Indice

Capitolo 3

Capitolo 4

Introduzione

Scopo del progetto

Nel giugno 2007 è stato approvato, come standard a livello mondiale, dalla IEEE-Standard Association, un nuovo formato per rappresentare l'informazione musicale basato sul linguaggio di mark-up XML e chiamato *Standard IEEE 1599[1]*. Questo nuovo modo di scrivere e rappresentare i file musicali consente di far interagire l'informazione musicale con tutti i livelli del testo, del suono, delle partiture, del timbro attraverso una gestione e sincronizzazione spazio-temporale degli elementi. Permette, quindi, di rappresentare la musica in tutti i modi con la quale la si può identificare (notazione, simbolico, strutturale, audio).
Dalla sua creazione e standardizzazione sono stati scritti molti articoli a livello europeo e mondiale riguardanti lo standard IEEE 1599, e molte sono state le tesi presentate, atte a sviluppare l'utilizzo di questo nuovo linguaggio, sia a livello teorico, sia a livello pratico come, ad esempio, la creazione di software per la conversione da altri linguaggi basati su XML ad IEEE 1599, o software di analisi di partiture o analisi di file audio attraverso la lettura di file standard IEEE 1599.
Il progetto qui descritto nasce dall'esigenza di avere una collezione di file XML musicali standard IEEE 1599 archiviata in una base dati capace di rappresentarne la struttura, per quanto riguarda i meta-dati contenuti nei file.

Ottenere un Repository che sia interrogabile ed aggiornabile dall'applicazione di gestione dei file IEEE1599 Framework rende molto semplice la gestione dell'archivio dei brani permettendo ricerche veloci, modifiche, consultazione e riproduzione dei contenuti xml musicali direttamente dall'applicazione.

L'interfaccia di accesso via web inoltre, in futuro raggiungibile attraverso il link “Scores” presso il sito del Laboratorio di Informatica Musicale dell' Università degli Studi di Milano, permette di consultare il Repository anche al di fuori del campo applicativo utilizzando la tecnologia php/sql tramite un qualsiasi browser web. E' quindi possibile consultare anche da remoto la base dati per avere informazioni sullo stato del repository o effettuare ricerche.

Il tirocinio di sviluppo iniziato in dicembre 2009 e terminato a febbraio 2010 è stato supervisionato dal Laboratorio di Informatica Musicale dell'Università degli Studi di Milano, il quale ha fornito le specifiche da rispettare e gli aspetti più importanti da sviluppare durante la creazione della base dati e delle interfacce di interrogazione, è stato inoltre consigliato l'uso del software Object-relational database management system PostgreSQL nella sua versione 4.8.2 per eseguire la creazione del Repository. Infine il Laboratorio ha fornito una consistente quantità di file xml musicali IEEE 1599 con cui effettuare test di funzionamento sulla base dati, sui trigger di automazione, sulle interfacce del plugin integrato nel Framework IEEE 1599 e via Web.

Una volta acquisiti i mezzi e la dimestichezza con i software per svolgere il lavoro si è pensato a quali dati aveva senso estrarre dai file XML in nostro possesso per la loro indicizzazione e per l'eventuale ricerca, potendone estrarre e leggere il codice in ogni momento, oppure facendo interpretare l'XML direttamente dal Framework IEEE 1599.

Ciò che ne è scaturito è che le informazioni più utili e rappresentative ai fini di indicizzazione sono i meta-dati contenuti nel layer general dello standard IEEE 1599. Questo layer non è il cuore dello standard che invece è rappresentato dallo <Spine> e dalla capacità di sincronizzazione tra i vari aspetti musicali ma; come in un archivio musicale non si identificano i brani dal tema principale bensì dai meta-dati come autore, titolo, data di composizione, opera. Così anche per i file XML musicali IEEE 1599, ai fini di archiviazione e consultazione, l'aspetto fondamentale dell'informazione non consiste nell'informazione stessa ma, nei meta-dati ad essa associati in modo da distinguere e riconoscere univocamente un brano da tutti gli altri.

Nei capitoli successivi verrano illustrati più precisamente lo standard IEEE 1599 negli aspetti di maggior interesse per il progetto svolto, i software utilizzati facendo riferimenti alle guide ed alle fonti usate per apprendere meglio il loro uso e le funzioni utili al raggiungimento degli scopi del progetto e come è stato realizzato il Repository vero e proprio, spiegandone le logiche relazionali e le funzioni che sono state costruite al di sopra della base dati per facilitarne l'utilizzo da parte di utenti anche non esperti.

Capitolo 1

1.1 eXtensible Markup Language

XML è un metalinguaggio di markup, ovvero è un linguaggio formato da marcatori che definisce un meccanismo sintattico il quale consente di estendere o controllare il significato di altri linguaggi marcatori[2]. Il nome infatti indica che si tratta di un linguaggio marcatore (markup language) estensibile (eXtensible) in quanto permette di creare tag personalizzati arbitrari e liberi.

A prima vista il codice XML può essere confuso o associato con il linguaggio HTML essendo entrambi linguaggi di markup, anche se i due hanno scopi totalmente diversi.

HTML infatti si occupa di definire una grammatica per la descrizione e la formattazione di pagine web e, più in generale, di iper-testi, mentre XML si occupa di fornire la possibilità, tramite il metalinguaggio stesso, di creare nuovi linguaggi atti a descrivere documenti strutturali.

Esempio di HTML	Esempio di XML
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"> <html> <head> <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1"> <title>pagina</title> </head> <body> pagina web </body> </html>	<?xml version="1.0" encoding="UTF-8"?> <!DOCTYPE ieee1599 SYSTEM "http://standards.ieee.org/downloads/1599/1599-2008/ieee1599.dtd"[]> <ieee1599 version="1.0" creator="Luca A. Ludovico"> <general> <description> <main_title>Gottes Macht</main_title> <author type="composer">Beethoven, </author> <author type="poet">Gellert</author>

Fig. 1 – Confronto tra porzioni di codice HTML ed XML IEEE 1599

Allo scopo di rappresentare l'informazione musicale in tutti suoi aspetti è stato scelto XML per le sue principali caratteristiche che sono: l'interscambiabilità in rete, l'indipendenza dalla piattaforma di utilizzo, l'estensibilità, l'intelligibilità, e soprattuto per la sua struttura gerarchica simile a alla struttura propria dell'informazione musicale dove vi è una forte gerarchizzazione.

Come è stato detto XML è un linguaggio di markup in grado di gerarchizzare e organizzare le informazioni, come si vede nell'esempio i tag non sono espressioni standard di nessun linguaggio ma elementi descrittivi di ciò che si deve rappresentare.

Detto ciò per creare uno standard come IEEE 1599 è necessario specificare un significato preciso per ogni elemento, definendone gli attributi, i possibili contenuti e i possibili tag figli, generando così un documento che definisce lo standard di scrittura di un dato tipo di file.

Questo documento prende il nome di DTD (Document Type Definition) e normalmente si suddivide in: Element Type Definition, dove vengono specificate le standardizzazioni degli elementi dell' XML e, Attribute List Declaration dove per ogni elemento vengono definiti gli attributi possibili e la loro tipologia.

1.2 Lo standard IEEE1599

Si è scelto di sviluppare la base dati del Repository IEEE 1599 ad hoc per raccogliere il maggior numero possibile di informazioni riguardanti i meta-dati contenuti nei file XML standard IEEE 1599 perché si ritiene che, in futuro, questo tipo di file potrà crescerà di numero e in quantità di utilizzi, soprattuto da parte di chi lavora nella musica. Sarà quindi necessario, per agevolarne la consultazione e la ricerca, organizzarli in un Repository apposito consultabile dal Laboratorio di Informatica Musicale ma anche dall'esterno della rete universitaria. Ovviamente servono restrizioni e permessi differenti in base all'utenza che ne sta facendo uso ed in base anche alle richieste che vengono avanzate al database.

Si ritiene che lo standard xml musicale IEEE 1599 inoltre sia ad oggi il modo più completo di rappresentare la musica permettendo la sincronizzazione tra le informazioni audio, quelle di partitura, le informazioni di esecuzione elettronica e simulata ed infine, come avviene anche per gli mp3, il formato standard IEEE 1599 può contenere anche meta-dati relativi all'opera in esame, non limitandosi ai classici meta-dati riguardanti “titolo” “autore” “album” etc. ma contenendo anche informazioni più precise relative ai file audio oppure alle partiture ad esso associati e sincronizzati. In questo modo si ha una rappresentazione completa dell' informazione musicale, sia dal punto di vista sonoro ed acustico sia dal punto di vista scritto.

Questi aspetti della musica nello standard IEEE 1599 vengono rappresentati come strati (layers) dell'informazione musicale e ogni strato ne rappresenta un livello di astrazione differente, con diverse modalità di codifica possibili in base allo strato in cui ci si trova.

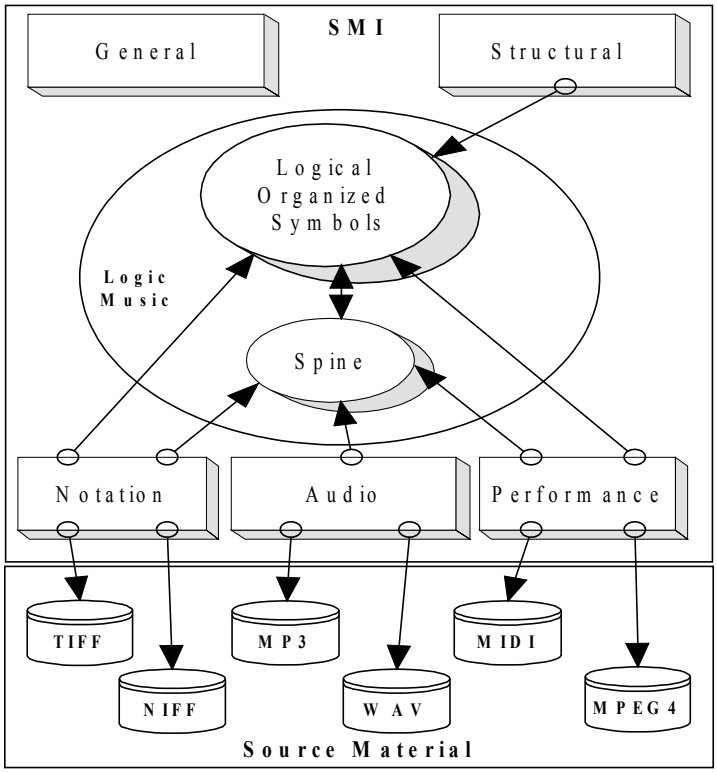

Fig. 2 - I Layer dello standard IEEE 1599[3]

Per quanto riguarda ad esempio l'aspetto scritto della musica, che fa riferimento al layer <notational>, un file standard IEEE 1599 contiene al suo interno appositi elementi XML utili a rappresentare, ad esempio, il "gruppo" di istanze grafiche ad esso collegate e sincronizzate, in pratica è possibile avere molte immagini connesse ad un brano IEEE 1599 ed ognuna di esse sarà figlia, insieme a tutte le altre appartenenti allo stesso gruppo, del tag <graphic_instance_group> (come visibile in Fig. 3). Questo elemento raccoglie le immagini di una stessa partitura e ne specifica l'editore oppure il software dal quale sono state esportate le pagine della stessa oppure, in mancanza di queste informazioni, un dato significativo che distingua quel gruppo di immagini come appartenenti ad una sola ed unica partitura.

L'evidenza della struttura gerarchica dell'XML qui appare molto chiara e definita, non è fatti possibile associare un immagine ad un file standard IEEE 1599 se prima non si è definito il gruppo di appartenenza dell'immagine. Il tutto deve avvenire all'interno del tag (layer) <notational> per far si che un file sia corretto ed abbia senso.

```
<notational>
 <graphic_instance_group description="Edizione Henle">
    <graphic_instance file_name="bist_du_bei_mir/score/henle/bist_du_bei_mir_01.tif"
        file_format="image_tiff" encoding_format="image_tiff" position_in_group="1"
        measurement_unit="pixels">
      <graphic_event event_ref="part_1_voice0_measure1_ev0" … lower_right_y="430" />
      <graphic_event event_ref="part_1_voice0_measure21_ev2" ... lower_right_y="1241" />
    </graphic_instance>

    <graphic_instance file_name="bist_du_bei_mir/score/henle/bist_du_bei_mir_02.tif"
        file_format="image_tiff" encoding_format="image_tiff" position_in_group="2"
        measurement_unit="pixels">
      <graphic_event event_ref="part_1_voice0_measure34_ev2" … lower_right_y="378" />
      <graphic_event event_ref="part_1_voice0_measure34_ev4".... lower_right_y="381" />
    </graphic_instance>
  </graphic_instance_group>

<graphic_instance_group description="Finale">
    <graphic_instance file_name="bist_du_bei_mir/score/finale/Bist_du_bei_mir_1.TIF"
        file_format="image_tiff" encoding_format="image_tiff" position_in_group="1"
        measurement_unit="pixels">
      <graphic_event event_ref="KeySignature_part_3_1" ... lower_right_y="882" />
    </graphic_instance>
  </graphic_instance_group>
</notational>
```

Fig. 3 – Esempio di struttura di graphic_instance e graphic_instance_group

L'idea di archiviare i file ed estrarre i dati automaticamente è nata analizzando i file XML musicali ed è il risultato dell'aver osservato come un file IEEE 1599 completato in ogni suo elemento e attributo contenga realmente tutto il brano musicale. Annesse ad esso si trovano infatti informazioni importanti per la sua indicizzazione e corretta archiviazione; dovendo sviluppare una base dati ove sistemare questi file non è stato necessario analizzare il tag <spine> creato per la

sincronizzazione dei diversi contenuti, ma l'attenzione si è rivolta in primo luogo all'unico strato non connesso allo spine e responsabile dei meta-dati principali. Il layer general infatti contiene le prime informazioni utili e quelle più importanti per l'archiviazione del brano XML IEEE 1599. General organizza al suo interno infatti; il titolo del brano, riconoscibile nel tag <main_title>, il titolo dell'opera a cui il brano appartiene, nel tag <work> con attributo "title" ed il nome dell'autore o degli autori che hanno partecipato alla stesura del brano con il relativo ruolo come ad esempio compositore, librettista etc. rintracciabili nel tag <author> con attributo “type”.

