# DESARROLLO DE UNA TIFLO-APLICACIÓN PARA TRADUCIR PARTITURAS MUSICALES A BRAILLE

DEVELOPMENT OF A TIFLO-APPLICATION FOR TRANSLATING MUSIC SCORES TO BRAILLE

**ÓSCAR DÍAZ RIBAGORDA**
**CLAUDIA GUERRERO GARCÍA-HERAS**
**LUCAS DE TORRE BARRIO**

DOBLE GRADO EN INGENIERÍA INFORMÁTICA Y MATEMÁTICAS. FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

---


Trabajo Fin de Grado en Informática

Junio 2020

Directores:
**María Guijarro Mata-García**
**Borja Manero Iglesias**

# Agradecimientos

Queremos empezar por agradecer a nuestros tutores María y Borja por ofrecernos la posibilidad de realizar un proyecto con implicaciones reales como este y de colaborar con una institución como la ONCE. Gracias además, por orientarnos durante el desarrollo de este trabajo y por instarnos a escribir nuestro primer artículo para el congreso ICCE 2020, lo que ha resultado ser una experiencia enriquecedora.

También queremos agradecer a la ONCE por su propuesta y su colaboración con el proyecto. En especial, a Pablo Carreño Gea por su gran ayuda en el desarrollo de la aplicación LiveDots.

Otra parte muy importante han sido los voluntarios que nos han prestado su tiempo probando la aplicación y rellenándonos las encuestas, muchas gracias.

Por último, agradecemos a nuestras familias por habernos apoyado y aguantado durante todos los meses en los que hemos estado trabajando en este proyecto, en especial durante los meses de confinamiento trabajando desde casa.

Y por su apoyo y compañía durante toda la carrera que cerramos con este proyecto, gracias, Manolo.

# Resumen

La música es una de las mayores formas de expresión que tenemos los seres humanos. Actualmente, la forma más común para representarla es el uso de partituras en tinta. Esta representación es completamente visual, por lo que una persona invidente no puede hacer uso de ella. Por ello, existe la notación llamada braille musical, que permite a las personas ciegas leer una partitura usando las manos a través de un papel perforado o de una línea braille.

Sin embargo, el uso y modificación de este tipo de partituras es más complicado que el de las partituras en tinta, algo que se pone de manifiesto en ámbitos como las clases de música o las orquestas. Con los instrumentos que hay hoy en día, modificar en tiempo real una partitura con notación braille es más complicado que hacerlo en una partitura en tinta. Esto se debe a que existen programas que modifican partituras en tinta fácilmente, mientras que los que modifican partituras en braille son más escasos y menos completos.

Existen programas, como MuseScore (Schweer, s.f.), que permiten modificar partituras y otros, como FreeDots (Repain y col., s.f.), que las traducen a notación braille. Aunque este sistema es útil, es evidentemente lento en el contexto de una clase. Sin embargo, en otros campos encontramos soluciones de editores interactivos en tiempo real, como es el caso de EDICO (Carenas y col., 2018), editor de matemáticas, física y química desarrollado por la ONCE ("ONCE", s.f.).

Del éxito de EDICO nació la idea de LiveDots, un editor musical interactivo en tiempo real. LiveDots es una aplicación accesible que utiliza al mismo tiempo la representación mediante partitura en tinta y mediante notación braille. Permite la modificación de la partitura en tinta, algo que cambia en tiempo real la notación braille. Además, es posible el uso de un revisor de pantalla para que vaya leyendo los elementos musicales de la partitura en notación braille.

Así, la idea de LiveDots es facilitar la docencia musical accesible. Por eso, realizamos un experimento a gente vidente para comprobar si LiveDots facilitaría la inclusión de alumnos invidentes en clases de música. Los participantes utilizaron la aplicación dos veces, la primera con los ojos tapados y la segunda sin tapar. De esta manera, probaron la parte desarrollada para gente invidente, que serían los estudiantes, y la parte desarrollada para gente vidente, que serían los profesores.

