ScoreFlow. Aplicación web para validar sistemas de reconocimiento de partituras musicales(OMR)

Marc Martín Martínez

Resumen– Este proyecto surge de la necesidad de corregir partituras musicales en formato MusicXML creadas a partir de *Optical Music Recognition* (OMR). El usuario sube a la aplicación un archivo MusicXML junto con una o varias imágenes de la partitura que usará como *ground truth* para revisar la pieza musical. A continuación, se muestra el grupo de imágenes que se hayan subido junto con un conjunto de compases establecidos por el archivo MusicXML. Se puede puede navegar libremente entre las diferentes imágenes que haya subido y los diferentes grupos de compases existentes. Usando una barra de herramientas, el usuario puede añadir, cambiar o eliminar: notas, accidentes, silencios... y cambiar características del compás: armadura, clave y signatura de tiempo. Finalmente, cuando se haya terminado de corregir la partitura, el usuario puede descargarse un nuevo archivo MusicXML con todos los cambios que haya hecho previamente.
Palabras clave– Partitura, OMR, MusicXML, NodeJS, Vue.js

Abstract– This project comes up from the necessity of correcting musical scores in MusicXML format that were created from OMR *Optical Music Recognition*. The user uploads to the application a MusicXML file along with one or more images of the score that will be used as a ground truth to review the musical piece. Then, the group of images that have been uploaded are shown along with a set of measures that are established by the MusicXML file.You can freely navigate between the different images that were uploaded and the set of measures that exist. Using a toolbar, the user can add, modify or remove: notes, accidentals, rests... and change characteristics of the measure: key signature, clef and time signature. Finally, when the score is corrected, the user can download a new MusicXML file with all the changes that were made.

Keywords– Score, OMR, MusicXML, NodeJS, Vue.js

1 Introducción

DURANTE la década de los 50 se funda la primera compañía dedicada al desarrollo y venta de dispositivos OCR *(Optical Character Recognition)*, creada por David H. Shepard[1], desarrollador del primer dispositivo OCR. Con el paso de los años surge un nuevo concepto de reconocimiento de carácteres, OMR (Optical Music Recognition), dedicado obviamente a la lectura de elementos musicales. La primera aplicación desarrollada es MIDISCAN, por *Musitek Corporation*[2]. Actualmente, aunque la tecnología de OMR ha avanzado, aún existen errores de lectura por parte del software de reconocimiento que da lugar a resultados incorrectos. De ahí la importancia del desarrollo de este proyecto. Aunque no dedicado exclusivamente a corregir archivos creados a partir de OMR, existen otras aplicaciones web de creación de partituras *float.io*[3] y *Noteflight*[4].

• E-mail de contacto: marcmartin315@gmail.com
• Mención realizada: Enginyeria del Software
• Trabajo tutorizado por: Alicia Fornes Bisquerra (Ciencias de la Computación)
• Curso 2019/20

2 Estado del Arte

3 Objetivos

En esta sección se explicarán los objetivos tenidos en cuenta para poder realizar este proyecto:

- Crear entorno donde el usuario puede subir un fichero MusicXML y una o varias imágenes de la partitura.

–Falta gestion archivos/usuarios–

- Visualización de la partitura.
  A partir de la API *VexFlow*[6], representar un conjunto de compases respectivos al fichero MusicXML. El espacio asignado a los compases en la página se adapta al tamaño de la pantalla del navegador.

- Edición de la partitura.
  Usando *Vue.js*[7] se crea una interfaz de usuario que permite añadir, cambiar o eliminar elementos de la partitura.

- Exportar nuevo archivo MusicXML.
  Una vez el usuario considere oportuno, se puede descargar un nuevo archivo con todos los cambios hechos previamente.

4 Planificación

La planificación del proyecto puede dividirse en 3 fases.

4.1. Introducción al tema

Esta fase consistía en hacer un primer contacto con la materia.

- Funcionamiento de MusicXML.
  Estudio del formato universal de representación de partituras. Al tratarse de un archivo XML, todo funciona con tags.

- Creación de glosario.
  Documento con un listado de elementos musicales básicos y sus descripciones correspondientes. Útil en las primeras fases del proyecto donde aún no se dominaba completamente los conocimientos musicales requeridos para crear partituras coherentes.

- Comprobar el estado del arte.
  Estudio de cual era la situación actual referente al OCR y OMR, además de aplicaciones similares al proyecto.

4.2. Análisis y Diseño

La meta de esta fase era identificar las funciones básicas que necesitaba el proyecto y como implementar estas de la manera mas óptima posible.

