
                voci
165         let duration;
166         for (let k = 0; k < measureChildren.length; k++) {
167           if (measureChildren[k].nodeName === "note") {
168             let voiceId, eventId;
169             let num, den;
170             let note = measureChildren[k];
171             let grace = note.getElementsByTagName("grace")[0];
172             if (grace !== undefined) {
173               // Sviluppi futuri: gestione grace notes (
                      appoggiatura etc.)
174               continue;
175             }
176             let voice = getChildTextContent(note, "voice");
177             if (voice === "") {
178               voice = "v1";
179             }
180             voiceId = partId + '_' + voice;
181             let staff = getChildTextContent(note, "staff");
182             if (staff === "") {
183               staff = "1";
184             }
185             let staffId = "Staff_" + partId + "_" + staff;
186             if (this.staves.indexOf(staffId) === -1) {
187               this.staves.push(staffId);
188             }
189             if (this.voices[voiceId] === undefined) {
190               this.voices[voiceId] = new XMLVoice(staffId,
                      partId);
191             }
192             if (eventCounters[voice] === undefined) {
193               eventCounters[voice] = 0;
194             }
195             let chord = note.getElementsByTagName("chord")[0];
196             if (chord === undefined) {
197               eventCounters[voice]++;
198             }
199             eventId = voiceId + "_" + formatNumberLength(
                    measureNumber, measures.length.toString().
                    length) + "_" + eventCounters[voice];
200             duration = getChildIntTextContent(note, "duration")
                    ;
201             let durationWeighted = duration === undefined ? 0 :
                     duration * vtu / divisions;
202             let timeModifications = note.getElementsByTagName("
                    time-modification");
203             if (timeModifications.length > 0) {
```

```
204                 // c'è una tupla e devo correggere la durata =
                        durata * actualnote / normalnote
205                 let timeModification = timeModifications[0];
206                 let actualNote = getChildIntTextContent(
                        timeModification, "actual-notes");
207                 if (actualNote !== undefined) {
208                   duration *= actualNote;
209                 }
210                 let normalNote = getChildIntTextContent(
                        timeModification, "normal-notes");
211                 if (normalNote !== undefined) {
212                   duration /= normalNote;
213                 }
214               }
215               // controllo i punti e correggo la durata
216               duration /= dotEval(note.getElementsByTagName("dot"
                      ).length);
217               for (let i = 8; i >= (1 / 1024); i /= 2) {
218                 // cerco la prima nota che dura meno della durata
219                 if ((duration >= (whole * notesMap[i][0] /
                        notesMap[i][1]))) {
220                   num = notesMap[i][0];
221                   den = notesMap[i][1];
222                   break;
223                 }
224               }
225               if (chord === undefined) {
226                 spine[eventId] = counter;
227                 prevCounter = counter;
228                 counter += durationWeighted;
229                 // la nota non fa parte di un accordo
                        preesistente: ne creo uno nuovo
230                 let rests = note.getElementsByTagName("rest");
231                 if (rests.length > 0) {
232                   // creare una pausa
233                   if (rests[0].getAttribute("measure") === "yes")
                          {
234                     // measure long rest
235                     num = beats;
236                     den = beatType;
237                   }
238                   let rest = new MusicXMLRest(note, voiceId,
                          measureNumber, num, den);
239                   this.rests[eventId] = rest;
240                 }
241                 else {
242                   // creare un accordo
243                   this.chords[eventId] = new MusicXMLChord(note,
                          voiceId, measureNumber, num, den);
```

```
244                 }
245               }
246               else {
247                 //aggiungi nota ad accordo precedente
248                 this.chords[eventId].pushNoteheads(note);
249                 spine[eventId] = prevCounter;
250               }
251               //lyrics
252               let lyrics = note.getElementsByTagName("lyric");
253               for (let l = 0; l < lyrics.length; l++) {
254                 let lyricNumber = lyrics[l].getAttribute("number"
                      );
255                 let text = getChildTextContent(lyrics[l], "text")
256                 let syll = getChildTextContent(lyrics[l], "
                      syllabic");
257                 let hyphen = (syll === "begin" || syll === "
                      middle");
258                 let lyricId = "Lyric_" + lyricNumber;
259                 if (this.lyrics[lyricId] === undefined) {
260                   this.lyrics[lyricId] = [];
261                 }
262                 this.lyrics[lyricId].push(new MusicXMLSyllable(
                      eventId, undefined, hyphen, text));
263               }
264             }
265             else if (measureChildren[k].nodeName === "backup") {
266               let backupDuration = getChildIntTextContent(
                      measureChildren[k], "duration");
267               let durationWeighted = backupDuration === undefined
                      ? 0 : backupDuration * vtu / divisions;
268               prevCounter = counter;
269               counter -= durationWeighted;
270             }
271             else if (measureChildren[k].nodeName === "forward") {
272               let forwardDuration = getChildIntTextContent(
                      measureChildren[k], "duration");
273               let durationWeighted = forwardDuration ===
                      undefined ? 0 : forwardDuration * vtu /
                      divisions;
274               prevCounter = counter;
275               counter += durationWeighted;
276             }
277           }
278         }
279       }
280       let SpineKeysSorted = Object.keys(spine).sort(function (a,
            b) { return spine[a] - spine[b] })
281       let previous = 0;
282       for (let key of SpineKeysSorted) {
```