Durante el desarrollo de la aplicación, comprobamos las diferencias entre diseñar una

aplicación accesible y una no accesible, buscando en la primera que el uso sea posible sin necesidad de visualizar la pantalla. A partir de los resultados del experimento, comprobamos que LiveDots es una aplicación accesible que podría facilitar la inclusión de estudiantes invidentes en clases de música.

**Palabras clave**: accesibilidad, tiflotecnología, educación inclusiva, música, interactividad en tiempo real, WPF, música braille, JAWS, revisor de pantalla, aplicación de escritorio.

# Abstract

Music is one of the major forms of expression that the humankind has. In this day and age, the most common way of representing it is through sheet music. This representation is completely visual, therefore a blind person cannot make any use of it. For that reason, there exists the notation called musical braille, which allows blind people to read a music score using their hands on an embossed paper or a Braille line.

However, the use and modification of this type of scores are more complicated than that of the sheet music, something that is brought to light in situations like music classes or orchestras. A modification on a score with Braille notation is more complicated than a modification in a score in sheet music since the first one cannot be done in real-time.

There exists programs, like MuseScore (Schweer, s.f.), that allow to modificate sheet music scores and others, like FreeDots (Repain y col., s.f.), that translate them to Braille notation. Although this system is useful, it is obviously slow in the context of a class. However, in other fields we find solutions of real-time interactive Braille editors, like the case of EDICO (Carenas y col., 2018), a mathematics, physics and chemistry editor developed by the ONCE ("ONCE", s.f.).

From the success of EDICO was born the idea of LiveDots, an interactive musical Braille editor in real-time. LiveDots is an accessible application that uses at the same time the representation in sheet music and in Braille notation. It allows modification of the sheet music, which changes in real-time the braille notation. Moreover, it is possible to use a screen reader to read the musical elements of the score in braille notation.

That way, the idea of LiveDots is to make accessible musical education easier. For that reason, we made an experiment to non-blinded people to check if LiveDots would help with the inclusion of blind students in music classes. The participants used the application twice, the first time blindfolded and the second time without the blindfold. This way, they tested both the part developed for the blind people, which would be the students, and the part developed for the non-blinded people, which would be the teachers.

Throughout the development of the application, we verified the differences between designing an application that is accessible and one that is not, making sure that the use of the first did not require screen visualisation. Following the results of the experiment, we confirmed that LiveDots is an accessible application that could help with the inclusion of blind students in music classrooms.

**Keywords**: Accessibility, tiflotechnology, inclusive education, music, real time interactivity, Braille music, WPF, screen reader, JAWS, desktop application.

# Índice general

# Índice de figuras

# Capítulo 1

# Introducción

Desde el principio de los tiempos la música ha acompañado al ser humano ocupando un papel muy importante a lo largo de su historia. Es un elemento ubicuo en todas las culturas, una manifestación cultural universal que tiene el poder de comunicar sin importar barreras de tiempo, espacio o idioma (Wallin y col., 2001). Es por tanto un elemento muy poderoso.

Notación musical es el nombre genérico que se da a cualquier sistema de escritura utilizado para representar gráficamente una pieza de música (de Candé, 2002). Los primeros testimonios de partituras escritas usado notación musical en la cultura occidental están datados entre finales del siglo IV y comienzos del siglo III a. C. Burkholder y col., 2008. De hecho sabemos que la teoría musical se remonta aún más en el tiempo, la afinación pitagórica que es el sistema de construcción de la escala musical que da lugar al círculo de quintas se atribuye al filósofo Pitágoras en el siglo VI a. C. y se sigue usando en la actualidad (Benward y Saker, 2009). La capacidad de plasmar la música en papel nos permite compartirla y comunicarla a través del tiempo y el espacio.

El método de representación musical más extendido entre las personas ciegas es el braille musical, que es una adaptación del sistema de lectoescritura braille. El sistema braille se basa en diferentes combinaciones de puntos en relieve (Kent, 2012). El braille musical es una técnica de transcripción que permite representar cualquier partitura musical convencional con una notación precisa (de Candé, 2002). El conocimiento del braille musical dota al músico invidente no solo de una herramienta mediante la cual comprender y expresar la música sino de una concepción de la música muy distinta a la de una persona vidente acostumbrada a la representación gráfica en forma de pentagrama (Abramo y Pierce, 2013; Johnson, 2015).