Lo stesso lavoro di analisi si è poi andato a svolgere per il layer notational, per il layer performance, relativo all'esecuzione realtime del brano tramite software, ed il più significativo e corposo per quanto riguarda i meta-dati di nostro interesse: il layer audio.

Il layer audio infatti ricopre un area significativa del Repository vista la notevole quantità ed organizzazione delle informazioni relative ai file audio possibilmente associati ad un brano XML musicale standard IEEE 1599.

Nello standard infatti ad un singolo brano possono essere associate più esecuzioni in formato audio, magari relative ad edizioni diverse (labels) e contenute in raccolte diverse (album). E' pertanto presente un apposito tag <album> che rappresenta la raccolta o l'opera madre a cui la traccia audio fa riferimento, con la relativa etichetta discografica che ne ha curato la produzione e le possibili informazioni sugli esecutori di quella specifica performance registrata sulla traccia audio, informazioni contenute nel tag figlio <performers>.

1.2.1 Il layer general

Il layer general[3] prende in considerazione il brano musicale nella sua interezza, dandone un inquadramento generale e raggruppando le informazioni sulle relative istanze. Esso contiene gli elementi <description> (di cui faremo un analisi più approfondita), <notes> (che rappresenta un elemento opzionale di tipo testo dedicato ad annotazioni generiche), <casting>, <related_files>, <analog_media> e <rights> (che non verranno trattati ai fini del progetto).

La parte relativa all'elemento <description> ai fini di archiviazione e ricerca risulta la più importante e di maggior rilievo perciò verrà ora trattata con particolare attenzione e con analisi dei tags trattati dal Repository IEEE 1599.

Description è un elemento figlio del layer general atto a descrivere l'informazione musicale dal suo lato informativo e descrittivo, in particolare tratta quelle informazioni utili a catalogare ed archiviare un file XML musicale standard IEEE 1599 analizzandone i seguenti aspetti tramite i tag sottoelencati:

<work_title>

Rappresenta un elemento testuale contenente il titolo dell'opera complessiva a cui il brano o movimento fa riferimento.

<work_number>

Rappresenta un numero di catalogo relativo all'opera descritta da <work_title>, se presente. Il dato non è per forza numerico ma può contenere anche caratteri testuali pertanto verrà codificato in un campo di tipo testuale.

<movement_title>

Rappresenta un elemento testuale contenente il titolo del movimento, ossia del brano che il file XML IEEE 1599 rappresenta.

<movement_number>

Rappresenta il numero ricoperto dal brano di titolo <movement_title> all'interno della raccolta di titolo <work_title>. Come per <work_number> il dato è di tipo testuale (ad esempio: “5 lato B”).

<author>

Rappresenta il nome o i nomi dell'autore del brano, deve esistere almeno un autore per ogni brano IEEE 1599 ma possono presentarsi autori multipli all'interno di <description>.

<author type= ”ruolo dell'autore”>

Type è un attributo del tag visto in precedenza e per ogni <author> ne specifica il ruolo che egli ha occupato nella scrittura del movement.
Ad esempio, sono ammessi type = “compositore”, type = “librettista”, type = “revisore” e via dicendo.

<genre>

Rappresenta un elemento testuale riservato alla descrizione del genere musicale. Ai fini del progetto è stato utilizzato il genere descritto nel layer audio perché più preciso e meglio rappresentativo.

```
...
 <general>
  <description>
   <main_title>Gottes Macht und Vorsehung</main_title>
   <author type="composer">Beethoven, Ludwig van</author>
   <author type="poet">Christian Fürchtegott Gellert</author>
   <number>5</number>
   <work_title>6 Geistliche Lieder Von Gellert</work_title>
   <work_number>op. 48</work_number>
   <genres>
    <genre name="Vocal music" />
    <genre name="Romanticism" />
   </genres>
  </description>
 </general>
…
```

Fig. 4 – Layer general dello standard XML musicale IEEE 1599

1.2.2 Il layer audio

Si è deciso di trattare in maniera più approfondita rispetto ai layer notational e performance il layer audio[3]; in primis per la sua consistente presenza in ambito del progetto ed in secondo luogo per la sua importanza a livello musicale.

Audio è il livello più basso dello standard musicale XML IEEE 1599 e descrive le proprietà del materiale musicale sonoro nei diversi formati che possono rappresentarlo. Vi sono due possibili categorie di formati per rappresentare l'informazione sonora: i formati compressi ed i formati non compressi.

I suoni infatti possono essere codificati in formati come WAV, PCM, AIFF oppure possono essere soggetti ad algoritmi che ne riducono le dimensioni su disco, questi algoritmi possono inoltre essere di due tipi; lossles oppure lossy. I primi non prevedono perdita di informazione acustica e sfruttano algoritmi come i codici di Huffman, che eliminano semplicemente le ridondanze nei bit e con i quali è possibile tornare al segnale originario senza perdere informazione.

Gli Algoritmi lossy (con perdita di informazione), invece, utilizzano funzioni specifiche che eliminano parti di segnale audio. Tipicamente viene cancellato il segnale non percepito dall'orecchio umano, come ad esempio le bande di frequenza non udibili oppure i mascheramenti. Mascheramenti che possono verificarsi sia nel dominio delle frequenze che in quello temporale e, come già detto, l'orecchio umano non è in grado di percepire per le sue caratteristiche fisiologiche.

Gli algoritmi che sfruttano questi principi per eliminare informazioni “inutili” arrivando a tassi di compressione molto alti pur mantenendo un livello qualitativo sonoro simile all'originale.

Avendo, qui sopra, parlato di che tipi di tracce descrive il layer audio si andrà ora ad illustrarne la struttura ed i tag utilizzati ai fini dello sviluppo del Repository IEEE 1599.

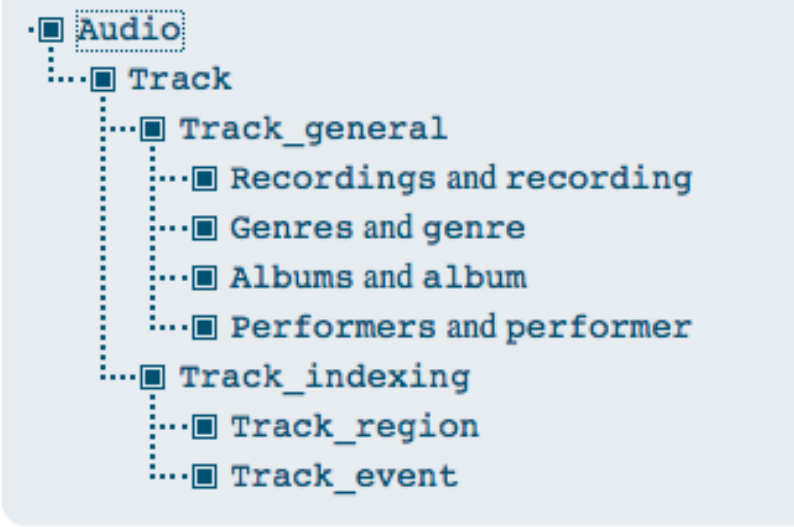

Fig. 5 – Albero rappresentativo del layer audio

<audio>

E' il tag padre ed il contenitore di tutto lo strato.

<track>

Elemento figlio diretto del layer <audio> contiene i riferimenti alla specifica traccia sonora in esame, possono esservi più tracce connesse ad un unico documento IEEE 1599 e saranno rappresentate da più <track>.

Questo tag possiede due sotto elementi; <track_indexing> di cui non si specificheranno i particolari perché è utile allo spine ed alla sincronizzazione dell'audio con eventuali altri elementi ma non ai fini di archiviazione, e <track_general> di cui ora si specificheranno i dettagli.

<track_general>

E' il tag che descrive i meta-dati di una traccia audio a cui un documento XML musicale IEEE 1599 fa riferimento. Come si vede dalla struttura in *Figura 5* track_general raccoglie al suo interno altri elementi con funzioni precise.

<recording>

Descrive gli attributi della registrazione audio in esame: luogo di registrazione, data, e dati riguardanti lo studio di ripresa.

Dato il suo ancora scarso utilizzo non è stato indicizzato ai fini di archivio nel Repository ma è possibile, se necessario in futuro, implementare anche queste informazioni molto facilmente nel Repository aggiungendone le relative tabelle all'interno dell'area audio.

<album>

descrive i meta-dati relativi all'album in cui la traccia è contenuta utilizzando gli attributi:

title	Descrive il titolo della raccolta di appartenenza della traccia
track_number	Specifica la posizione della traccia nella raccolta di appartenenza
catalogue_number	Specifica il numero di catalogo (se presente) della raccolta
number_of_tracks	Specifica il numero totale di tracce della raccolta
publication_date	Descrive la data di pubblicazione della raccolta
label	Contiene il nome dell'etichetta discografica che ha pubblicato la raccolta

<performer>

Descrive gli esecutori della traccia audio di riferimento sfruttando l'attributo “type”, ove si indica il ruolo del musicista nell'esecuzione del pezzo (es. “primo violinista”).

<genre>

Descrive il genere di appartenenza della traccia audio, fornendone il nome con l'attributo “name”, una descrizione tramite l'attributo “description”, ove si possono specificare particolarità del genere stesso o diverse correnti, ed infine la possibilità di dare una percentuale di appartenenza tramite l'attributo “weight”.

L'attributo “weight” consiste in un numero decimale rappresentante la “quantità” di appartenenza della traccia al genere indicato in “name”, dando così una precisa descrizione del brano in esame.

1.3 Repository e basi dati

Un Repository è un ambiente di un sistema informativo in cui vengono gestiti i meta-dati attraverso tabelle relazionali, ossia l'insieme di tabelle, regole e motori di calcolo tramite cui si gestiscono i meta-dati si basa sulle tecnologie dei database relazionali [4].

Si tratta di qualcosa di più sofisticato del mero dizionario dati; è un ambiente che può essere implementato attraverso numerose piattaforme hardware e sistemi di gestione delle basi dati, implica la collaborazione di più software o tecnologie che ne permettano il funzionamento.

Nel nostro caso il database è stato generato tramite PostgreSQL 8.4.2 e viene interrogato ed aggiornato attraverso un plugin sviluppato per il software di gestione e visualizzazione dei file XML IEEE 1599 chiamato Framework IEEE 1599.

Il plugin è stato sviluppato nell'ambiente .NET 3.5 in linguaggio c# e si connette al repository usando il driver ODBC PSQLODBC attraverso la tecnologia ado.net, inoltre i dati posso essere visualizzati anche da web, attraverso un qualsiasi browser, grazie all'interfaccia sviluppata in SQL, PHP, ed HTML e caricata sul sito ufficiale del Laboratorio di Informatica Musicale dell'Università degli Studi di Milano.

L'interfaccia accessibile da web non permette il caricamento, la modifica o la cancellazione di dati archiviati sul Repository IEEE 1599 in quanto essa è progettata per essere utile solo a scopo di ricerca di un file e di consultazione dello stesso.

E' permesso però ottenere e visualizzare il codice XML del file ricercato per analizzarne la struttura ed il codice oppure ricercare eventuali errori. Al momento i file IEEE 1599 sono archiviati sulla base dati senza i file multimediali connessi per ragioni di copyright e di spazio occupato, non è quindi possibile effettuare il download delle tracce audio connesse o delle partiture, pratica che se implementata dovrà essere regolata da vincoli sui diritti d'autore e diritti connessi.