```
283         let abstiming = spine[key];
284         let reltiming = abstiming - previous;
285         this.spine[key] = new XMLEvent(abstiming, reltiming);
286         this.spineIds.push(key);
287         previous = abstiming;
288       }
289       for (let i = 0; i < this.spineIds.length; i++) {
290         this.spineHash[this.spineIds[i]] = i;
291       }
292       this.parseAllHash();
293       this.stats = new XMLStatistics(this);
294     }
295     catch (error) {
296       console.log(error);
297     }
298 }
```

Appendice B

Calcolo statistiche

Questa appendice contiene la rivisitazione del codice di calcolo statistiche dell'*ieee1599-framework*, adattato per l'utilizzo delle nuove strutture dati: il metodo si occupa di valorizzare diversi contatori (altezza note, durata, durata pesata etc.) sulla base delle proprietà delle note codificate nell'oggetto passato per parametro.

Listing B.1: codice JS per calcolo statistiche

```
1  parseStatistics(MusicDocument) {
2    try {
3      // rivisitazione del codice di generazione statistiche
           framework ieee1599
4      this.numParts = Object.keys(MusicDocument.parts).length;
5      this.numEvents = Object.keys(MusicDocument.spineIds).
           length;
6      this.numChords = Object.keys(MusicDocument.chords).length;
7      this.numRests = Object.keys(MusicDocument.rests).length;
8      this.numMeasures = Object.keys(MusicDocument.measures).
           length;
9      this.numNotes = 0;
10     for (let key in MusicDocument.chords) {
11       this.numNotes += MusicDocument.chords[key].
             noteheads.length;
12     }
13     this.pitchClass = [];
14     this.pitchClassWeighted = [];
15     for (let i = 0; i < 12; i++) {
16       this.pitchClass[i] = 0;
17       this.pitchClassWeighted[i] = 0;
18     }
19     this.midiPitch = [];
20     this.midiPitchWeighted = [];
21     for (let i = 0; i < 128; i++) {
22       this.midiPitch[i] = 0;
23       this.midiPitchWeighted[i] = 0;
```

```
24      }
25      this.chordLength = {};
26      this.noteLength = {};
27      this.restLength = {};
28      this.noteByLengthPC = {};
29      for (let i in MusicDocument.rests) {
30        let restnum = MusicDocument.rests[i].durationNum;
31        if (typeof restnum === undefined || restnum === null) {
32          break;
33        }
34        let restden = MusicDocument.rests[i].durationDen;
35        if (typeof restden === undefined || restden === null) {
36          restden = 1;
37        }
38        if (this.restLength[restnum / restden] == undefined) {
39          this.restLength[restnum / restden] = 1;
40        }
41        else {
42          this.restLength[restnum / restden]++;
43        }
44      }
45      for (let i in MusicDocument.chords) {
46        let chordnum = MusicDocument.chords[i].durationNum;
47        if (typeof chordnum === undefined || chordnum === null) {
48          break;
49        }
50        let chordden = MusicDocument.chords[i].durationDen;
51        if (typeof chordden === undefined || chordden === null) {
52          chordden = 1;
53        }
54        if (this.chordLength[4 * chordnum / chordden] ==
             undefined) {
55          this.chordLength[4 * chordnum / chordden] = 1;
56        }
57        else {
58          this.chordLength[4 * chordnum / chordden]++;
59        }
60        if (this.noteByLengthPC[4 * chordnum / chordden] ==
             undefined) {
61          this.noteByLengthPC[4 * chordnum / chordden] = {};
62          for (let k = 0; k < 12; k++) {
63            this.noteByLengthPC[4 * chordnum / chordden][k] = 0;
64          }
65        }
66        // Sviluppi futuri: gestire tuplets e aug_dots (richiede
             modifiche al visualizzatore di statistiche del player
              multimediale)
67        for (let j in MusicDocument.chords[i].noteheads) {
68          var notehead = MusicDocument.chords[i].noteheads[j];
```