4.2.1. Análisis de Requisitos

Listar y clasificar los requisitos en funcionales y no funcionales. Después, usando el modelo de Kano, clasificarlos en *dissatisfiers, satifiers y delighters.*

4.2.2. Diseño

Creación de elementos considerados útiles para poder hacer un buen diseño de la aplicación.

- Diagramas UML
  Creación de diagramas de casos de uso, de secuencia y de clase. Utilización de *VisualParadigm* [8] para componer estos diagramas. – Diagramas en apéndice?–

- Prototipo
  Creación de un prototipo simple para representar el editor de partituras usando la herramienta de edición de imágenes digitales *GIMP*[9]. Este prototipo sería desechado mas adelante, ya que se decidiría usar una barra de herramientas para la edición de la partitura.

- Persona
  Desarrollo de una *persona* para facilitar que la interfaz del usuario estuviera enfocada en un posible usuario de la aplicación, y no en el desarrollador de la aplicación. Utilización de la herramienta *Xtensio*[10] para crear la persona.

4.3. Implementación

La planificación inicial de esta fase consistía en visualizar la partitura y después poder editarla, para finalmente exportar el nuevo archivo MusicXML. Todo esto siguiendo el paradigma de testing *Test-Driven Development*. Una vez acabado esto se esperaba tener hasta 3 iteraciones de una semana de duración cada una. En estas iteraciones se detectarían errores mediante *user testing* con usuarios con conocimientos musicales para después modificar requisitos y/o el diseño, e implementar estos cambios. Esta fase sufrió algunos cambios con respecto lo planificado al principio del proyecto por una planificación demasiado optimista, no teniendo suficiente tiempo para las iteraciones. Además, la parte de testing del proyecto también se tuvo que omitir.

5 Metodología

La metodología a seguir del proyecto era iterativa. A medida que el proyecto avanzaba en el tiempo se iban modificando elementos del mismo, como por ejemplo el prototipo anteriormente mencionado. El diseño del programa también iba sufriendo pequeños cambios, al ver que surgían nuevas necesidades en la interfaz de usuario o se detectaban características obsoletas o daban lugar a dudas.

6 Resultados

La aplicación esta escrita en Nodejs. Esta esta dividida en *backend*, la parte del servidor, y *frontend*, la del cliente. –revisar–

6.1. Backend

En este apartado se mencionan las funcionalidades desarrolladas en el servidor y su funcionamiento. Se utilizan las extensiones:

- *Express* [11]. Framework para crear el servidor y hacer *routing*.

- *Express-fileupload* [12], Modulo para *express* para la subida de archivos del clienta al servidor.

El proposito principal del servidor es gestionar los archivos subidos por el usuario y servirlos cuando sea necesario al cliente.

6.2. Frontend

Esta sección muestra los elementos creados para el cliente.

6.2.1. Visualización de la partitura

Se crea la clase **XMLParser**, responsable de representar la partitura a partir del archivo MusicXML y la de aplicar los cambios a este hechos por el usuario.
Inicialmente se lanza una petición al servidor para obtener el archivo MusicXML del usuario utilizando un *fetch*. Después, se parsea todo el documento para extraer los tags necesarios para crear una partitura. Dado que es un XML, se utiliza la interfaz *Document* [13] para poder modificar el árbol *DOM*. La información obtenida es almacenada en otras dos clases:

- **Attributes.** Contiene la información de los tags del compás: clave, signatura de tiempo y armadura. Esta clase solo puede tener una instancia. El tag de atributos ha de estar presente en el primer compás de la pieza. Actualmente, la aplicación no es capaz de identificar cambios en los atributos en mitad de la partitura, por lo que solo se tendrán en cuenta los atributos del primer compás.
- **Note.** Contiene la información de los tags de una nota: duración, tono y si es o no un silencio. Además de los elementos que se le pueden añadir a una nota: accidentes, *legatos* y ligaduras y prolongaciones (puntillo, calderón).