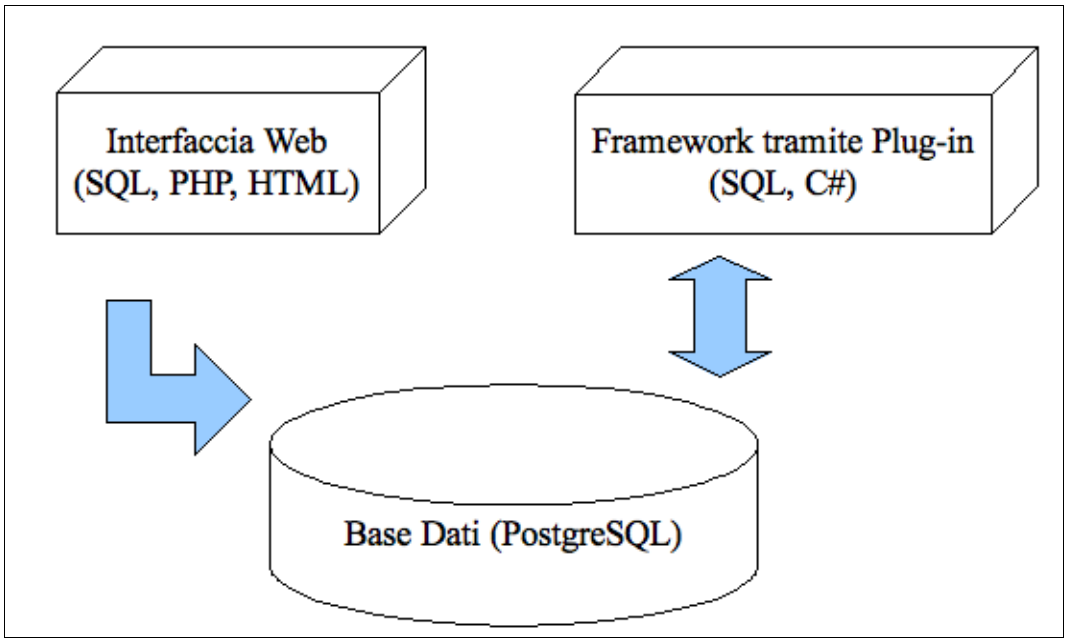

Fig. 6 – Schema di collegamento tra le tre applicazioni del Repository IEEE 1599

Uno schema architetturale di questo tipo implica che i sottosistemi che compongono il software accedono e modifichino una singola struttura dati chiamata appunto Repository.

Questi vari sottosistemi sono fra loro "relativamente indipendenti", in quanto interagiscono mediante la base dati ma comunque solamente attraverso quest'ultima. Il flusso di controllo è dettato o dal repository, attraverso un cambiamento dei dati in esso memorizzati, o come nel nostro caso dai singoli sottosistemi generando un cambiamento nei dati archiviati (il caso di inserimento o cancellazione di file).

Nella Figura 6 si nota che l'interfaccia web può solo accedere al database richiedendo informazioni, mentre il Framework (che ha anche un verso della freccia verso di se) ha la possibilità di modificare il contenuto del database PostgreSQL.

Capitolo 2

2.1 Il linguaggio SQL

Nel capitolo precedente si è analizzato oltre che il linguaggio XML anche i database relazionali ed in particolare il concetto di Repository.
Ora si tratterà di come interagire con la base dati per estrarne o modificarne le informazioni contenute, in particolare, si esporrà come, tramite il linguaggio SQL, è possibile generare interrogazioni e ottenere risultati andando a fare delle richieste specifiche (query) al Repository.
Tramite lo **Structured Query Language** infatti è consentito eseguire varie operazioni sia sui dati che sulle strutture che li contengono come, ad esempio, le tabelle.
Questo linguaggio viene definito di alto livello o linguaggio dichiarativo, perché permette di svolgere operazioni dichiarando cosa si deve ottenere e non come si deve ottenere, esso è infatti considerato uno dei linguaggi di terza generazione o, meglio definiti, procedurali[5].
I linguaggi dove bisogna specificare come qualcosa si fa e non è sufficiente dichiarare cosa si deve fare infatti si dicono linguaggi procedurali.
Più da vicino ora si analizzeranno le funzioni base che sono state utili alla stesura del Repository ed occorrono alla sua interrogazione.

```
CREATE TABLE "TFile_MX"
(
  "ID_file" bigserial NOT NULL,
  main_title text,
  file_name text,
  other_title text,
  notes text,
  file_xml xml NOT NULL,
  upload_date date,
  CONSTRAINT "PK_file" PRIMARY KEY ("ID_file")
)
```

Fig. 7 – codice di creazione di "TFile_MX"

Il codice riportato in Figura 7 è l'SQL utilizzato per creare la tabella "TFile_MX" tramite la funzione di CREATE e ne definisce ogni colonna con il relativo tipo, se necessario inoltre ne specifica la condizione di esistenza necessaria: NOT NULL.
In fase di creazione di una tabella è anche necessario definire una chiave primaria (PK), la chiave primaria è una colonna che possiede due caratteristiche fondamentali che permettono di distinguere i record tra loro, essa infatti deve essere non nulla e univoca.
Questa doppia condizione garantisce che ogni record sia unico e diverso da tutte gli altri record della tabella. Il modo più sicuro ed efficiente per generare una PRIMARY KEY è assegnare un identificativo numerico intero generato sequenzialmente ed automaticamente dal Repository per ogni record creato.
L'operazione di CREATE, come si è visto, è una funzione che modifica la base dati, quindi un operazione strutturale sul Repository, vi sono anche altre operazioni possibili tramite il linguaggio SQL:

```
INSERT INTO "tabella" (colonna1, colonna2) VALUES (dato1, dato2);
```

Permette di inserire nuovi dati all'interno di una tabella, nuovi record vengono quindi creati inserendo i dati rispettivamente in corrispondenza delle colonne selezionate tra parentesi dopo il nome della tabella.

```
SELECT colonna1,colonna2 FROM “tabella” WHERE [condizione];
```

Questa riga di codice permette di generare una tabella temporanea di visualizzazione dei dati estratti dalla tabella “tabella”, in particolare le colonne specificate dopo il codice di SELECT e filtrando i dati tramite condizioni da mettere dopo la clausula WHERE.

```
UPDATE "tabella" SET  colonna1 = dato1  WHERE [condizione];
```

La funzione di UPDATE invece permette di selezionare una “tabella” e modificare il contenuto di una sua colonna, come per la SELECT è applicabile una clausula WHERE che se non viene specificata genere un UPDATE per ogni record della tabella selezionata.

```
DELETE FROM "tabella" WHERE [condizione];
```

Ultima delle più semplici funzioni di manipolazione dei dati che si presenterà qui è la funzione che permette di cancellare un record dalla tabella specificata dopo la clausula FROM e quando si verifica la condizione specificata nella clausula WHERE, ed anche in questo caso omettendo quest'ultima verranno eliminati tutti i record della tabella selezionata.

2.2 PostgreSQL

Il Software utilizzato per generare il Repository IEEE 1599 è un Object-relational DataBase Management System (ORDBMS) chiamato PostgreSQL basato su Postgres, completamente Open Source e sviluppato in America.

Al Berkeley Computer Science Department della University of California nella seconda metà degli anni novanta è stato sviluppato questo strumento di creazione e gestione per database relazionali completo e compatibile con molti linguaggi e standard internazionali[6].

PostgreSQL si integra perfettamente con lo standard di SQL, raccogliendone e utilizzandone tutte le funzioni. In più questo ORDBMS permette di generare query complesse, fornisce un facile uso delle Foreign Keys, permette l'uso di Trigger, la creazione di viste sui dati ed infine la scrittura di funzioni e procedure in linguaggio PL/pgSQL , ma anche in Java o in C per esempio.

Inoltre in una base dati PostgreSQL sono generabili campi che in altre piattaforme non si possono generare, uno ad esempio molto utile allo sviluppo del Repository IEEE 1599 è il campo XML in grado di contenere un file XML codificato senza alterarlo e permettendone il parsing con apposite funzioni, come documento XML e non come testo.

Infine PostgreSQL è un Open Source project[6], non richiede quindi l'aquisto di una licenza per farne uso ed esiste una comunità ufficiale dove gli utenti più e meno esperti possono scambiarsi pareri, consigli e contribuire a migliorare l'ambiente di lavoro ed il software stesso.

2.2.1 Le Sequenze

PostgreSQL non permette, durante la creazione di una tabella, la dichiarazione esplicita di una Primary Key diretta come avviene in MySQL, ma sfrutta un meccanismo più articolato.

Durante la creazione di una tabella viene definita una variabile che si decide essere la Primary key per quella tabella. Dopo averla creata e definita con il corretto tipo di colonna (il tipo di una chiave primaria deve consentirne sempre l'esistenza e l'univocità), si deve generare una sequenza legata a quella colonna come si vede nel codice in figura 8.

In caso si generi un identificativo per le tuple questo dovrà essere preferibilmente di tipo numerico intero e di una lunghezza congrua alle esigenze di capacità del Repository, nel caso in esame, un dato di tipo biginteger garantisce valori da 1 a 9.223.372.036.854.775.807 quindi abbondantemente sufficiente a generare id sequenziali per i file da archiviare.

Il meccanismo utilizzato da PostgreSQL come detto è quello di associare alla variabile biginteger una "sequenza".

```
CREATE SEQUENCE "TFile_MX_ID_file_seq"
  INCREMENT 1
  MINVALUE 1
  MAXVALUE 9223372036854775807
  START 1
  CACHE 1;
ALTER TABLE "TFile_MX_ID_file_seq" OWNER TO postgres;
```

Fig. 8 – Codice di creazione di una sequenza

La sequenza è infatti una funzione che viene invocata ad ogni INSERT effettuata sulla tabella a cui è collegata e si occupa di inserire, nel campo selezionato, un nuovo valore numerico intero incrementando questo del passo specificato dal parametro INCREMENT rispetto all'ultimo valore inserito.

Per fare ciò è chiaro che il sistema deve tenere traccia dell'ultimo valore generato dalle sequenze per poterlo incrementare, questa particolarità risulterà molto comoda in seguito in quanto dato utile alla scrittura del trigger di inserimento dove tramite un apposita funzione (currval) è possibile recuperare l'ultimo dato generato da una precisa sequenza e quindi capire qual'è l'ultimo record inserito nella base dati.

Altri possibili fattori della sequenza, come si può leggere in figura 8, sono il valore minimo che la stessa può assumere (MIN VALUE), il suo valore massimo che di default viene importato dal sistema al numero massimo che il tipo di colonna può sostenere ed un valore di partenza determinato dal comando START.

2.2.2 PgAdmin III

Per scrivere la struttura della base dati e delle funzioni di trigger si è fatto uso della suite messa a disposizione direttamente dal sito ufficiale di PostgreSQL chiamata PgAdmin e giunta alla sua terza versione.

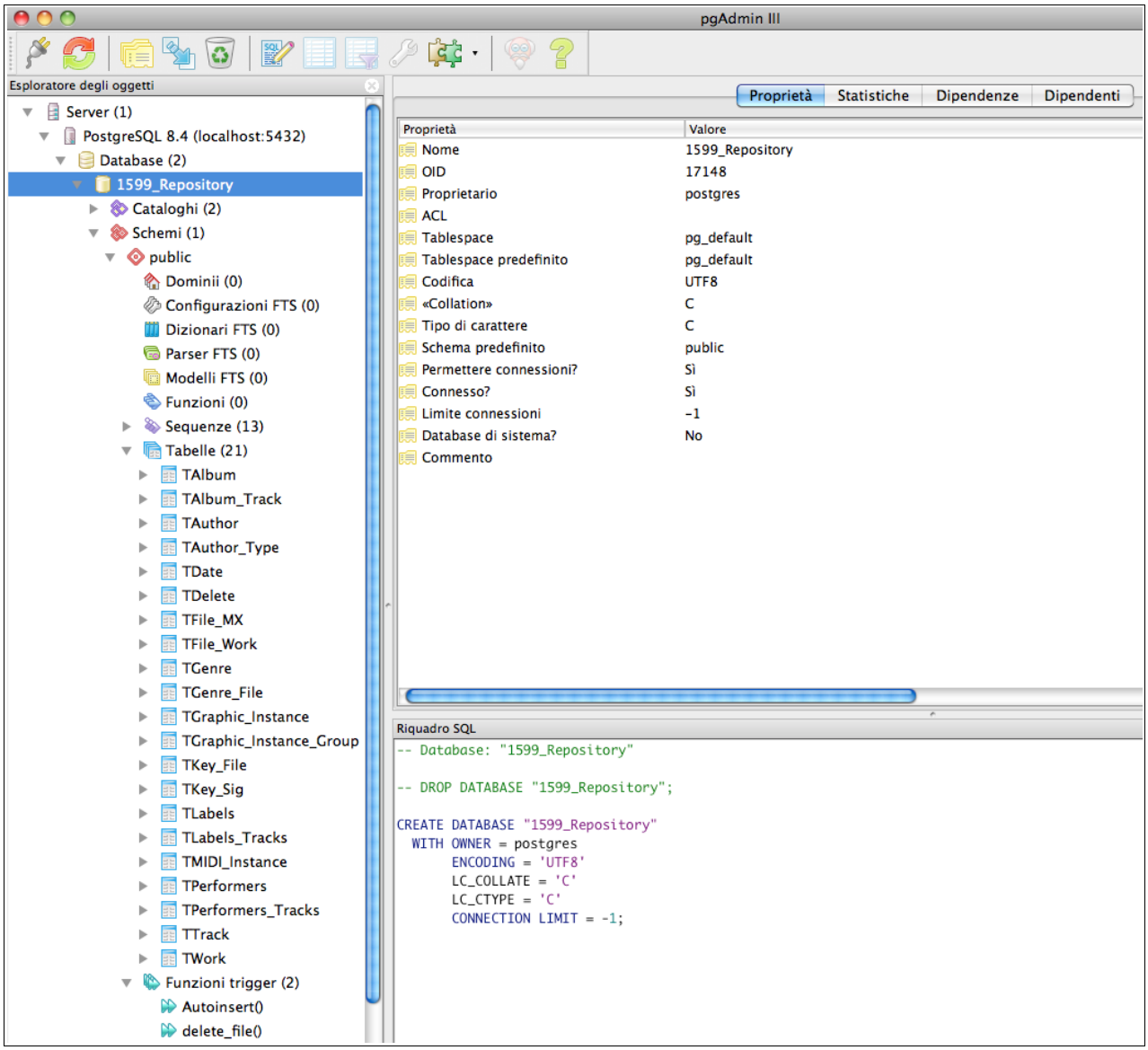

Fig. 9 – Finestra di PgAdmin III con visualizzate tabelle e trigger del Repository IEEE 1599

Questa graphic user interface (GUI) permette la visualizzazione di tutte le tabelle della base dati, delle sequenze, delle funzioni, dei triggers e delle basi dati che si sono scritte utilizzando una visualizzazione ad albero dove è possibile nascondere o chiudere gli elementi di scarso interesse per visualizzare solo ciò che al momento è in uso.