```
69          if (this.noteLength[4 * chordnum / chordden] ==
                undefined) {
70            this.noteLength[4 * chordnum / chordden] = 1;
71          }
72          else {
73            this.noteLength[4 * chordnum / chordden]++;
74          }
75          if ((notehead.step == "C" && (notehead.actualAccidental
                 == "none" || notehead.actualAccidental == "natural
                ")) ||
76            (notehead.step == "B" && notehead.actualAccidental ==
                 "sharp") ||
77            (notehead.step == "D" && notehead.actualAccidental ==
                 "double_flat")) {
78            this.pitchClass[0]++;
79            this.midiPitch[12 * notehead.octave]++;
80            this.pitchClassWeighted[0] += 4 * chordnum /
                chordden;
81            this.midiPitchWeighted[12 * notehead.octave] += 4 *
                chordnum / chordden;
82            this.noteByLengthPC[4 * chordnum / chordden][0]++;
83          }
84          else if ((notehead.step == "C" &&
                notehead.actualAccidental == "sharp") ||
85            (notehead.step == "B" && notehead.actualAccidental ==
                 "double_sharp") ||
86            (notehead.step == "D" && notehead.actualAccidental ==
                 "flat")) {
87            this.pitchClass[1]++;
88            this.midiPitch[1 + 12 * notehead.octave]++;
89            this.pitchClassWeighted[1] += 4 * chordnum /
                chordden;
90            this.midiPitchWeighted[1 + 12 * notehead.octave] += 4
                 * chordnum / chordden;
91            this.noteByLengthPC[4 * chordnum / chordden][1]++;
92          }
93          else if ((notehead.step == "D" && (
                notehead.actualAccidental == "none" ||
                notehead.actualAccidental == "natural")) ||
94            (notehead.step == "C" && notehead.actualAccidental ==
                 "double_sharp") ||
95            (notehead.step == "E" && notehead.actualAccidental ==
                 "double_flat")) {
96            this.pitchClass[2]++;
97            this.midiPitch[2 + 12 * notehead.octave]++;
98            this.pitchClassWeighted[2] += 4 * chordnum /
                chordden;
99            this.midiPitchWeighted[2 + 12 * notehead.octave] += 4
                 * chordnum / chordden;
```

```
100             this.noteByLengthPC[4 * chordnum / chordden][2]++;
101           }
102           else if ((notehead.step == "D" &&
                  notehead.actualAccidental == "sharp") ||
103             (notehead.step == "E" && notehead.actualAccidental ==
                    "flat") ||
104             (notehead.step == "F" && notehead.actualAccidental ==
                    "double_flat")) {
105             this.pitchClass[3]++;
106             this.midiPitch[3 + 12 * notehead.octave]++;
107             this.pitchClassWeighted[3] += 4 * chordnum /
                    chordden;
108             this.midiPitchWeighted[3 + 12 * notehead.octave] += 4
                    * chordnum / chordden;
109             this.noteByLengthPC[4 * chordnum / chordden][3]++;
110           }
111           else if ((notehead.step == "E" && (
                  notehead.actualAccidental == "none" ||
                  notehead.actualAccidental == "natural")) ||
112             (notehead.step == "D" && notehead.actualAccidental ==
                    "double_sharp") ||
113             (notehead.step == "F" && notehead.actualAccidental ==
                    "flat")) {
114             this.pitchClass[4]++;
115             this.midiPitch[4 + 12 * notehead.octave]++;
116             this.pitchClassWeighted[4] += 4 * chordnum /
                    chordden;
117             this.midiPitchWeighted[4 + 12 * notehead.octave] += 4
                    * chordnum / chordden;
118             this.noteByLengthPC[4 * chordnum / chordden][4]++;
119           }
120           else if ((notehead.step == "F" && (
                  notehead.actualAccidental == "none" ||
                  notehead.actualAccidental == "natural")) ||
121             (notehead.step == "E" && notehead.actualAccidental ==
                    "sharp") ||
122             (notehead.step == "G" && notehead.actualAccidental ==
                    "double_flat")) {
123             this.pitchClass[5]++;
124             this.midiPitch[5 + 12 * notehead.octave]++;
125             this.pitchClassWeighted[5] += 4 * chordnum /
                    chordden;
126             this.midiPitchWeighted[5 + 12 * notehead.octave] += 4
                    * chordnum / chordden;
127             this.noteByLengthPC[4 * chordnum / chordden][5]++;
128           }
129           else if ((notehead.step == "F" &&
                  notehead.actualAccidental == "sharp") ||
```