La música es un elemento de unión, sin embargo en la educación musical aún hay una brecha entre videntes y personas con discapacidad visual. En muchos casos los alumnos invidentes no reciben la formación necesaria para comprender la notación braille musical y una gran parte de profesores videntes desconocen cómo facilitar el aprendizaje del estudiante como denuncia David Goldstein director del *National Resource Center for Blind Musicians* en Estados Unidos. Una alternativa que tienen los estudiantes ciegos para mejorar sus conocimientos musicales es asistir a una escuela para ciegos. En este escenario, los estudiantes

videntes e invidentes son enseñados usando diferentes metodologías que conducen a una dicotomía entre el braille musical y la escritura musical convencional. La falta de familiaridad con la escritura musical convencional hace más difícil que los estudiantes ciegos se incorporen más tarde en entornos con músicos videntes, por ejemplo un conservatorio o una orquesta (Goldstein, 2000).

Hay por tanto una necesidad de integrar a estudiantes videntes e invidentes en el mismo aula y enseñar a cada uno de ellos los conocimientos esenciales para desarrollar sus habilidades musicales (Quaglia, 2015; Buhagiar y Tanti, 2011). Es decir, queremos que los estudiantes ciegos puedan seguir una clase de música con estudiantes mayormente videntes mientras aprenden la notación braille musical y se familiarizan con la concepción musical de la música en tinta tradicional.

Algunas de las estrategias más utilizadas por los estudiantes ciegos o con discapacidad visual para facilitar la participación en el aula son: partituras en tinta ampliadas, que una tercera persona (compañeros de clase, padres o profesores) les lea la partitura y el uso de braille musical siempre que sea posible (Frederick, 2009; Smaligo, 1998). Todas estas herramientas necesitan la ayuda de una persona que transcriba la partitura oralmente o de un profesor que conozca el braille musical y se lo enseñe al alumno, lo cual no siempre es posible. Esto hace que el estudiante dependa de las personas que lo rodean.

Con el desarrollo de la tecnología tenemos en nuestras manos la posibilidad de crear herramientas que nos ayuden a integrar la concepción musical de videntes e invidentes, facilitando así aspectos como la integración de alumnos invidentes y videntes en clases de música o la colaboración de músicos videntes e invidentes en una orquesta. Ya hay programas diseñados para que personas con discapacidad visual puedan visualizar y editar música en notación braille (Homenda, 2008), estudios sobre cómo enseñar braille musical a niños mediante aplicaciones de ordenador (Borges y Tomé, 2014) y programas que traducen partituras de música en tinta a braille y viceversa.

El avance de la tecnología en los últimos años nos ha dado la oportunidad de facilitar la integración de los estudiantes ciegos. Existen proyectos como Braitico (ONCE, s.f.), un método de alfabetización en Braille inclusivo desarrollado por la ONCE (Organización Nacional de Ciegos de España) destinado a que los niños aprendan braille. En el campo de la música, ya hay programas informáticos para que las personas con discapacidad visual puedan visualizar y editar música usando notación braille (Homenda, 2008) como el BME, Braille Music Editor (Giuseppe Paccini, s.f.), estudios sobre cómo enseñar braille musical a niños utilizando aplicaciones informáticas (Nicotra y Quatraro, 2008; Borges y Tomé, 2014), estudios y aplicaciones enfocados a enseñar música a los estudios de ciegos utilizando *música hablada* en lugar de braille (Capozzi y col., 2012), aplicaciones de traducción de partituras de música en tinta a braille musical y viceversa como *FreeDots* (Repain y col., s.f.) e incluso existe un estándar para compartir partituras en notación musical braille en la web llamado BMML, *Braille Music Markup Language* (Encelle y col., 2009).