Offre una finestra SQL dove scrivere le query di interrogazione, ma anche quelle di creazione delle tabelle in SQL. Le tabelle e le query strutturali si possono anche far generare da PostgreSQL utilizzando le maschere di creazione automatica messe a disposizione da PgAdmin III, il codice generato sarà comunque sempre visualizzabile e modificabile. Per quanto riguarda le query di interrogazione, anch'esse possono essere direttamente scritte in SQL oppure possono essere generate utilizzando il costruttore grafico di query con cui è possibile far generare il codice SQL al PgAdmin III disegnando le relazioni tra le tabelle e ponendovi i vincoli necessari.

Questa funzione è molto semplice da utilizzare e comoda se si devono creare velocemente query per effettuare controlli in fase di test del Repository, è simile a quello che si può fare in Microsoft Acces ma genera query anche complesse e l'SQL generato è privo di codici proprietari.

Come si può vedere in Fig. 9 inoltre sono visualizzate le funzioni di trigger che è possibile scrivere modificare direttamente attraverso PgAdmin tramite la scheda “Descrizione” del menù proprietà raggiungibile cliccando con il tasto destro sulla funzione desiderata.

La scrittura del codice di trigger avviene direttamente nell'interfaccia della suite e non servono quindi compilatori separati, o editor per codice, PgAdmin provvedere anche, quando si clicca sul bottone “Aplica” posto sotto la funzione ad effettuarne un controllo e restituire eventuali errori di compilazione del codice. Come avviene in Java quindi si posso correggere gli errori di compilazione prima di tentare l'esecuzione della funzione di trigger.

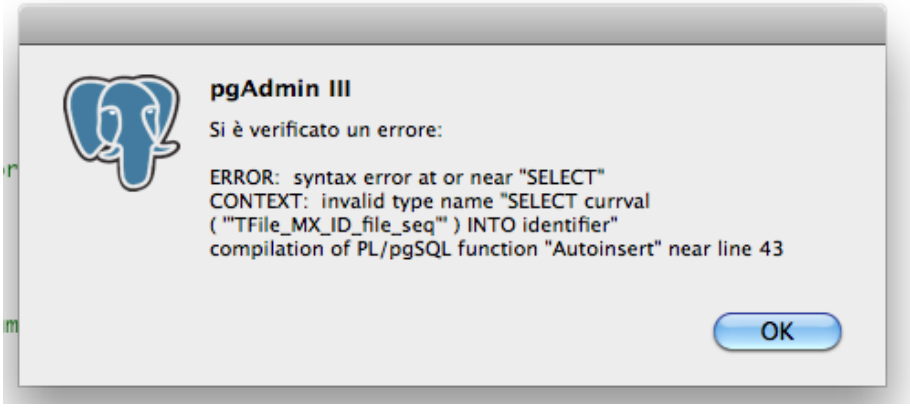

Fig. 10 – Esempio di schermata di errore di PgAdmin

2.3 Linguaggi Procedurali

Nei linguaggi procedurali o imperativi (assembler, Basic, Fortran) l'istruzione è un comando esplicito, che opera su una o più variabili oppure sullo stato interno della macchina, e le istruzioni vengono eseguite in un ordine prestabilito. Scrivere un programma in un linguaggio procedurale significa essenzialmente occuparsi di cosa la macchina deve fare per ottenere il risultato che si vuole, e il programmatore è impegnato nel mettere a punto gli algoritmi necessari a manipolare i dati.

Le strutture di controllo assumono la forma di istruzioni di flusso, come ad esempio GOTO, FOR, IF/THEN/ELSE ecc. e il calcolo procede per iterazione piuttosto che per ricorsione. I valori delle variabili sono spesso assegnati a partire da costanti o da altre variabili (assegnamento) e raramente per passaggio di parametri (istanziazione).

2.3.1 Pl/PgSQL

PostgreSQL permette l'uso di alcuni linguaggi al suo interno per ovviare a problemi che l'SQL non può risolvere perché inadatto come linguaggio di programmazione e non essendo in grado di costruire logiche complesse. Ad esempio SQL non supporta molti dei principali operatori di base dei linguaggi di programmazione, come le strutture di controllo quali cicli e strutture condizionali[7].

Perciò è stato integrato in PgAdmin III ed in PostgreSQL il linguaggio procedurale PL/PgSQL (Procedural Language/PostgreSQL Structured Query Language) molto simile al PL/SQL implementato in Oracle.

PL/pgSQL, da vero linguaggio di programmazione, permette un maggior controllo del semplice SQL, includendo l'abilità di usare cicli e strutture di controllo avanzate. I programmi creati nel linguaggio PL/pgSQL sono chiamati funzioni, e possono essere chiamati come parti di un'istruzione SQL, o attivati da trigger.

Questo linguaggio procedurale è stato creato in modo da svolgere operazioni complesse al di là delle potenzialità dell'SQL pur rimanendo semplice da utilizzare e capire, è sufficiente avere conoscenza delle principali strutture presenti nei linguaggi di programmazione procedurali.

Una funzione scritta in Pl/pgSQL avrà una struttura del tipo:

```
DECLARE
dichiarazione di variabili da utilizzare
BEGIN
qui viene scritta la funzione da svolgere
END;
```

Molto simile quindi ad un programma in cui si specificano le variabili da utilizzare prima di scrivere il vero e proprio codice sequenziale che la macchina dovrà eseguire al momento del richiamo della procedura.

2.3.2 Xpath

XPath è un linguaggio parte della famiglia XML che permette di individuare i nodi all'interno di un documento XML. Le espressioni XPath, a differenza delle espressioni XML, non servono a identificare la struttura di un documento, bensì a localizzarne con precisione i nodi.

XPath è nato originariamente dall'esigenza di fornire una sintassi e un comportamento comune fra XPointer e XSL; è stato successivamente adottato dagli sviluppatori come metodo di interrogazione dei dati in formato XML.

La versione 1.0 di XPath è diventata uno standard W3C il 16 novembre 1999[8].

In PostgreSQL viene implementata come funzione XML la funzione Xpath per processare dati di tipo XML basandosi sulla versione 1.0 standard sopracitata.

```
xpath('xpath', xml);
```

La funzione possiede due argomenti, il primo deve contenere il percorso (path) del nodo XML che si desidera raggiungere e di cui si vuole prelevare il contenuto, il secondo argomento è il documento, campo, o variabile XML sulla quale si vuole effettuare la ricerca del nodo.

Xpath restituisce un vettore di elementi XML contenente i risultati del parsing restituendo il contenuto del tag se la stringa di ricerca sarà del tipo:

```
xpath ('/ieee1599/general/description/author/text()', codice xml);
```

Mentre se si desidera estrarre un attributo, la stringa di ricerca sarà del tipo:

```
xpath ('/ieee1599/general/description/author/@type');
```

Capitolo 3

3.1 Il Database

3.1.1 Introduzione e tabella madre

Il progetto, come spiegato in precedenza, vuole creare un Repository adatto a rappresentare file XML musicali standard IEEE 1599 e a facilitarne la ricerca e consultazione, pertanto il primo passo intrapreso è stato analizzare il codice XML fornito dal Laboratorio di Informatica Musicale e trovare il modo migliore per rappresentarlo in una base dati sviluppata con PostgreSQL.

I meta-dati necessari ad una corretta indicizzazione sono stati raggruppati nelle opportune tabelle evitando ridondanze e fornendo i relazioni logiche opportune per ogni file da immagazzinare nel Repository.

Le informazioni più rilevanti ai nostri scopi in un file di tipo xml musicale IEEE1599 sono quelle contenute negli elementi figli del layer <general>; come ad esempio <main_title>, <author> con il relativo attributo "type", <work> ed i tag ad esso legati. Questi elementi rappresentano il brano in analisi con i suoi eventuali autori e l'eventuale opera d'origine in cui esso è contenuto o raccolto.

Un altro motivo che ha spinto lo sviluppo del Repository IEEE 1599 è quello di permettere la visualizzazione del codice XML sia attraverso il Framework IEEE 1599, creato dal Laboratorio di Informatica Musicale, che da web tramite un'interfaccia implementata con il linguaggio PHP e HTML.

Anche per questo motivo è stato opportuno creare una tabella “madre” centrale che contenesse l'intero codice XML del brano IEEE 1599 in un apposito campo di tipo XML chiamato "codice_xml".

Questo campo è inserito nella main table del Repository IEEE 1599 a cui tutte le altre informazioni, accessorie e non obbligatorie, sono connesse tramite relazioni logiche.

"TFile_MX" è quindi il fulcro della base dati, l'unica tabella a contenere il campo garantito e fondamentale del Repository, in essa sono stati poi inseriti un campo “ID_file” per contraddistinguere univocamente i brani IEEE1599 e alcune delle informazioni accessorie ma importanti contenute nel codice xml come il titolo, le note relative al brano, il possibile "other_title", il file name generato dal plugin di upload scritto per il Framework IEEE 1599 e la data di upload del brano: "Upload_date" anch'essa generata dal plugin di caricamento e utile a capire quando un file è stato caricato, modificato o aggiornato.

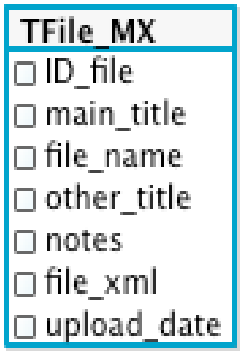

Fig. 11 – Tabella “TFile_MX”

3.1.2 Layer general

Per loro natura le altre informazioni sono state raccolte opportunamente in tabelle separate organizzando la base dati in aree corrispondenti ai layer dello standard IEEE1599.

Il layer general è rappresentato dalle tabelle: "TAuthor" e "TAuthor_Type", "TWork" e TFile_Work", dalla look-up table "TDate" e dalla coppia di tabelle "TGenre" e "TGenre_File". Strutturati come si vede nella figura 12 i legami logici che legano le tabelle “TAuthor” e “TWork” alla tabella principale sono legami di tipo molti a molti.

Un autore può essere presente in più file XML dato che nella sua vita creerà molto probabilmente più di un solo brano, ed un brano a sua volta può essere stato scritto da più autori che partecipano alla sua creazione con ruoli probabilmente diversi; il campo "type" della tabella di relazione "TAuthor_Type" a questo proposito si occupa di differenziare il ruolo di ogni autore per i singoli brani archiviati sul Repository.

Le opere raccolte nella tabella “TWork” hanno una struttura uguale a quella degli autori, il campo "number" specifica la posizione del brano nella collezione di appartenenza ("work_title") e fa riferimento all' elemento <number> dello standard XML musicale IEEE 1599.

Number si differenzia dal campo "work_number" della tabella "TWork" in quanto quest'ultimo si occupa di registrare il numero di catalogo della collezione o opera in cui il brano è contenuto se la suddetta collezione è parte di un archivio numerato.

La tabella che contiene le date presenti nel (o nei) tag <date> seleziona e archivia la data e la sua descrizione in quanto, nello standard IEEE 1599, possono essere presenti più date aventi significati differenti; ad esempio "prima rappresentazione pubblica", "data di composizione" eccetera. Diventa quindi opportuno registrare nel repository ogni data (anche se ripetuta) con il relativo significato che essa possiede.