```
130             (notehead.step == "G" && notehead.actualAccidental ==
                    "flat") ||
131             (notehead.step == "E" && notehead.actualAccidental ==
                    "double_sharp")) {
132             this.pitchClass[6]++;
133             this.midiPitch[6 + 12 * notehead.octave]++;
134             this.pitchClassWeighted[6] += 4 * chordnum /
                   chordden;
135             this.midiPitchWeighted[6 + 12 * notehead.octave] += 4
                    * chordnum / chordden;
136             this.noteByLengthPC[4 * chordnum / chordden][6]++;
137           }
138           else if ((notehead.step == "G" && (
                   notehead.actualAccidental == "none" ||
                   notehead.actualAccidental == "natural")) ||
139             (notehead.step == "F" && notehead.actualAccidental ==
                    "double_sharp") ||
140             (notehead.step == "A" && notehead.actualAccidental ==
                    "double_flat")) {
141             this.pitchClass[7]++;
142             this.midiPitch[7 + 12 * notehead.octave]++;
143             this.pitchClassWeighted[7] += 4 * chordnum /
                   chordden;
144             this.midiPitchWeighted[7 + 12 * notehead.octave] += 4
                    * chordnum / chordden;
145             this.noteByLengthPC[4 * chordnum / chordden][7]++;
146           }
147           else if ((notehead.step == "G" &&
                   notehead.actualAccidental == "sharp") ||
148             (notehead.step == "A" && notehead.actualAccidental ==
                    "flat")) {
149             this.pitchClass[8]++;
150             this.midiPitch[8 + 12 * notehead.octave]++;
151             this.pitchClassWeighted[8] += 4 * chordnum /
                   chordden;
152             this.midiPitchWeighted[8 + 12 * notehead.octave] += 4
                    * chordnum / chordden;
153             this.noteByLengthPC[4 * chordnum / chordden][8]++;
154           }
155           else if ((notehead.step == "A" && (
                   notehead.actualAccidental == "none" ||
                   notehead.actualAccidental == "natural")) ||
156             (notehead.step == "G" && notehead.actualAccidental ==
                    "double_sharp") ||
157             (notehead.step == "B" && notehead.actualAccidental ==
                    "double_flat")) {
158             this.pitchClass[9]++;
159             this.midiPitch[9 + 12 * notehead.octave]++;
```

```
160             this.pitchClassWeighted[9] += 4 * chordnum /
                    chordden;
161             this.midiPitchWeighted[9 + 12 * notehead.octave] += 4
                     * chordnum / chordden;
162             this.noteByLengthPC[4 * chordnum / chordden][9]++;
163           }
164           else if ((notehead.step == "A" &&
                  notehead.actualAccidental == "sharp") ||
165             (notehead.step == "B" && notehead.actualAccidental ==
                     "flat") ||
166             (notehead.step == "C" && notehead.actualAccidental ==
                     "double_flat")) {
167             this.pitchClass[10]++;
168             this.midiPitch[10 + 12 * notehead.octave]++;
169             this.pitchClassWeighted[10] += 4 * chordnum /
                    chordden;
170             this.midiPitchWeighted[10 + 12 * notehead.octave] +=
                    4 * chordnum / chordden;
171             this.noteByLengthPC[4 * chordnum / chordden][10]++;
172           }
173           else if ((notehead.step == "B" && (
                  notehead.actualAccidental == "none" ||
                  notehead.actualAccidental == "natural")) ||
174             (notehead.step == "A" && notehead.actualAccidental ==
                     "double_sharp") ||
175             (notehead.step == "C" && notehead.actualAccidental ==
                     "flat")) {
176             this.pitchClass[11]++;
177             this.midiPitch[11 + 12 * notehead.octave]++;
178             this.pitchClassWeighted[11] += 4 * chordnum /
                    chordden;
179             this.midiPitchWeighted[11 + 12 * notehead.octave] +=
                    4 * chordnum / chordden;
180             this.noteByLengthPC[4 * chordnum / chordden][11]++;
181           }
182         }
183       }
184     }
185     catch (error) {
186       console.log(error);
187     }
188   }
```

Appendice C

Pagina HTML di test

Questa appendice contiene il codice sorgente della pagina HTML utilizzata per visualizzare i risultati dell'attività di validazione degli oggetti.

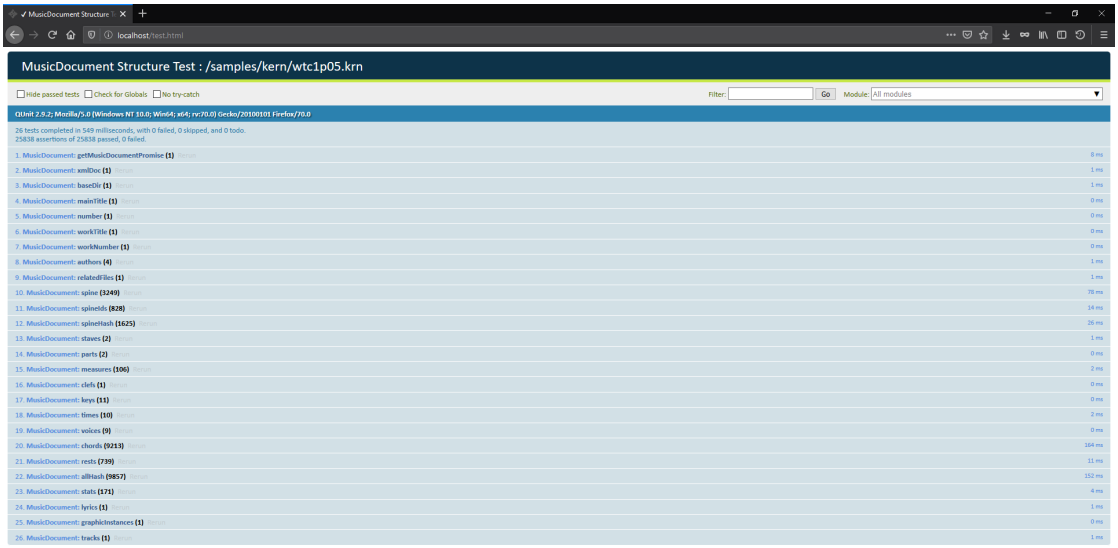


Figura 91: Pagina di test aperta con il browser Firefox

Listing C.1: test.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="utf-8">
6    <meta name="viewport" content="width=device-width">
```

```
7     <title>MusicDocument Structure Test</title>
8     <link rel="stylesheet" href="/qunit/qunit-2.9.2.css">
9   </head>
10
11  <body>
12    <div id="qunit"></div>
13    <div id="qunit-fixture"></div>
14    <script type="text/javascript" src="/js/utils.js"></script>
15    <script type="text/javascript" src="/js/generic/
          musicDocumentPrototype.js"></script>
16    <script type="text/javascript" src="/js/xml/
          xmlMusicDocumentPrototype.js"></script>
17    <script type="text/javascript" src="/js/xml/ieee1599/
          ieee1599Document.js"></script>
18    <script type="text/javascript" src="/js/xml/musicxml/
          musicxmlDocument.js"></script>
19    <script type="text/javascript" src="/js/txt/
          txtMusicDocumentPrototype.js"></script>
20    <script type="text/javascript" src="/js/txt/kern/
          kernDocument.js"></script>
21    <script type="text/javascript" src="/js/
          getMusicDocumentPromise.js"></script>
22    <script type="text/javascript" src="/qunit/qunit-2.9.2.js">
          </script>
23    <script type="text/javascript" src="/test/tests.js"></script>
24  </body>
25
26  </html>
```