El problema de integrar a los estudiantes ciegos en el aula está presente en casi todos los campos de la educación. Hay muchos enfoques diferentes para buscar una solución, como *Aim-Math*, un sistema de aprendizaje de matemáticas interactivo para estudiantes ciegos y con discapacidad visual que utiliza un sintetizador de texto a voz para leer en voz alta expresiones matemáticas (Naruedomkul, 2013). Sin embargo, este enfoque no promueve la

integración en una única clase de estudiantes videntes e invidentes. Otro enfoque es EDICO (Editor Científico de la ONCE) (Carenas y col., 2018), un proyecto promovido por la ONCE y desarrollado en cooperación con la Universidad Complutense de Madrid. Es un editor de matemáticas, física y química accesible. Traduce en tiempo real lenguaje científico en tinta a notación científica en braille y viceversa. Esto permite a los estudiantes ciegos seguir una clase de ciencias interactuando en tiempo real con un profesor que no sabe Braille.

Tras el éxito de EDICO, la ONCE quiso desarrollar soluciones similares para integrar a los estudiantes ciegos en otras asignaturas, como la música. Así nació la idea de *LiveDots*: un editor musical que traduce partituras en tinta a braille musical en tiempo real.

## 1.1. Motivación

Hoy en día, cualquier persona puede conseguir partituras para poder acceder a ellas más tarde. Esto resulta muy útil para el aprendizaje musical, entre otras cosas.

Por ejemplo, un profesor puede utilizar una herramienta de edición de partituras (como *MuseScore* o *Sibelius*, entre otras) para crear unas partituras que permitan a sus alumnos practicar algún tipo de ejercicio concreto. Otra opción es, durante el desarrollo de una clase, proyectar una partitura para estudiarla.

Desgraciadamente, estos sistemas suelen estar menos desarrollados para gente con discapacidad visual. En el caso del ejemplo anterior, sí existen programas (como *FreeDots*) que permiten traducir una partitura a braille y usar la línea braille para su lectura y otros que permiten escribir una partitura en braille y traducirlas para su visualización (por ejemplo, *Braille Music Editor*, que permite escribir una partitura en braille y exportarla en un formato que otra aplicación, como *MuseScore*, permite visualizar como una partitura en tinta).

Una situación típica durante el desarrollo de una clase consiste en que el profesor proponga un ejercicio (supongamos de crear una partitura) para posteriormente corregirlo. Sin embargo, un profesor de música vidente no tiene porqué saber braille. El alumno entonces puede usar un programa para escribir su partitura y traducirla. Después, el profesor podrá leerla, corregirla y escribirla en un programa para traducción de partituras a braille y, así, finalmente, el alumno tendrá su partitura corregida.

Aunque este método funciona, es claro que no es prácticamente nada interactivo, porque ni el profesor ni el alumno tienen acceso a las creaciones del otro hasta que son finalizadas. Se podría decir que, en este caso, la relación profesor-alumno tiene un componente epistolar, ya que, hasta que cada uno no acaba su "carta", el otro no puede leerla y escribir la suya.

Con este ejemplo se pone de manifiesto que resulta realmente complicado que alumnos con discapacidad visual se sientan incluidos en una clase de música, ya que hay muchas actividades en las que no pueden participar. Es por eso que resulta necesario un sistema que permita aumentar su participación y, con ello, su inclusión en la clase.

Otra situación se da en el ámbito profesional de la música. Por ejemplo, es mucho más

complicado integrar a un músico con discapacidad visual en una orquesta ya que los cambios o adaptaciones que realice el director de orquesta en una partitura en tinta deberán ser traducidos a braille para que este músico pueda leerlos. Esto también ocurre con las propuestas del propio músico, que también deberá traducir al completo para que sean legibles en tinta. Esto da lugar también a una comunicación poco interactiva, ya que ninguno de los dos podrá ver los cambios en tiempo real, sino que tendrán que esperar a la traducción completa de la otra persona para poder leer los cambios en la partitura.

Así, se ve que, aunque hay sistemas que permiten compartir las partituras entre personas videntes e invidentes, la interacción es lenta, lo que provoca una menor inclusión de gente con esta discapacidad. Además, esto pasa en diferentes niveles dentro de la música, tanto a gente que está aprendiendo a tocar un instrumente como a músicos profesionales.