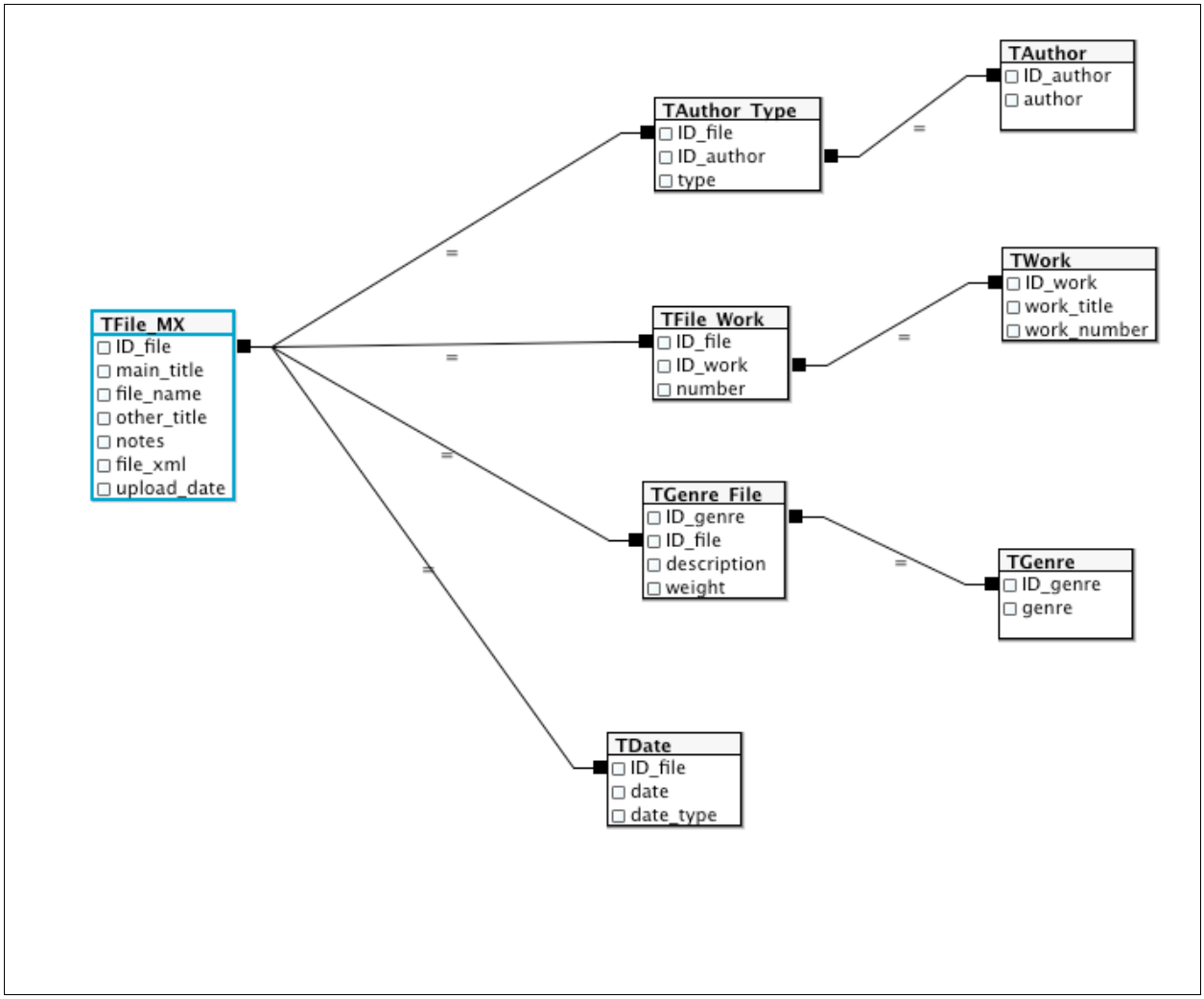

Fig. 12 – Tabelle appartenenti al layer general con le relative relazioni

L'ultima coppia di tabelle relative al layer general dello standard IEEE 1599 rappresentano il ramo della base dati relativo alla rappresentazione del genere (o dei generi) a cui il brano fa riferimento. Per la sua struttura nello standard XML, si è scelto di utilizzare una tabella di appoggio "TGenre_File" dove inserire anche i parametri di peso "weight" e di descrizione "description" che sono utili a dare una percentuale del genere a cui il pezzo appartiene ed una descrizione più accurata di quest'ultimo per definire con più precisione il contenuto audio del file.

Nello standard IEEE 1599, inoltre, è possibile associare più generi per ogni singolo brano. Risulta dunque necessario procedere con una relazione molti a molti con la tabella principale dato che ovviamente un genere musicale è legato a più brani.

3.1.3 Il layer performance

Analizzando il layer performance l'attenzione cade sulle istanze midi raccolte all'interno dei singoli file e utili all'esecuzione in realtime, da parte di un computer, tramite l'utilizzo di strumenti virtuali comandati con il protocollo MIDI.

La tabella "TMIDI_Instance" si occupa di raccogliere le informazioni contenute all'interno degli attributi "file_name" e "format" dell'elemento <midi_instance> figlio appunto del layer <performance>.

Il file midi (come i file audio o le immagini delle partiture) associato non viene effettivamente registrato nella base dati per ragioni di peso sulla struttura e per lo scopo di sola ricerca e visualizzazione dei dati che si desidera avere tramite l'interfaccia di consultazione via web. Inoltre non si è ritenuto opportuno inserire anche le informazioni riguardanti le instance CSound perché ancora poco presenti nei file finora in nostro possesso e ove presenti poco rilevanti.

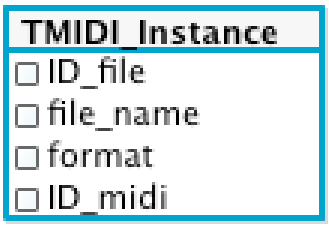

Fig. 13 – Tabella relativa alle istanze midi (layer performance)

3.1.4 Il layer notational

Nello standard XML musicale IEEE 1599 come già detto in precedenza può essere raccolto ogni aspetto dell'informazione musicale, per questo è fondamentale rappresentare dove presenti le forme scritte del brano in esame.

Il layer notational insieme al LOS (Logically Organized Symbols) si occupa di rappresentare ciò che si sente dai formati audio o viene letto per il layer performance sotto forma di partitura o di tablatura e archivia il risultato in file esterni al file XML in possibili vari formati.

Le partiture o tablature infatti possono essere in formato: pdf, tiff o jpeg. Le informazioni riguardanti questi dati e i file accessori sono raccolte all'interno degli attributi dell'elemento <graphic_instance> i quali forniscono il nome del file, il suo formato, la codifica adottata per la scrittura di quel file e la posizione che l'immagine occupa nel gruppo di appartenenza.

Ogni pagina della partitura o tablatura infatti deve appartenere ad un "gruppo" definito nel tag padre di <graphic_instance> che è appunto <graphic_instance_group> che nel suo attributo "description" definisce, ad esempio, a quale edizione appartiene quel gruppo di immagini oppure da quale software è stato esportata la partitura come ad esempio Finale.

Nel Repository, l'informazione riguardante il gruppo di immagini è stata inserita in una look-up table per minimizzare le ridondanze e velocizzare le ricerche e l'indicizzazione dell'area graphic_instance del database. È stata fatta questa scelta perché una partitura di un brano classico, spesso, è composta da un numero considerevole di pagine ed ogni pagina è scansionata come singola immagine nella maggior parte dei casi. Perciò è utile effettuare ricerche in primis per “gruppo”, in modo da identificare la partitura interessata, e successivamente si possono selezionare le singole immagini appartenenti a quella specifica partitura del brano in esame.

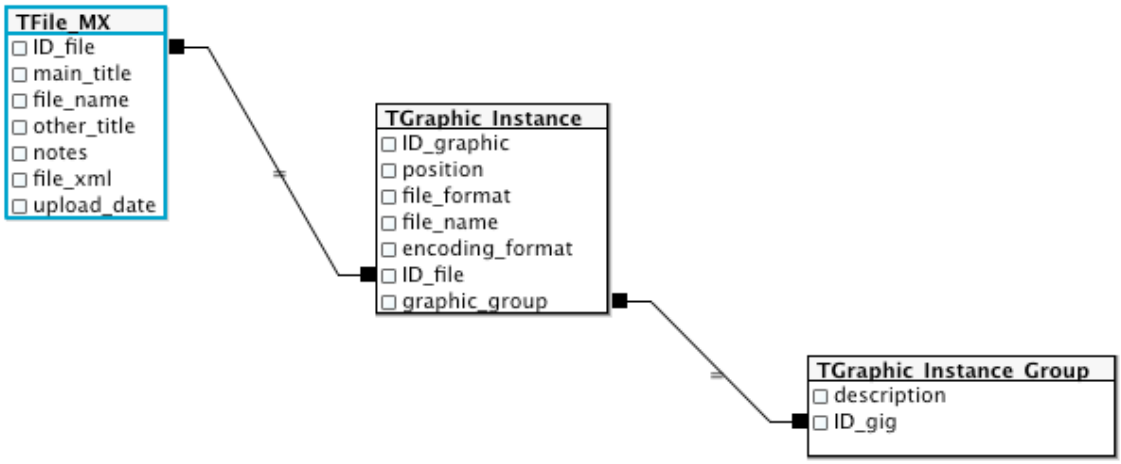

Fig. 14 – Tabelle appartenenti al layer notational

3.1.5 Layer audio

Il più analizzato e meglio rappresentato, trattandosi di informazione musicale, è il layer audio dello standard IEEE 1599.

Questo strato è adatto a raccogliere le tracce audio connesse al file XML in esame e data la sua struttura nello standard si è proceduto ad un analisi partendo dal tag <track>.

L'elemento <track> figlio diretto del layer <audio>, possiede a sua volta l'elemento figlio <track_general> il quale, in modo simile al layer general per l'intero file XML, raccoglie al suo interno tutte le informazioni relative alla traccia audio.

Gli elementi figli di <track_general> infatti si occupano di descrivere in particolare l'album o gli album di appartenenza della traccia, la sua label ed i musicisti che hanno preso parte a quella precisa registrazione raccogliendo così informazioni possibilmente differenti per ogni traccia audio e non connesse alle informazioni di "work" del layer general.

È possibile infatti avere più registrazioni di un'opera, magari raccolte in “compilation” diverse, suonate da diverse orchestre ed edite da case discografiche diverse ma nonostante la loro indipendenza come tracce audio, se queste fanno riferimento alla stessa opera, allora saranno associate ad un unico file XML IEEE 1599 pur mantenendo la loro identità ed i loro attributi identificativi.

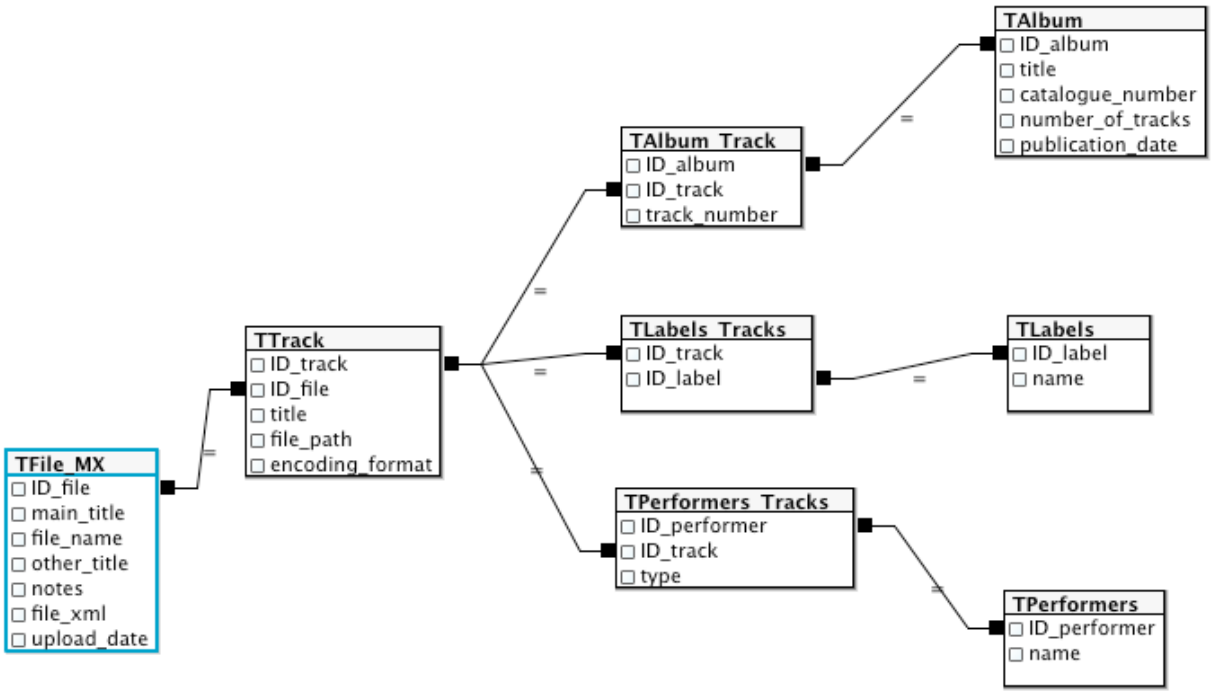

Fig. 15 – Il layer audio

Il layer audio in conclusione rappresenta una sotto-area della base dati che potrebbe avere anche esistenza propria se si volesse rappresentare solo la parte prettamente musicale ed acustica della musica.

È invece inserita qui in una visione più ampia del brano musicale e quindi raccolta all'interno di una struttura madre più grande in grado di rappresentare la musica in tutti i suoi aspetti legandoli tra loro attraverso la “main_table” che nel suo campo "file_xml" raccoglie il codice da cui vengono estratte tutte le informazioni utili.