Appendice D

Pagina HTML demo

Questa appendice contiene il codice sorgente della pagina HTML che, invocando le funzioni di creazione documenti, istanzia oggetti MusicDocument da un lista di URL e li stampa a console.

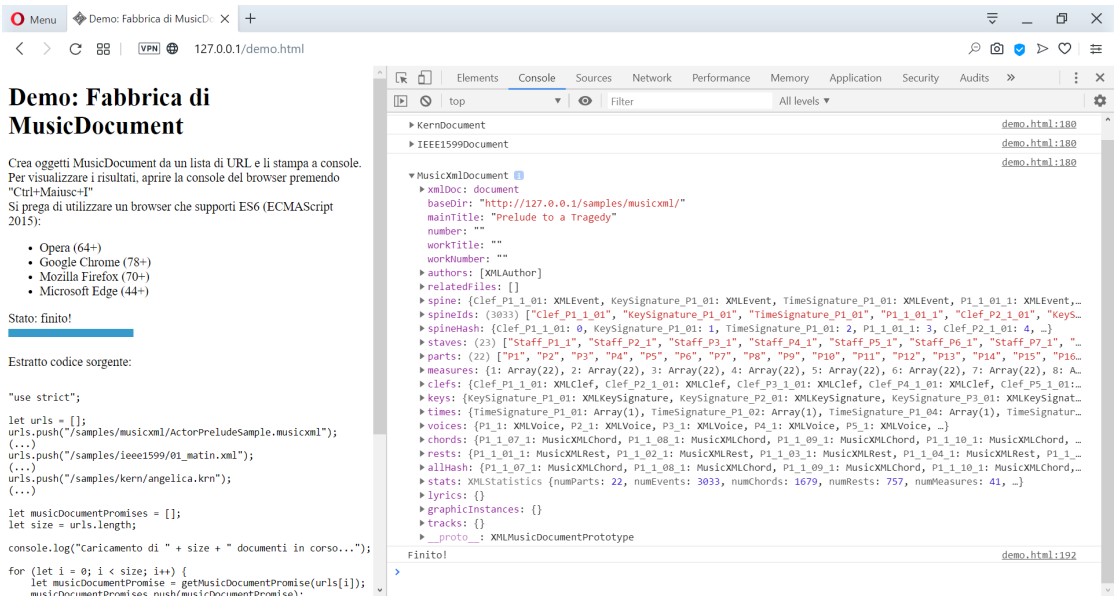


Figura 92: Pagina demo aperta con il browser Opera

Listing D.1: demo.html

```
1  <!DOCTYPE html>
2  <html lang="en">
3
4  <head>
5    <meta charset="UTF-8">
```

```
6     <meta name="viewport" content="width=device-width,
          initial-scale=1.0">
7     <meta http-equiv="X-UA-Compatible" content="ie=edge">
8     <title>Demo: Fabbrica di MusicDocument</title>
9     <script src="/js/utils.js"></script>
10    <script src="/js/generic/musicDocumentPrototype.js"></script>
11    <script src="/js/xml/xmlMusicDocumentPrototype.js"></script>
12    <script src="/js/xml/musicxml/musicxmlDocument.js"></script>
13    <script src="/js/xml/ieee1599/ieee1599Document.js"></script>
14    <script src="/js/txt/txtMusicDocumentPrototype.js"></script>
15    <script src="/js/txt/kern/kernDocument.js"></script>
16    <script src="/js/getMusicDocumentPromise.js"></script>
17  </head>
18
19  <body>
20    <h1>Demo: Fabbrica di MusicDocument</h1>
21    <div id="example">
22      Crea oggetti MusicDocument da un lista di URL e li stampa a
            console.<br />
23      Per visualizzare i risultati, aprire la console del browser
            premendo "Ctrl+Maiusc+I"<br />
24      Si prega di utilizzare un browser che supporti ES6 (
          ECMAScript 2015):
25      <ul style="list-style-type:disc;">
26        <li>Opera (64+)</li>
27        <li>Google Chrome (78+)</li>
28        <li>Mozilla Firefox (70+)</li>
29        <li>Microsoft Edge (44+)</li>
30      </ul>
31    </div>
32    <div id="status">
33      Stato:
34    </div>
35    <progress id="progressBar" value="0" max="100"></progress>
36    <br /><br />
37    <div id="code">
38      Estratto codice sorgente:
39      <pre>
40        <code>
41  "use strict";
42
43  let urls = [];
44  urls.push("/samples/musicxml/ActorPreludeSample.musicxml");
45  (...)
46  urls.push("/samples/ieee1599/01_matin.xml");
47  (...)
48  urls.push("/samples/kern/angelica.krn");
49  (...)
50
```

```
51 let musicDocumentPromises = [];
52 let size = urls.length;
53
54 console.log("Caricamento di " + size + " documenti in corso..."
       );
55
56 for (let i = 0; i < size; i++) {
57   let musicDocumentPromise = getMusicDocumentPromise(urls[i]);
58   musicDocumentPromises.push(musicDocumentPromise);
59   musicDocumentPromise
60     .then((musicDocument) => {
61       console.log(musicDocument);
62     })
63     .catch((err) => {
64       console.log("Errore : " + err)
65     });
66 }
67
68 Promise.all(musicDocumentPromises)
69   .then(() => {
70     console.log("Finito!");
71   })
72   .catch((err) => {
73     console.log("C'è stato un errore : " + err);
74   });
75       </code>
76     </pre>
77   </div>
78   <script>
79     "use strict";
80     let urls = [];
81     urls.push("/samples/musicxml/ActorPreludeSample.musicxml");
82     urls.push("/samples/musicxml/BeetAnGeSample.musicxml");
83     urls.push("/samples/musicxml/Binchois.musicxml");
84     urls.push("/samples/musicxml/BrahWiMeSample.musicxml");