Por tanto, es importante desarrollar algún tipo de sistema que permita "agilizar" la comunicación de los cambios en las partituras entre gente vidente e invidente para favorecer la interacción entre ellos. Si se consiguiera, habría más gente que podría disfrutar de la música y, sobre todo, habría más gente que podría sentirse incluida en el ámbito del aprendizaje y desarrollo musical.

## 1.2. Objetivos

Acabamos de ilustrar la necesidad de tener una forma interactiva de compartir partituras entre videntes e invidentes. Queremos integrar los elementos que necesitan ambos para entender la música en una aplicación que les permita interactuar en tiempo real con la partitura. Así nace la idea de *LiveDots*.

La aplicación mostraría al mismo tiempo la partitura en tinta por pantalla y la partitura en braille mediante una línea braille. Un vidente podría ver un pentagrama en tinta y modificarlo, reflejándose, en tiempo real, estas modificaciones en la partitura braille. Del mismo modo, una persona con discapacidad visual podría leer en la línea braille la misma partitura e introducir modificaciones en la misma, bien mediante la propia línea o bien mediante pulsaciones de teclado, que se reflejarían en tiempo real en el pentagrama en tinta.

### 1.2.1. Objetivos Iniciales

Desarrollar una aplicación como la que acabamos de describir es un proyecto muy extenso. Por ello nuestro objetivo principal es hacer una primera prueba de concepto de una aplicación de traducción de partituras en tinta a braille en tiempo real, dejando la modificación de partituras para una posible futura ampliación. Decidimos centrarnos en algunas funcionalidades y desarrollar código modular y legible para que el proyecto pudiese continuarse en el futuro y aprovechar así el trabajo hecho. Inicialmente, los objetivos de nuestro trabajo eran los siguientes:

- Estudiar y analizar el estado actual del campo de las aplicaciones de notación musical

en braille.

  - Estudiar qué hay hecho.
  - Estudiar qué falta por hacer.
  - Ver qué podemos aportar.

- Hacer una primera prueba de concepto de aplicación de traducción de partituras con las siguientes especificaciones

  - Traducción en tiempo real de partitura en tinta a braille
  - Traducción en tiempo real de braille a tinta.
  - Compatibilidad con MusicXML
  - Salida por pantalla en braille y partitura en tinta
  - Salida por línea braille
  - Accesible mediante un revisor de pantalla

- Probar la aplicación en una línea braille

- Probar la aplicación con usuarios con discapacidad visual y realizar una evaluación de la misma

- Redactar un **artículo** sobre el desarrollo de la aplicación *LiveDots* así como sobre sus posibles aplicaciones y las implicaciones que puede tener en otros campos (ver anexo A) para el congreso ICCE 2020 ($28^{th}$ *International Conference on Computers in Education*) organizado por APSCE (*Asia-Pacific Society for Computers in Education*). Este es un congreso de tipo Core B. La intención de artículo es divulgar las implicaciones de nuestro trabajo en el mundo académico para intentar que llegue a tener un impacto real sobre el panorama educativo actual.

### 1.2.2. Modificación de los Objetivos

El 15 de marzo de 2020 se decretó en España el estado de alarma debido a la pandemia mundial causada por el COVID-19. Una de las medidas fue el confinamiento de la población. Debido a esta situación social que ha surgido durante el desarrollo de nuestro proyecto ha habido objetivos que no hemos podido llevar a cabo por falta de medios. La principal consecuencia de esta situación ha sido no poder colaborar con la ONCE como teníamos planeado. Esto nos ha hecho modificar o eliminar los siguientes objetivos:

- **Traducción en tiempo real de braille a tinta.** Mientras que la traducción de tinta a braille es unívoca y está recogida en un manual internacional (Krolick, 1996), la traducción inversa, de braille a tinta, no es unívoca. Por tanto para poder hacer una implementación de esta traducción se necesita la colaboración directa mano a mano con un experto en la materia, ya que hay que evaluar caso por caso como realizar dicha traducción. Al no poder contar con dicho experto hemos suprimido esta funcionalidad de nuestro proyecto, dejándolo como una posible ampliación en un trabajo futuro.