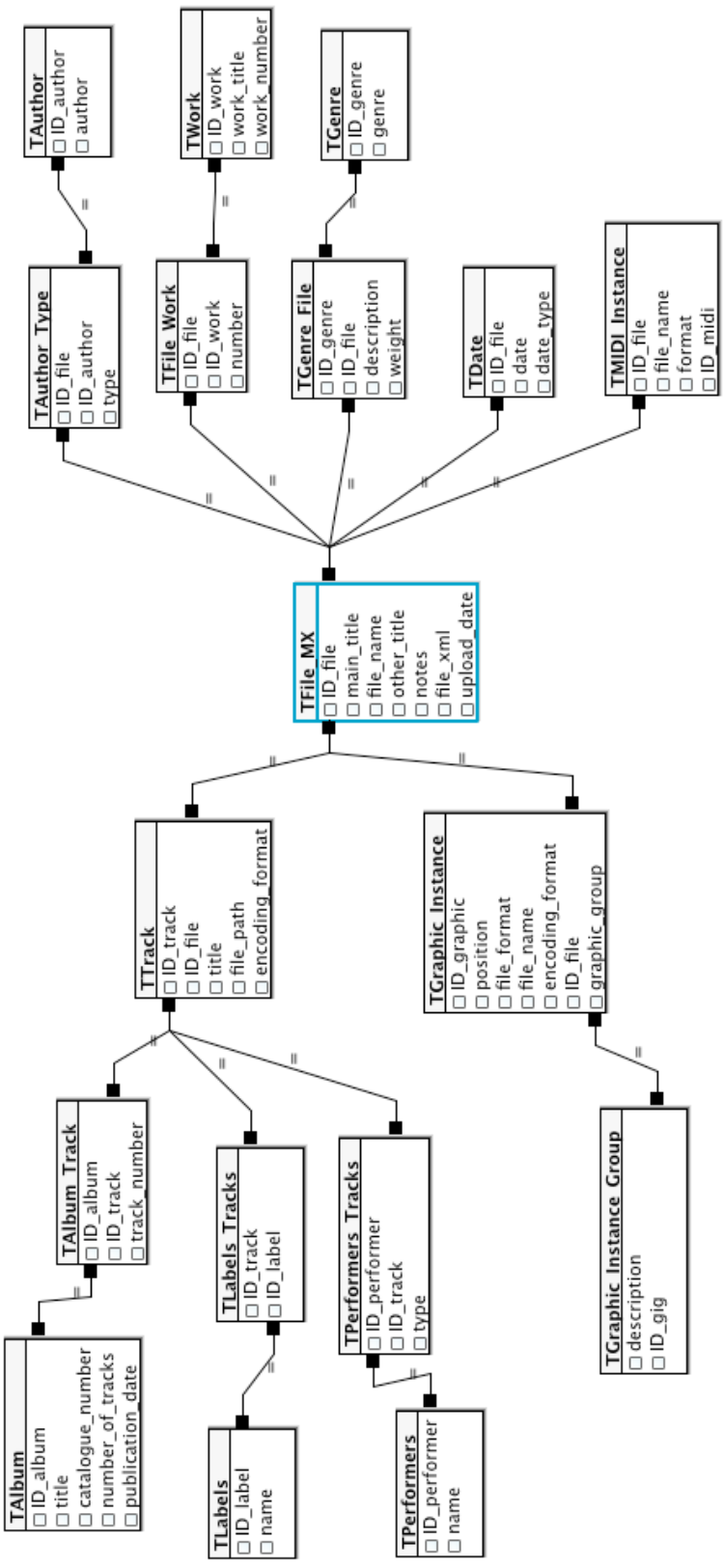

Fig. 16 – Visione completa del Repository

Capitolo 4

4.1 I trigger

In questo capitolo si tratterà delle procedure di automazione[9] di inserimento e di cancellazione scritte in linguaggio procedurale PL/PgSQL, tali procedure sono delle funzioni ideate per effettuare il parsing del documento XML, estrarne i contenuti utili e inserirli nelle corrispettive tabelle che li rappresentano.

Verranno esposti i procedimenti logici e le variabili utilizzate cercando di chiarire come e perché sono state effettuate alcune delle scelte che sono state operate in fase di scrittura.

4.2 Il Trigger di Inserimento

4.2.1 L'idea di automazione

PostgreSQL 8.4.2 fornisce alcuni strumenti per la scrittura avanzata di query ed è in grado di integrare diversi linguaggi di programmazione con il linguaggio SQL utile alla manipolazione ed al controllo dei dati depositati in una base di dati (come visto nel capitolo 2).

Di particolare interesse è stato, per il progetto, il linguaggio procedurale PL/pgSQL integrato in PostgreSQL e simile al PL/SQL di Oracle; con questo linguaggio è stato possibile far leggere al software i file XML che il Repository IEEE 1599 dovrà contenere, PL/pgSQL infatti possiede al suo interno la funzione XMLPARSE in grado di effettuare il parsing di un documento XML valido e scritto correttamente (in caso di errori nel codice il documento non viene riconosciuto e quindi non letto). Dopo averlo "parsato" diviene possibile inserire il codice XML estratto all'interno della base dati, più precisamente nell'apposito campo XML "file_xml" creato per contenere il codice del file standard IEEE1599.

Oltre alla funzione XMLPARSE, in PL/pgSQL è utilizzabile un'altra funzione indispensabile per automatizzare il Repository: Xpath[8]. Questa funzione come visto permette di selezionare un preciso elemento all'interno di un documento XML qualsiasi fornendone l'albero di appartenenza ed estrarne i contenuti o gli attributi. I dati salvati in un vettore di elementi XML possono essere usati per trasferire le informazioni nei campi desiderati tramite procedura come verrà illustrato meglio in seguito.

Uno dei concetti chiave del Repository risulta essere quindi la totale automazione del Repository stesso. Bisogna infatti richiede all'utente solo l'inserimento di un file XML musicale standard IEEE 1599 valido senza doverne specificar i meta-dati manualmente infatti se questi sono contenuti nel file il Repository completerà in automatico i campi relativi agli elementi presenti nell' XML sorgente.

Un procedimento analogo avverrà in fase di cancellazione o aggiornamento dove sarà la base dati stessa con gli appositi trigger ad eliminare i record rimasti nelle tabelle secondarie senza nessuna relazione e quindi inutili ai fini di ricerche e consultazioni.

4.2.2 Le variabili utilizzate

Nel linguaggio procedurale PL/pgSQL è possibile, similarmente ad un linguaggio di programmazione come C o Java, definire delle variabili da utilizzare per i propri fini, visibili all'interno della procedura e passabili se necessario ad altre procedure.

Per effettuare l'inserimento automatico di dati nel Repository IEEE 1599 è necesario innanzitutto effettuare una INSERT sulla tabella “TFile_MX” nel campo “file_xml”;

```
INSERT INTO "TFile_MX" (file_xml) VALUES (XMLPARSE(document(E'<?xml version="1.0" encoding="UTF-8"?>....

….

</ieee1599>')));
```

Fig. 17 – Esempio di Insert di un documento XML IEEE 1599

Un codice di questo tipo scatenerà l'avvio della funzione di trigger Autoinsert(), la quale provvederà prima di tutto a salvare il contenuto del campo “file_xml” della tabella “TFile_MX” all'interno di una variabile di tipo XML chiamata “codice_xml”.

Questo passaggio evita continue connessioni al database tramite delle SELECT per richiedere un informazione facilmente salvabile come appunto il codice XML. Un altro dato indispensabile da immagazzinare è L'”ID_file” relativo al documento IEEE 1599 appena inserito; il dato è facilmente ottenibile grazie alla funzione currval('nome_sequenza'), già citata nel secondo

capitolo, che restituisce l'ultimo identificativo generato dalla sequenza legata alla chiave primaria selezionata, in questo caso “TFile_MX_ID_file_seq”, e lo salva all'interno della variabile “identifier”, ossia l'identificativo dell'ultimo inserimento appena effettuato di tipo biginteger.

Oltre a queste due variabili chiave si sono rivelate necessarie cinque variabili “result[N]” di tipo vettore XML (xml[]) per immagazzinare i contenuti di tag ed attributi richiesti.

Legate a queste sono presenti anche cinque variabili “container[N]” di tipo “text”, utili a immagazzinare il dato estratto dalla corrispettiva variabile “result[N]” effettuandone il casting a tipo testo. Le variabili container sono l'ultimo passo prima di inserire il dato nel campo relativo del Repository IEEE 1599.

Sono presenti inoltre alcune variabili di controllo per la scrittura di routines (lunghezza, lunghezza2, i, j, z) di tipo intero ed infine alcune variabili temporanee per il salvataggio di identificativi di tabelle secondarie utili a riempire le tabelle di relazione molti a molti.

In particolare l'ID temporaneo “IDTrack” che immagazzina l'identificativo univoco della traccia contenuta nella tabella “TTracks” per completare tutti i meta-dati del layer audio relativi alla singola traccia.

Infine per inserire il campo “weight” relativo al genere viene utilizzata una variabile temporanea di tipo reale, per via della natura del dato da trattare. Per svolgere il cast da XML a real però occorre un doppio passaggio di CAST, effettuando un passaggio da xml a testo e successivamente da testo a real per evitare problemi di codifica ed inserimenti errati.

```
 3 DECLARE
 4
 5 identifier bigint;       -- contiene l'ID del file appena inserito
 6 codice_xml xml;          -- contiene il codice XML del file appena inserito
 7
 8 --vettori XML ove vengono estratti i dati tramite Xpath
 9 result1 xml[];
10 result2 xml[];
11 result3 xml[];
12 result4 xml[];
13 result5 xml[];
14
15 --variabili di cast da XML a text dei dati estratti
16 container1 text;
17 container2 text;
18 container3 text;
19 container4 text;
20 container5 text;
21
22 -- contatori e quantità relative ai cicli
23 i integer;
24 j integer;
25 z integer;
26 lunghezza integer;
27 lunghezza2 integer;
28
29 --ID di tabelle secondarie temporanei
30 IDT bigint;
31 IDT2 bigint;
32 IDTrack bigint;
33
34 -- variabili di cast per tipi numerici real ed integer
35 t real;
36 temp1 int;
37 temp2 int;
```

Fig. 18 – Variabili utilizzate nella funzione Autoinsert()

4.2.3 Funzionamento ed esempi di codice

La funzione di trigger che provvede ad inserire i dati prelevati dal codice XML nelle tabelle opportune scatta nel momento in cui viene effettuata una INSERT all'interno della tabella principale “TFile_MX”, come già detto in precedenza. In particolare il plugin per il Framework IEEE 1599 genera una query di INSERT all'interno del campo “file_xml” e di “upload_date” passando come parametro il codice IEEE 1599, parsato attraverso XMLPARSE come documento. Questo tipo di parsing permette di mantenerne il tag <xml version> ed il DTD del Laboratorio di Informatica Musicale relativo allo standard IEEE 1599 che altrimenti andrebbero cancellati, perché interpretati dal campo xml. Viene inoltre viene inserita la data di caricamento del file tramite il plugin.

L'inserimento nella tabella “TFile_MX” fa partire la funzione Autoinsert() che si occupa in primo luogo di salvare l'identificativo univoco del file appena inserito attraverso la stringa:

```
SELECT currval('"TFile_MX_ID_file_seq"') INTO identifier;
```

Sfruttando l'ultimo valore generato dalla sequenza applicata alla chiave primaria della main table si memorizza all'interno di “identifier” l'identificativo del file appena inserito, utile a tutta la procedura di auto-inserimento.

Il secondo passo da svolgere è la memorizzazione del contenuto XML del documento da cui verranno successivamente estratte tutte le informazioni utili ed i meta-dati necessari:

```
SELECT "file_xml" INTO codice_xml FROM "TFile_MX" WHERE "ID_file" = identifier;
```

Utilizzando subito identifier come parametro di controllo si seleziona il record appena inserito e ne si copia il campo “file_xml” nella variabile XML appositamente creata.

Giunti a questo punto si hanno gli elementi per iniziare la vera e propria procedura che estrae i dati dal codice XML IEEE 1599 tramite la funzione Xpath:

```
SELECT INTO result1 xpath ('/ieee1599/general/description/main_title/text()', codice_xml);
SELECT INTO result2 xpath ('/ieee1599/general/notes/text()', codice_xml);
```

Come si può leggere dal codice sopra riportato la funzione xpath salva all'interno dei vettori result[N] ciò che trova nel tag selezionato dal primo dei suoi argomenti (il percorso tra apici) cercandolo all'interno del secondo elemento (la variabile codice_xml).

Nel caso del codice presentato qui sopra ed applicato su un codice XML di tipo:

```
<ieee1599 version="1.0" creator="Luca A. Ludovico">
 <general>
  <description>
   <main_title>Gottes Macht und Vorsehung</main_title>
    <author type="composer">Beethoven, Ludwig van</author>

   <number>5</number>
   <work_title>6 Geistliche Lieder Von Gellert</work_title>
  </description>
  <notes>Example</notes>
        ….
```

Fig. 19 – Esempio di XML IEEE 1599

i vettori conterranno i seguenti risultati:

Posizione	1
result1[1]	Gottes Macht und Vorsehung
result2[1]	Example

Fig. 20 – Rappresentazione del contenuto delle variabili result1 e result2 dopo l'esecuzione di Xpath

I Dati vengono poi passati alle variabili container effettuando il casting da XML a testo tramite le seguenti righe PL/pgSQL:

```
SELECT INTO container1 CAST (result1[1] AS text);
SELECT INTO container2 CAST (result2[1] AS text);
```

Ed infine i campi atti a rappresentare gli elementi estratti vengono riempiti effettuando un aggiornamento del record appena inserito per modificarne le colonne “main_title” e “notes” ancora vuote fino questo punto:

```
UPDATE "TFile_MX" SET main_title = container1 WHERE "ID_file" = identifier;
UPDATE "TFile_MX" SET notes = container2 WHERE "ID_file" = identifier;
```

Si inizia già a notare come un uso della variabile “identifier” risparmi, già dopo poche righe, due interrogazioni della “TFile_MX_ID_file_seq” evitando di contattare costantemente la base dati ad ogni inserimento da effettuare o controllo da eseguire.