85     urls.push("/samples/musicxml/BrookeWestSample.musicxml");
86     urls.push("/samples/musicxml/Chant.musicxml");
87     urls.push("/samples/musicxml/DebuMandSample.musicxml");
88     urls.push("/samples/musicxml/Dichterliebe01.musicxml");
89     urls.push("/samples/musicxml/Echigo-Jishi.musicxml");
90     urls.push("/samples/musicxml/FaurReveSample.musicxml");
91     urls.push("/samples/musicxml/MahlFaGe4Sample.musicxml");
92     urls.push("/samples/musicxml/MozaChloSample.musicxml");
93     urls.push("/samples/musicxml/MozartPianoSonata.musicxml");
94     urls.push("/samples/musicxml/MozartTrio.musicxml");
95     urls.push("/samples/musicxml/MozaVeilSample.musicxml");
96     urls.push("/samples/musicxml/Saltarello.musicxml");
97     urls.push("/samples/musicxml/SchbAvMaSample.musicxml");
98     urls.push("/samples/musicxml/Telemann.musicxml");
```

```
99      urls.push("/samples/musicxml/timewise/
           ActorPreludeSample.musicxml");
100     urls.push("/samples/musicxml/timewise/
           BeetAnGeSample.musicxml");
101     urls.push("/samples/musicxml/timewise/Binchois.musicxml");
102     urls.push("/samples/musicxml/timewise/
           BrahWiMeSample.musicxml");
103     urls.push("/samples/musicxml/timewise/
           BrookeWestSample.musicxml");
104     urls.push("/samples/musicxml/timewise/Chant.musicxml");
105     urls.push("/samples/musicxml/timewise/
           DebuMandSample.musicxml");
106     urls.push("/samples/musicxml/timewise/
           Dichterliebe01.musicxml");
107     urls.push("/samples/musicxml/timewise/Echigo-Jishi.musicxml
           ");
108     urls.push("/samples/musicxml/timewise/
           FaurReveSample.musicxml");
109     urls.push("/samples/musicxml/timewise/
           MahlFaGe4Sample.musicxml");
110     urls.push("/samples/musicxml/timewise/
           MozaChloSample.musicxml");
111     urls.push("/samples/musicxml/timewise/
           MozartPianoSonata.musicxml");
112     urls.push("/samples/musicxml/timewise/MozartTrio.musicxml")
           ;
113     urls.push("/samples/musicxml/timewise/
           MozaVeilSample.musicxml");
114     urls.push("/samples/musicxml/timewise/Saltarello.musicxml")
           ;
115     urls.push("/samples/musicxml/timewise/
           SchbAvMaSample.musicxml");
116     urls.push("/samples/musicxml/timewise/Telemann.musicxml");
117     urls.push("/samples/ieee1599/01_matin.xml");
118     urls.push("/samples/ieee1599/02_promenade.xml");
119     urls.push("/samples/ieee1599/03_historiette.xml");
120     urls.push("/samples/ieee1599/ave_maria.xml");
121     urls.push("/samples/ieee1599/a_chloris.xml");
122     urls.push("/samples/ieee1599/bist_du_bei_mir.xml");
123     urls.push("/samples/ieee1599/catedral.xml");
124     urls.push("/samples/ieee1599/der_holle_rache.xml");
125     urls.push("/samples/ieee1599/eleanor_rigby.xml");
126     urls.push("/samples/ieee1599/gottes_macht.xml");
127     urls.push("/samples/ieee1599/gymnopedie_01.xml");
128     urls.push("/samples/ieee1599/gymnopedie_02.xml");
129     urls.push("/samples/ieee1599/gymnopedie_03.xml");
130     urls.push("/samples/ieee1599/il_mio_ben_quando_verra.xml");
131     urls.push("/samples/ieee1599/intermezzo_sinfonico.xml");
132     urls.push("/samples/ieee1599/introitus.xml");
```

```