- **Probar la aplicación en una línea braille.** Para poder cumplir este objetivo era indispensable disponer de una línea braille. Íbamos o bien a hacer estas pruebas directamente en la sede de la ONCE, o bien, pedirles una línea braille prestada para poder probarlo. Ninguna de estas opciones se pudieron llevar a cabo por la situación de confinamiento. Sin embargo, está implementada esta funcionalidad a falta de probarla.
- **Probar la aplicación con usuarios con discapacidad visual y realizar una evaluación de la misma.** De nuevo por el mismo motivo, no hemos podido colaborar presencialmente con usuarios con discapacidad visual.

Hemos transformado el esfuerzo que íbamos a dedicar a estos objetivos en añadir funcionalidad al programa y realizar un experimento distinto:

- **Modificación en tiempo real de partitura en tinta.** Hemos añadido esta funcionalidad a nuestro programa. Se permite arrastrar las notas en el pentagrama en tinta cambiándolas de tono y se puede ver en tiempo real esta modificación en braille.
- **Reproducción de partitura.** Hemos añadido la posibilidad de reproducir una partitura.
- **Probar la aplicación con usuarios videntes y realizar una evaluación de la misma.** Hemos realizado un estudio que incluye una parte *blindfolded* (sin usar la vista) con usuarios videntes para comprobar la usabilidad de la aplicación desarrollada desde el punto de vista de ambos usuarios finales (vidente e invidente) y para comprobar si una aplicación con modificación en tiempo real como la que hemos desarrollado puede ser útil para integrar al alumno ciego en el aula de música.

## 1.3. Estructura de este documento

Este documento se divide en 7 capítulos, cada uno dedicado a una temática. Este sección está encuadrada en el primer capítulo introductorio en el que se han definido la motivación y los objetivos del proyecto.

El capítulo 2 recoge todo el estudio inicial del estado del arte que nos pone en antecedentes de la temática del proyecto. Está dividido en distintas secciones, las tres primeras explican el uso que hacen las personas invidentes de un computador, el uso de aplicaciones accesibles en la docencia y la musicografía braille, mientras que las dos últimas hacen un estudio de distintas aplicaciones musicales y de edición de partituras creadas para ciegos entorno a la accesibilidad y de aplicaciones musicales dirigidas mayormente a público vidente, respectivamente.

En el capítulo 3 explicamos todo lo relacionado con la implementación de la aplicación *LiveDots* que ha sido la parte que más tiempo ha llevado de este proyecto. Las dos primeras secciones cuentan en lineas generales la gestión del proyecto y las tecnologías usadas. La tercera sección cuenta pormenorizadamente cómo se ha llevado a cabo el desarrollo de la aplicación y da detalles sobre partes relevantes de la implementación. También habla de todas las características relacionadas con accesibilidad.

El capítulo 4 recopila el experimento que hemos llevado a cabo para probar la aplicación desarrollada. Las diferentes secciones describen el objetivo del experimento, los participantes, el diseño experimental, los instrumentos usados para realizar el experimento, los resultados obtenidos y la discusión sobre estos resultados.

El capítulo 5 recoge la narración de cada autor de este trabajo sobre su aportación del proyecto y su visión del mismo.

En el capítulo 6 damos una visión general del trabajo futuro que inspira este proyecto. Por último, el capítulo 7 explica detalladamente las conclusiones obtenidas después de realizar este trabajo.

# Introduction

Since de beginning of time, music has accompanied humankind having a very important roll through its history. It is a ubiquitous element in every culture, a universal cultural manifestation that has the power of communicating no matter the barriers in time, space or language (Wallin y col., 2001). It is, therefore, a very powerful element.

Musical notation is the generic name that is given to any writing system used for representing graphically any music piece (de Candé, 2002). The first testimonies of written scores using musical notation in the occidental culture are dated in between forth century and beginning of the III century B.C. Burkholder y col., 2008. In fact we know that musical theory goes back even further in time, the Pythagorean tuning which is the construction system for the music scale that gives birth to the circle of fifths is attributed to the philosopher Pythagoras in the sixth century B.C. and it is still used nowadays (Benward y Saker, 2009). The ability to portray music in a paper allows us to share it and communicate it through time and space.