Per questi due elementi presi in esempio da inserire nella tabella principale non vi sono dubbi; la loro quantità nello standard IEEE 1599 è definita. Il titolo principale e le note sono due elementi che sono presenti una sola volta nel codice XML ma, non sempre è così; nel momento in cui si analizzano i possibili autori questi possono essere più di uno e con ruoli diversi nella creazione

dell'opera. A questo proposito la funzione xpath è adatta comunque a svolgere l'estrazione dei dati. Se infatti, durante il parsing, la funzione trova per più volte il tag ricercato nel suo primo argomento essa andrà ad aggiungere posizioni al vettore contenente i dati di quell'elemento nell'ordine in cui sono scritti nel codice XML IEEE 1599. Nel caso di più autori come nell'esempio:

```
<ieee1599 version="1.0" creator="Luca A. Ludovico">
 <general>
  <description>
   <main_title>Gottes Macht und Vorsehung</main_title>
    <author type="composer">Beethoven, Ludwig van</author>
    <author type="poet">Christian Fürchtegott Gellert</author>
  </description>
 </general>
….
```

Fig. 21 – Esempio di XML IEEE 1599 con più autori

Si effettua l'estrazione dei dati utili ossia il contenuto del tag <author> e del suo attributo “type”:

```
SELECT INTO result1 xpath ('/ieee1599/general/description/author/text()', codice_xml);
SELECT INTO result2 xpath ('/ieee1599/general/description/author/@type', codice_xml);
```

Salvando autori e tipi nei due vettori XML è necessario utilizzare un ciclo per inserire questi dati nelle tabelle “TAuthor” e “TAuthor_Type”:

```
i = 1;
lunghezza := array_length(result1, 1);
```

Perciò prima si effettua un set dei contatori e del numero di clicli da eseguire in base al numero di elementi trovati (misurando il vettore con “array_lenght)) e poi si inizia la routine di casting ed inserimento:

```
WHILE i <= lunghezza
LOOP
	SELECT INTO container1 CAST (result1[i] AS text);
	SELECT INTO container2 CAST (result2[i] AS text);
	IDT2 = NULL;

	SELECT "ID_author" INTO IDT2 FROM "TAuthor" WHERE "author" = container1;

	IF (IDT2 IS NULL) THEN
		INSERT INTO "TAuthor" (author)
			VALUES (container1);
		SELECT "ID_author" INTO IDT FROM "Tauthor"
			WHERE "author" = container1;
		INSERT INTO "TAuthor_Type" ("ID_file", "ID_author", type)
			VALUES (identifier, IDT, container2);
	ELSE
		INSERT INTO "TAuthor_Type" ("ID_file", "ID_author", type) VALUES
(identifier, IDT2, container2);
	END IF;
	i = i+1;
END LOOP;
```

Fig. 22 – Esempio procedura per l'estrazione degli autori dal codice XML

L'inserimento di un nuovo autore viene effettuato solamente se la SELECT precedente all'operatore IF non trova un autore che abbia lo stesso nome di quello che si sta per inserire. Questo controllo serve ovviamente a non inserire record omonimi nel Repository IEEE 1599 evitando ridondanze inutili. Se non esistono autori che abbiano il nome uguale all'autore attuale, all'interno di “IDT2” verrà inserito il valore “NULL”. “NULL” eviterà, grazie alla condizione di IF, di eseguire il codice di inserimento sino all'istruzione ELSE successiva.

Questa istruzione si occupa di creare una relazione tra l'autore già presente “ID_Author”, inserito dalla SELECT all'interno “IDT2” ed il file appena inserito, tramite “identifier”. Si assegna nel mentre, alla relazione autore-brano, anche il tipo di autore; ossia il ruolo che l'autore ha avuto nella composizione di quel brano.

Analogamente si procede con l'inserimento delle opere e dei generi utilizzando lo stesso ragionamento, con la particolarità che il peso di un genere subisce il doppio CAST per diventare un dato numerico reale.

Le tabelle relative alle date ed alle istanze MIDI sono semplici tabelle di look-up pertanto per completarle si svolge l'inserimento in un ciclo per ovviare al problema di elementi multipli nello stesso file IEEE 1599. Ma per quanto riguarda queste istanze non ci sono altri particolari controlli da svolgere.

Leggermente diversa, nonostante debba sfruttare gli stessi meccanismi e funzioni, è la sottoprocedura che estrae ed inserisce i contenuti grafici descritti nel file XML musicale IEEE 1599 a causa della struttura XML con cui sono descritte le istanze grafiche (Vedi Figura 3).

Essendo ogni istanza grafica rappresentata in un elemento <graphic_instance> e quest'ultimo un tag figlio di <graphic_instance_group> è necessario strutturare l'estrazione partendo dall'elemento padre. Una volta estratto il suddetto elemento ed inserito nella tabella “TGraphic_Instance_Group” si procede con un nuovo ciclo annidato nel primo che svolge la routine di inserimento nella tabella relativa alle istanze grafiche “TGraphic_Instance” andando a prelevare gli attributi del tag in esame e assegnando al campo “graphic_group” l'identificativo del gruppo salvato precedentemente nella variabile “IDT”.

```
i = 1;
SELECT INTO result5 xpath
('/ieee1599/notational/graphic_instance_group/@description', codice_xml);

lunghezza := array_length(result5, 1);
WHILE i <= lunghezza
LOOP
        SELECT INTO container5 CAST (result5[i] AS text);
        SELECT "ID_gig" INTO IDT from "TGraphic_Instance_Group"
                WHERE description = container5;
        IF(IDT IS NULL)THEN
                INSERT INTO "TGraphic_Instance_Group" (description) VALUES (container5);
                IDT = currval('"TGraphic_Instance_Group_ID_gig_seq"');
        ELSE END IF;
        j = 1;

        SELECT INTO result1 xpath
('/ieee1599/notational/graphic_instance_group/graphic_instance/@file_name', codice_xml);
        SELECT INTO result2 xpath
('/ieee1599/notational/graphic_instance_group/graphic_instance/@file_format', codice_xml);
        SELECT INTO result3 xpath
('/ieee1599/notational/graphic_instance_group/graphic_instance/@encoding_format',
codice_xml);
        SELECT INTO result4 xpath
('/ieee1599/notational/graphic_instance_group/graphic_instance/@position_in_group',
codice_xml);

        lunghezza2 := array_length(result1, 1);
        WHILE j <= lunghezza2
        LOOP
                SELECT INTO container1 CAST (result1[j] AS text);
                SELECT INTO container2 CAST (result2[j] AS text);
                SELECT INTO container3 CAST (result3[j] AS text);
                SELECT INTO container4 CAST (result4[j] AS text);

                INSERT INTO "TGraphic_Instance" ("ID_file", file_name, file_format,
```

```
encoding_format, position, graphic_group)
                        VALUES (identifier, container1, container3, container3, container4, IDT);
                j = j+1;
        END LOOP;
        i = i+1;
END LOOP;
```

Fig. 23 – Procedura con doppio ciclo di estrazione usata per le Istanze Grafiche

Infine, si andrà a trattare brevemente l'ultima parte mancante del Repository IEEE 1599; quella cioè relativa al layer audio che come visto in precedenza è composta dalle tracce audio a cui sono connesse informazioni e meta-dati relativi alla singola traccia.

In primo luogo verrà analizzata la tabella relativa appunto alle tracce “TTrack”, la quale è direttamente connessa alla main table tramite un campo al suo interno di tipo biginteger. A questo campo viene assegnato l'identificativo del codice IEEE 1599 appena inserito.

Sono presenti inoltre i valori estratti dagli attributi “file_name” ed “encoding_format” del tag <track>. Completata così la tabella che rappresenta la traccia si procede a salvare l'“ID_track” appena generato, esattamente come come si è fatto per l'identificativo del brano, tramite sequenza all'interno di “IDTrack”.

```
        SELECT currval('"TTrack_ID_track_seq"') INTO IDTrack;
```

Sfruttando così l'uso delle sequenze è possibile utilizzare la variabile “IDTrack” esattamente come la variabile “identifier” creando le relazioni con i meta-dati relativi alla traccia audio d'interesse. Assicurato questo passaggio fondamentale è possibile procedere al completamento delle tabelle “TAlbum”, “TLabels” e “TPerformers” e delle loro tabelle di relazione alla tabella delle tracce come effettuato per le opere e gli autori. In questo modo si garantiscono relazioni multiple in tutti i sensi (molti a molti) e si fornisce integrità ai dati inseriti anche a livello di strato audio. Strato in cui tutte le relazioni tra album, etichette e musicisti (performers) possono essere di tipo molti a molti.

4.2.4 Aspetti non trattati

In ultima analisi si dirà che la procedura garantisce il riempimento dei campi del Repository IEEE 1599 con i relativi meta-dati inseriti nel file XML musicale standard IEEE 1599 analizzandone gli aspetti dove ha senso effettuare delle ricerche.

Alcuni di questi non sono stati trattati perché ritenuti inadatti ai fini di ricerca. La base dati risulta comunque espandibile per trattare anche possibili archiviazioni future come il trattamento di altri layer, ad esempio con trigger che siano in grado di estrarre dati come il tempo o la tonalità.

La tecnologia della funzione xpath e la struttura dello standard IEEE 1599 attuali non permettono un estrazione corretta della tonalità di un brano di questo tipo.

4.3 Il Trigger di Cancellazione

4.3.1 L'idea di Cancellazione Automatizzata

L'idea di automazione non ha interessato solamente l'ambito dell'aggiunta di nuovi file al Repository IEEE 1599 ma, si è pensato anche a cosa dovesse avvenire in caso di cancellazione di un file oppure di aggiornamento di un file già presente sul Repository.

Da subito si è pensato che le informazioni che sono legate solamente al brano che si desidera eliminare o aggiornare debbano essere eliminate o aggiornate con il brano stesso. Di contro invece informazioni comuni a più brani archiviati sulla base dati hanno senso di mantenersi sul Repository IEEE 1599 nel caso in cui almeno uno dei brani a cui esse sono legate rimanga in archivio.

Se si procedesse senza valutare queste condizioni anche dopo una piccola serie di inserimenti, aggiornamenti e cancellazioni la base dati avrebbe record ridondanti al suo interno, record di tabelle secondarie senza relazioni con la tabella centrale e quindi inutili perché non rappresentativi di nessun brano archiviato.

Ad esempio mantenere un autore di cui è stato cancellato l'unico brano archiviato è scorretto data la centralità del brano e non quella dell'autore sul Repository, inoltre non è di nessuna utilità sapere che che un autore o un opera esistono ma non sapere altro di queste informazioni.

Per operare queste automazioni sono molto utili le funzioni applicabili tramite il linguaggio SQL durante la creazione delle tabelle, in particolare la DELETE ON CASCADE.

Come si vede ad esempio nella tabella “TAuthor_Type”:

```
CREATE TABLE "TAuthor_Type"
(
  "ID_file" bigint NOT NULL,
  "ID_author" bigint NOT NULL,
  "type" text,
  CONSTRAINT "PKTIPO" PRIMARY KEY ("ID_file", "ID_author"),
  CONSTRAINT "TAuthor_Type_ID_file_fkey" FOREIGN KEY ("ID_file")
      REFERENCES "TFile_MX" ("ID_file") MATCH SIMPLE
      ON UPDATE NO ACTION ON DELETE CASCADE
);
```

Fig. 24 – Codice di creazione della tabella autori con clausula di ON DELETE CASCADE

Il solo SQL però non è sufficiente perché applicabile solamente alle tabelle direttamente connesse a “TFile_MX”, che è la tabella sulla quale deve avvenire la cancellazione per rendere il tutto automatizzato. Pertanto alcune tabelle non verranno automaticamente controllate e cancellate dal lato server del Repository IEEE 1599 ed in particolare occorre andare ad intervenire con un apposito trigger contente procedure di controllo e cancellazione per “TAuthor”, “TWork”, “TGenre”, “TAlbum”, “TLabels”, “TPerformers” e “TGraphic_Instance_Group” al fine di eliminare eventuali record che non possiedono più relazioni dopo la cancellazione del brano selezionato.

L'operazione di cancellazione effettuata senza l'uso di un trigger non permetterebbe di verificare se sono presenti tuple isolate in tabelle relazionate a TFile_MX tramite tabelle di appoggio già aggiornate attraverso la DELETE ON CASCADE; per questo si è proceduto in maniera differente dal lanciare una query di DELETE semplice ma ci si è avvalsi di una tabella di appoggio in cui viene eseguita una INSERT tramite il Plugin del Framework e con la quale viene passato alla tabella “TDelete” l'identificativo del brano da cancellare:

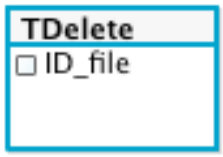	INSERT INTO "TDelete" (“ID_file”) VALUES ('variabile passata dal plugin');

Fig. 25 – Tabella di cancellazione e codice di attivazione del trigger

Il salvataggio temporaneo dell' “ID_file” ha il duplice vantaggio di poter far partire il trigger a connessione ormai chiusa con il client e permette di prelevare prima di cancellarlo il campo “file_xml” dalla main table, campo utile alle procedure di eliminazione delle tabelle sopracitate.