133     urls.push("/samples/ieee1599/inventio01.xml");
134     urls.push("/samples/ieee1599/i_pianti_che_grondano.xml");
135     urls.push("/samples/ieee1599/jesus_bleibet_bwv_147.xml");
136     urls.push("/samples/ieee1599/
          jesus_bleibet_bwv_147_ultima.xml");
137     urls.push("/samples/ieee1599/khomus.xml");
138     urls.push("/samples/ieee1599/
          les_sons_et_les_parfums_tournent.xml");
139     urls.push("/samples/ieee1599/lullaby_of_birdland.xml");
140     urls.push("/samples/ieee1599/pavane.xml");
141     urls.push("/samples/ieee1599/serie_in_9_8.xml");
142     urls.push("/samples/ieee1599/sleeping_beauty_var3.xml");
143     urls.push("/samples/ieee1599/star.xml");
144     urls.push("/samples/ieee1599/traviata.xml");
145     urls.push("/samples/ieee1599/ul_parisien.xml");
146     urls.push("/samples/ieee1599/weiss_prelude.xml");
147     urls.push("/samples/kern/angelica.krn");
148     urls.push("/samples/kern/ballad10-1.krn");
149     urls.push("/samples/kern/chor-021.krn");
150     urls.push("/samples/kern/coppini07.krn");
151     urls.push("/samples/kern/coppini13.krn");
152     urls.push("/samples/kern/de_bonte.krn");
153     urls.push("/samples/kern/etude10-02.krn");
154     urls.push("/samples/kern/han0436.krn");
155     urls.push("/samples/kern/k439b.krn");
156     urls.push("/samples/kern/mazurka41-1.krn");
157     urls.push("/samples/kern/op12-2.krn");
158     urls.push("/samples/kern/pineapple.krn");
159     urls.push("/samples/kern/piston295.krn");
160     urls.push("/samples/kern/plaudite.krn");
161     urls.push("/samples/kern/sonata02-1.krn");
162     urls.push("/samples/kern/sonata02-3.krn");
163     urls.push("/samples/kern/symph5-1.krn");
164     urls.push("/samples/kern/viennese1-1.krn");
165     urls.push("/samples/kern/wtc1f01.krn");
166     urls.push("/samples/kern/wtc1p05.krn");
167     let musicDocumentPromises = [];
168     let size = urls.length;
169     let status = document.getElementById("status");
170     let progressBar = document.getElementById("progressBar");
171     console.log("Caricamento di " + size + " documenti in
          corso...");
172     status.textContent += "caricamento di " + size + "
          documenti";
173     let progress = 0;
174     for (let i = 0; i < size; i++) {
175       let musicDocumentPromise = getMusicDocumentPromise(urls[i
            ]);
176       musicDocumentPromises.push(musicDocumentPromise);
```

```
177         musicDocumentPromise
178           .then((musicDocument) => {
179             console.log(musicDocument);
180           })
181           .catch((err) => {
182             console.log("Errore : " + err)
183           })
184           .finally(() => {
185             progress++;
186             progressBar.value = 100 * progress / size;
187           });
188       }
189       Promise.all(musicDocumentPromises)
190         .then(() => {
191           console.log("Finito!");
192           status.textContent = "Stato: finito!";
193         })
194         .catch((err) => {
195           console.log("C'è stato un errore : " + err);
196           status.textContent = "Stato: c'è stato un errore,
                controlla la console per i dettagli";
197         });
198     </script>
199 </body>
200
201 </html>
```