The most extended method of musical representation amongst the blind people is musical braille, which is an adaptation of the reading and writing braille system. The braille system consists of different combinations of embossed dots (Kent, 2012). The musical braille is a transcription technique which allows to represent any conventional music score with a precise notation (de Candé, 2002). The knowledge of musical braille gives the blind musician not only a tool to comprehend and express music but also a very different conception of music to that of a sighted person that is used to the graphical representation in a music staff (Abramo y Pierce, 2013; Johnson, 2015).

Music is an element of union, however in music education there is still a gap between sighted people and people with visual disabilities. In many cases, blind students do not receive the necessary training to understand musical braille notation and many sighted teachers do not know how to facilitate student learning, as denounced by David Goldstein, director of the *National Resource Center for Blind Musicians* in the United States. . An alternative for blind students to improve their musical knowledge is to attend a school for the blind. In this scenario, sighted and blind students are taught using different methodologies that lead to a dichotomy between musical braille and conventional musical writing. Lack of familiarity with conventional musical writing makes it more difficult for blind students to later enter settings with sighted musicians, such as a conservatory or orchestra (Goldstein, 2000).

There is therefore a need to integrate sighted and blind students in the same classroom and teach each of them the essential knowledge to develop their musical skills (Quaglia, 2015; Buhagiar y Tanti, 2011). That is, we want blind students to be able to follow a music class with mostly sighted students as they learn musical braille notation and become familiar with the traditional musical conception of printed music.

Some of the strategies most commonly used by blind or visually impaired students to facilitate participation in the classroom are: enlarged printed sheet music, that another person reads the sheet for them (classmates, parents or teachers) and the use of musical braille whenever possible (Frederick, 2009; Smaligo, 1998). All these tools need the help of a person to transcribe the score orally or a teacher who knows the musical braille and teaches it to the student, which is not always possible. This makes the student depend on the people around him.

With the development of technology we have in our hands the possibility of creating tools that help us integrate the musical conception of blind and sighted people, thus facilitating aspects such as the integration of blind and sighted students in music classes or the collaboration of sighted and blind musicians in an orchestra. There are already programs designed for visually impaired people to view and edit music in braille notation (Homenda, 2008), studies on how to teach musical braille to children using computer applications (Borges y Tomé, 2014), and programs that translate scores from printed music to braille and vice versa.

The progress of technology in recent years has given us the opportunity to facilitate the integration of blind students. There are projects such as Braitico (ONCE, s.f.), an inclusive Braille literacy method developed by the ONCE (National Organization of the Blind of Spain) aimed at children learning braille. In the field of music, there are already computer programs for the visually impaired to view and edit music using braille notation (Homenda, 2008) such as the BME, Braille Music Editor (Giuseppe Paccini, s.f.), studies on how to teach musical braille to children using computer applications (Nicotra y Quatraro, 2008; Borges y Tomé, 2014), studies and applications focused on teaching music to blind studies using *spoken music* instead of braille (Capozzi y col., 2012), applications for translating printed music scores into musical braille and vice versa such as *FreeDots* (Repain y col., s.f.) and there is even a standard for sharing scores in braille music notation on the web called BMML, *Braille Music Markup Language* (Encelle y col., 2009).

The problem of integrating blind students into the classroom is present in almost all fields of education. There are many different approaches to finding a solution, such as *AimMath*, an interactive math learning system for blind and visually impaired students that uses a text-to-speech synthesizer to read aloud mathematical expressions (Naruedomkul, 2013). However, this approach does not promote integration into a single class of sighted and blind students. Another approach is EDICO (ONCE Scientific Editor) (Carenas y col., 2018), a project promoted by ONCE and developed in cooperation with the Complutense University of Madrid. It is an accessible math, physics and chemistry editor. It translates in real time printed scientific language to braille scientific notation and vice versa. This allows blind students to follow a science class by interacting in real time with a teacher who does not know Braille.