Para gestionar el funcionamiento de los *legatos* y ligaduras, que son representados de la misma forma dentro de la aplicación, se crea la clase **TieGenerator**. Su función básica es almacenar la primera y la última nota que forma una ligadura. Una vez se tienen las dos notas, crea un objeto de VexFlow que representa dicha ligadura. Además, controla si la clase XMLParser ha de mostrar un compás adicional en el caso de que la nota inicial y la final se encuentren en dos compases distintos. Actualmente la aplicación solo puede mostrar correctamente una ligadura si el número de compases entre los que se encuentra el inicio y final de una ligadura es de uno.
Para poder representar toda esta información en forma de partitura se utiliza la API VexFlow. Para poder construir la partitura, esta información ha de ser *"traducida"* para que la API pueda utilizarla. Para ello se crea otra clase, **MusicXMLToVexFlow**. En ella se compara cada tag con todas las posibles opciones que puede tener el tag que este disponibles en la API y devuelve ese equivalente.
Finalmente, se inicializan los valores necesarios para VexFlow: tamaño de los compases, del SVG(Scalable Vector Graphics) donde se encontrará la partitura... y se crean los objetos de VexFlow para después dibujarlos y así conseguir representar la partitura. La aplicación representa hasta 5 compases a la vez, 6 en el caso de una ligadura con notas en diferentes compases.
A cada objeto que representa una nota se le asignan los atributos *measure y note*, que representan el número compás donde se encuentra la nota y el índice de la nota dentro de ese compás. Además de *event listeners* a *clicks*. Esta información será usada mas adelante cuando el usuario quiera modificar una nota.
Existen algunos elementos que la aplicación no es capaz de representar por que son poco utilizados. –Apéndice?–

Claves:

- Bass Clef (Down Octave)
- Vocal Tenor Clef

Armaduras fuera de la escala diatónica:

- Dorian
- Mixolydian
- Locrian
- Phrygian
- Aeolian
- Lydian
- Ionian

Armaduras teóricas:

- F♭
- G#
- d♭
- e#

Duración:

- Máxima
- Cuadrada
- Semigarrapatea
- Longa
- Garrapatea

6.2.2. Interfaz de Usuario

Para crear una interfaz de usuario reactiva a los cambios hechos por el usuario se utiliza el framework *Vue*. La interfaz se compone de: barra de herramientas, imagen o imágenes de la partitura y el navegador de partituras, que consiste en dos botones para avanzar o retroceder en el grupo de compases. Cada elemento de la interfaz de usuario se representa en componentes de *Vue*.
El componente *nav-bar*, o la barra de herramientas.Este componente permite al usuario modificar la nota *clickada* anteriormente o los atributos del compás. La tonalidad de una nota puede cambiarse utilizando uno de los botones que designa un tono mas la octava que se quiera utilizar. Los accidentes y prolongaciones pueden añadir o quitarse. La duración puede modificarse desde la redonda, 4 pulsos, hasta la semifusa, $\frac{1}{64}$ pulsos. En el compás puede modificarse la clave, la signatura del compás y la armadura. Además, la barra cuenta con dos modos de visualización de sus elementos: notación musical latina y anglosajona, para que el usuario pueda usar la notación con la que se sienta mas cómodo. La aplicación funciona internamente con la notación anglosajona.
Para poder aplicar los cambios que haga el usuario al archivo MusicXML se ha de volver a traducir la información transmitida por el usuario al formato MusicXML. Para ello se crea otra clase, **AlphabeticToMusicXML**, muy similar a la anteriormente mencionada MusicXMLToVexFlow.
Para visualizar las imágenes subidas por el usuario se cuenta con otro componente que cuenta con dos botones para avanzar o retroceder en las diferentes imágenes, y la imagen que quiera visualizar el usuario. La imagen seleccionada puede ser ampliada para ayudar al usuario a leerla mejor. Finalmente se muestra el conjunto de compases mas otros dos botones para poder avanzar o retroceder a lo largo de la partitura.

7 CONCLUSIONES

....

AGRADECIMIENTOS

... .

REFERENCIAS

[1] *David H. Shepard.* Character reading techniques. https://patents.justia.com/patent/4021777

[2] *Musitek Corporation.* SmartScore. https://www.musitek.com/

[3] *flat.io* https://flat.io

[4] *Noteflight.* https://www.noteflight.com/

[5] *Make Music.* MusicXML. https://www.musicxml.com/

[6] *Mohit Muthanna Cheppudira.* VexFlow www.vexflow.com

[7] *Evan You.* Vue.js https://vuejs.org/

[8] *Visual Paradigm.* https://www.visual-paradigm.com/

[9] *GIMP.* http://www.gimp.org/

[10] *Xtensio.* https://xtensio.com/

[11] *OpenJS Foundation.* Express. expressjs.com

[12] *Richard Girges and Roman Burunkov.* Express-fileupload. https://github.com/richardgirges/express-fileupload#readme

[13] Document. https://developer.mozilla.org/en-US/docs/Web/API/Document

APÉNDICE

A.1. Secció d'Apèndix

A.2. Secció d'Apèndix