Bibliografia

[1] Adriano Baratè, Goffredo Haus, and Luca Andrea Ludovico. State of the art and perspectives in multi-layer formats for music representation. In *Proceedings of the 2019 International Workshop on Multilayer Music Representation and Processing (MMRP 2019)*, pages 27–34. IEEE CPS, 2019.

[2] Goffredo Haus. Rescuing la scala. *Computer*, (3):88–89, 1998.

[3] Goffredo Haus and Luca A Ludovico. Digitization of historial music archives: preserving the past, embracing the future. In *TENOR 2017: International Conference on Technologies for Music Notation and Representation:[24-26 May 2017, University of A Coruña, Spain]*, pages 1–8. Facultade de Filoloxía, 2017.

[4] Luca Andrea Ludovico. IEEE 1599: A multi-layer approach to music description. *Journal of Multimedia*, 4(1):9–14, 2009.

[5] Jacques Steyn. Framework for a music markup language. In *Proceeding of the First International IEEE Conference on Musical Application using XML (MAX2002)*, pages 22–29, 2002.

[6] Goffredo Haus and Maurizio Longari. A multi-layered, time-based music description approach based on xml. *Computer Music Journal*, 29(1):70–85, 2005.

[7] Adriano Baratè, Goffredo Haus, and Luca Andrea Ludovico. A critical review of the IEEE 1599 standard. *Computer Standards & Interfaces*, 46:46–51, 2016.

[8] David Huron. Music information processing using the humdrum toolkit: Concepts, examples, and lessons. *Computer Music Journal*, 26(2):11–26, 2002.

[9] Laurent Pugin, Rodolfo Zitellini, and Perry Roland. Verovio: A library for engraving mei music notation into svg. In *ISMIR*, pages 107–112, 2014.

[10] Craig Stuart Sapp. Online database of scores in the humdrum file format. In *ISMIR*, pages 664–665, 2005.

[11] Donald Byrd, Roger D Boyle, Ulf Berggren, David Bainbridge, et al. *Beyond MIDI: the handbook of musical codes*. MIT press, 1997.

[12] Adriano Baratè, Goffredo Haus, Luca Andrea Ludovico, and Davide Andrea Mauro. IEEE 1599 for live musical and theatrical performances. *Journal of Multimedia*, 7(2):170–178, 2012.

[13] Denis L Baggi and Goffredo M Haus. *Music navigation with symbols and layers: Toward content browsing with IEEE 1599 XML encoding*. John Wiley & Sons, 2013.

[14] Adriano Barate, Goffredo Haus, Luca A Ludovico, and Davide Andrea Mauro. Ieee 1599 for live musical and theatrical performances. 2012.

[15] Ann Hutchinson, Ann Hutchinson Guest, and William Ambrose Hutchinson. *Labanotation: Or, Kinetography Laban: The System of Analyzing and Recording Movement*. Number 27. Taylor & Francis, 1977.

Progetto sviluppato presso il Laboratorio di Informatica Musicale
`https://www.lim.di.unimi.it`