4.3.2 Le Variabili Utilizzate

Come per il trigger di inserimento si è fatto uso di variabili d'appoggio per eseguire i cicli di controllo e memorizzare le informazioni utili.

Di particolare rilievo come per l'inserimento sono la variabile “identifier” che contiene l'identificativo univoco del file da eliminare e la variabile “codice_xml”, la quale contiene invece il codice su cui la funzione xpath eseguirà l'estrazione di informazioni utili ai controlli pre-eliminazione.

Sono anche qui presenti le variabili temporanee “container1” e “result1” rispettivamente di tipo testo e []xml per garantire l'estrazione da parte di xpath e il cast a “text” dei parametri di controllo. Sono presenti infine variabili utili alla generazione di routine (i, lunghezza) e identificativi temporanei (IDT1 e IDT2).

4.3.3 Funzionamento ed esempi di codice

Come già spiegato la cancellazione sfrutta una INSERT sulla tabella d'appoggio “TDelete” per fare partire la procedura di cancellazione delete() infatti il codice generatore del trigger si presenta in questo modo:

```
CREATE TRIGGER delete() AFTER INSERT ON "TDelete"
FOR EACH ROW EXECUTE PROCEDURE delete();
```

Richiamando la procedura ON INSERT su ogni riga della tabella appositamente creata.

Le prime operazioni effettuate sono analoghe a quelle di inserimento e prevedono la memorizzazione di “ID_file” e del codice contenuto in “file_xml”. Raccolte queste due informazioni si procede ad eliminare il record, selezionato tramite l'identificativo univoco dalla tabella “TFile_MX”, e il dato inserito per l'eliminazione dalla tabella “TDelete”. In modo non è necessario sfruttare le sequenze per recuperarlo ad ogni cancellazione, inoltre perché non vi è motivo per archiviare un dato non connesso a nulla e di puro scopo procedurale.

```
DELETE FROM "TFile_MX" WHERE "ID_file" = identifier;
DELETE FROM "TDelete";
```

Eliminando con la prima delle due righe il record dalla tabella principale il database stesso grazie alle DELETE ON CASCADE inserite nelle tabelle direttamente connesse a “TFile_MX” elimina i record connessi ad essa nelle tabelle sopracitate e usate per le relazioni molti a molti. Si libera così delle informazioni inutili gran parte del Repository IEEE 1599 ma, per verificare che non ci siano informazioni inutili residue (senza relazioni con la tabella principale), è necessario controllare e pulire le tabelle rimanenti con un codice controllo ed eliminazione.

Codice inserito appunto nel trigger di cancellazione:

```
i = 1;
SELECT INTO result1 xpath ('/ieee1599/general/description/author/text()', codice_xml);
lunghezza := array_length(result1, 1);


WHILE i <= lunghezza
LOOP
        SELECT INTO container1 CAST (result1[i] AS text);
        IDT2=NULL;
        SELECT "ID_author" INTO IDT FROM "TAuthor" WHERE author = container1;
        SELECT "ID_author" INTO  IDT2 FROM "TAuthor_Type" WHERE "ID_author" = IDT;
        IF (IDT2 IS NULL)THEN
        DELETE FROM "TAuthor" WHERE "ID_author" = IDT;
        END IF;
        i = i+1;
END LOOP;
```

Fig. 26 – Esempio di procedura di cancellazione

La routine di esempio effettua un parsing del codice eliminato e ne estrae gli eventuali autori all'interno della variabile result1, la quale viene letta in ogni posizione tramite il loop e di volta in volta estratto uno degli autori presenti. Tramite questo dato testuale attraverso una SELECT di controllo si può prelevare l' ”ID_author” dalla tabella degli autori e verificare tramite una seconda SELECT di controllo, che inserisce in IDT2 l'id dell'autore stesso prelevato però dalla tabella di relazione con “TFile_MX”. Se l'autore infatti non ha più relazioni attive verso questa tabella IDT2 avrà valore “NULL” dando accesso al ramo THEN della condizione IF subito successiva all'assegnamento dove, come è possibile leggere il ramo THEN del codice, è prevista la cancellazione dell'autore appena analizzato.

Se invece il codice:

```
SELECT "ID_author" INTO  IDT2 FROM "TAuthor_Type" WHERE "ID_author" = IDT;
```

Restituisce un valore identificativo non viene effettuato nulla passando direttamente all'istruzione END IF. Questo è il caso in cui l'autore in questione ha composto almeno un altra opera e non va quindi eliminato dalla base dati.

La struttura della procedura qui sopra descritta è analoga anche per tutte le tabelle da analizzare e per cui potrebbe essere necessario eliminare dei record.

In questo modo si mantiene sempre integrità in tutte le tabelle e non appena un file viene rimosso dalla base dati si ha la certezza che, da subito e senza effettuare ulteriori controlli, i dati saranno tutti congruenti e connessi alla tabella principale.

4.3.4 Aspetti non trattati

Per quanto riguarda la procedura di cancellazione non è stato necessario trattare l'aspetto di controllo e pulizia del database tramite interfaccia web in quanto non previsto dalle specifiche di sviluppo del Repository.

Inoltre inizialmente si era pensato ad una procedura automatica che effettuasse una scansione e controllo su tutte le tabelle secondarie per rilevare ed eliminare eventuali tuple non relazionate in qualsiasi momento ma, con lo sviluppo della procedura di cancellazione attivata tramite il trigger, per ogni eliminazione o aggiornamento si effettua già la procedura di scansione e pulizia garantendo integrità al Repository IEEE 1599.

4.4 Una soluzione per l'aggiornamento

Si è presa in esame, oltre ad inserimento e cancellazione, anche la possibilità di effettuare una modifica oppure un aggiornamento di un qualsiasi file IEEE 1599; se ad esempio si devono aggiungere informazioni trovate a posteriori o correggere eventuali errori nel codice XML.

Si è pensato in un primo momento ad un trigger di inserimento che si attivasse anche ON UPDATE sulla tabella “TFile_MX” ma questa soluzione non è praticabile per motivi logici e lascerebbe archiviati sul Repository dati (quelli vecchi) non coerenti. Andrebbe prima svolta una adatta procedura di controllo e cancellazione.

Valutato questo aspetto si poteva procedere scrivendo una funzione di UPDATE come precedentemente fatto per INSERT e DELETE. Una procedura che utilizzasse contemporaneamente parti delle funzioni di DELTE e di INSERT da attivare con un trigger apposito per effettuare l'aggiornamento di un file. Questa soluzione andrebbe a generare un codice abbastanza prolisso e con facile presenza di errori.

Perciò, si è pensato di effettuare, tramite il plugin integrato nel Framework IEEE 1599, una funzione che permetta di effettuare aggiornamenti sfruttando le procedure esistenti.

Tramite il framework quindi viene selezionato il file archiviato sul Repository da aggiornare e generata una finestra di confronto tra il vecchio codice XML ed il codice da inserire.

Il primo viene prelevato tramite una SELECT sul Repository IEEE 1599 mentre il secondo è letto direttamente dal Framework IEEE 1599.

A questo punto, nel caso sia necessario, viene eseguita la sostituzione attraverso l'uso del trigger di cancellazione, applicato sul file presente nel Repository e continuando poi l'aggiornamento con una normale procedura di INSERT dove viene inviato al Repository il codice già aperto e confrontato nel Framework IEEE 1599.

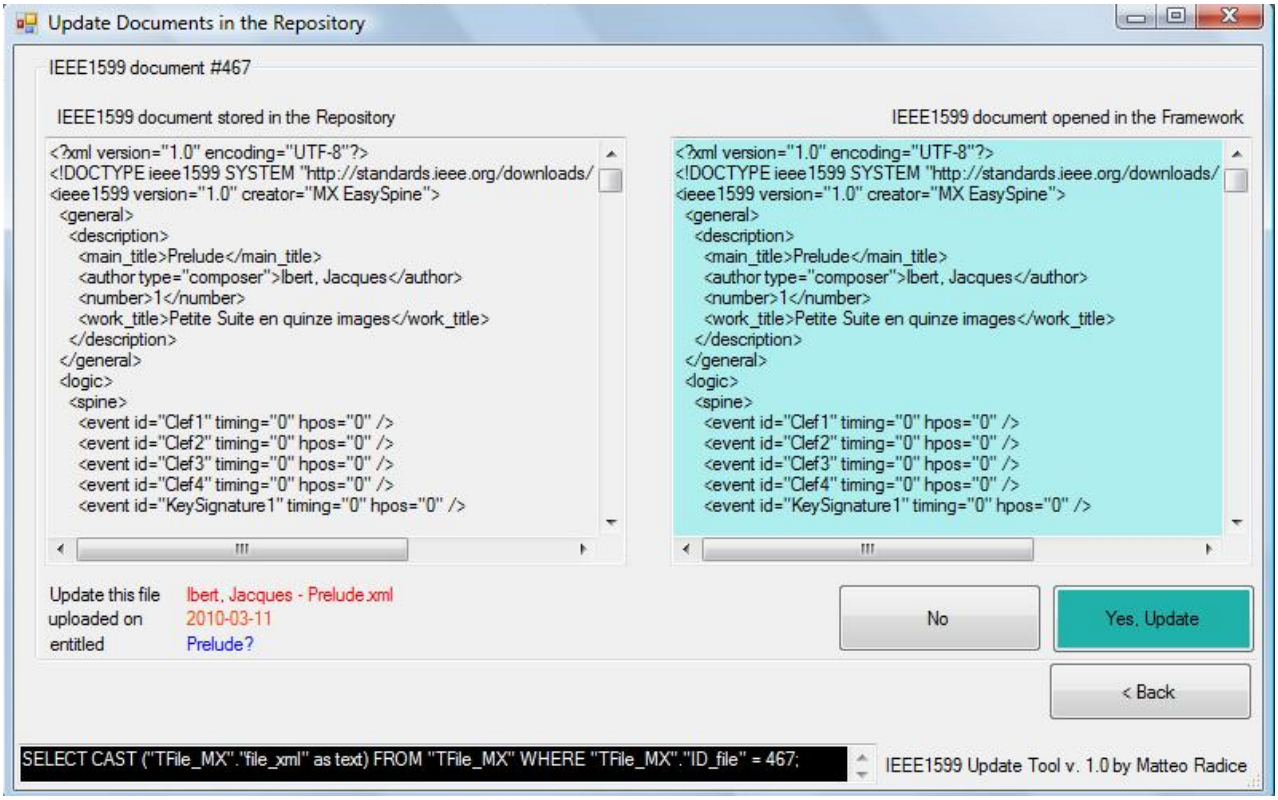

Fig. 27 – Schermata di Update del Framework IEEE 1599

Indice delle Figure

Riferimenti Bibliografici

1. Luca A. Ludovico, “Key concepts of the IEEE 1599 Standard”, Proceedings of the IEEE CS Conference The Use of Symbols To Represent Music And Multimedia Objects, IEEE CS, Lugano, Switzerland, 2008.

2. Adriano Baratè, Goffredo Haus, Luca A. Ludovico, “Music representation of score, sound, MIDI, structure and metadata all integrated in a single multilayer environment based on XML”*, Intelligent Music Information Systems: Tools and Methodologies*, ed. Jialie Shen, John Shepherd, Bin Cui and Ling Liu, Idea Group Reference, 2006.

3. Luca A. Ludovico, “Manuale di MX”, LIM – Laboratorio di Informatica Musicale / DICo – Dipartimento di Informatica e Comunicazione, Febbraio 2005.

4. David Marco, “Building and Managing the Meta Data Repository: A Full Lifecycle Guide” pubblicato da: “Wiley” nel luglio 2000.

5. Alan Beaulieu, “Learning SQL” pubblicato da: “O'Reilly Media” nell'aprile 2009.

6. The PostgreSQL Global Development Group, “The PostgreSQL Reference Manual Volume 1: SQL Language Reference” pubblicato da “Network Theory LTD” nel 2007

7. The PostgreSQL Global Development Group, “The PostgreSQL Reference Manual Volume 2: Programming Guide” pubblicato da “Network Theory LTD” nel 2007

8. John Simpson, “Xpath and Xpointer: Locating Content in XML Documents” pubblicato da: “O'Reilly Media” nel luglio 2002.

9. The PostgreSQL Global Development Group, “The PostgreSQL Reference Manual Volume 3: Server Administration Guide”pubblicato da “Network Theory LTD” nel 2007.

Ringraziamenti

Intendo ringraziare innanzitutto i miei Genitori, per il supporto, la pazienza e la disponibilità.

I compagni con cui ho condiviso l'esperienza universitaria ed in modo particolare Giovanni Sala e Matteo Radice con cui è stato sviluppato progetto del Repository.

Ringrazio inoltre Goffredo Haus ed il Laboratorio di Informatica Musicale, per la disponibilità e l'attenzione che ha posto al progetto, con loro tutto il Dipartimento di Informatica e Comunicazione e l'Università degli Studi di Milano.

Voglio ringraziare tutti i miei amici, e coloro che mi sono stati accanto durante questi tre anni e durante la scrittura dalla tesi tra cui: Gabriele Sala, Luca Ferrari, Giulia Paggiarin, Andrea Guardigli, Francesca Clerici, Stefania Rendina, i Mnemosyne e soprattutto Claudia Camporese che mi ha sostenuto e motivato particolarmente in questi ultimi mesi.