Following EDICO's success, the ONCE wanted to develop similar solutions to integrate

blind students into other subjects, such as music. This is how the idea of *LiveDots* was born: a music editor that translates printed sheet music into musical braille in real time.

## Motivation

Nowadays, anyone can get access to sheet music and be able to access them later. This is very useful for musical learning, among other things.

For example, a teacher can use a score editing tool (such as *MuseScore* or *Sibelius*, among others) to create scores that allow their students to practice some type of specific exercise. Another option is, during a class, to project a score to study it.

Unfortunately, these systems are usually less developed for visually impaired people. In the case of the previous example, there are programs (such as *FreeDots*) that allow you to translate a score into Braille and use the braille display for reading, and others that allow you to write a score in Braille and translate them for display (for example , *Braille Music Editor*, which allows you to write a score in Braille and export it in a format that another application, such as *MuseScore*, allows you to view as a printed score).

A typical situation during the development of a class is for the teacher to propose an exercise (let's suppose creating a score) to then correct it. However, a sighted music teacher does not necessarily know braille. The student can then use a program to write their score and translate it. Afterwards, the teacher will be able to read it, correct it and write it in a program for translating scores into Braille and, thus, finally, the student will have their score corrected.

Although this method works, it is clearly not interactive, because neither the teacher nor the student have access to each other's creations until they are finished. We could say that, in this case, the teacher-student relationship has an epistolary component, since, until each one finishes their "letter", the other cannot read and write theirs.

This example shows that it is really difficult for students with visual disabilities to feel included in a music class, since there are many activities in which they cannot participate. That is why there is the need of a system that increases their participation and, with it, their inclusion in the class.

Another situation occurs in the professional field of music. For example, it is much more complicated to integrate a visually impaired musician in an orchestra since the changes or adaptations made by the conductor in a printed score will have to be translated into braille for this musician to read them. This also occurs with the musician's own proposals, which must also be translated in full so that they are legible in a printed sheet. This also results in a poor interactive communication, since neither of them will be able to see the changes in real time, but they will have to wait for the complete translation from the other person to be able to read the changes in the score.

This way, it can be seen that, although there are systems that allow the scores to be shared between sighted and blind people, the interaction is slow, which causes less inclusion

of people with this disability. Furthermore, this happens at different levels within music, both to people who are learning to play an instrument and to professional musicians.

Therefore, it is important to develop some type of system that allows to "speed up" the communication of changes in the scores between sighted and blind people to favour the interaction between them. If this was achieved, there would be more people who could enjoy music and, above all, there would be more people who could feel included in the field of musical learning and development.

# Objectives

We have just illustrated the need for an interactive way of sharing sheet music between sighted and blind. We want to integrate the elements that both need to understand the music in an application that allows them to interact in real time with the score. This is how the idea of *LiveDots* was born.

The application would display the printed sheet music on the screen and the braille sheet music at the same time using a braille display. A sighted person could see a printed staff and modify it, reflecting, in real time, these modifications in the Braille score. Similarly, a visually impaired person could read the same score on the braille display and make changes to it, either by the display itself or by keystrokes, which would be reflected in real time on the printed staff.

## Initial Objectives

Developing an application like the one just described is a very extensive project. Therefore, our main objective is to make a first proof of concept of an application for translating printed scores into Braille in real time, leaving the modification of scores for a possible future expansion. We decided to focus on some functionalities and develop modular and readable code so that the project could be continued in the future and thus take advantage of the work done. Initially, the objectives of our work were as follows:

- Study and analyze the current state of the field of braille music notation applications.
  - Study what has been done.
  - Study what remains to be done.
  - See what we can contribute.
- Make a first proof of concept of a sheet music translation application with the following specifications
  - Real-time translation of printed sheet music to Braille
  - Real-time translation of braille to printed music.
  - MusicXML compatibility

    - Screen output in Braille and printed sheet music
    - Output through braille line
    - Accessible through a screen reader
- Test the application in a braille line
- Test the application with visually impaired users and make an evaluation of it
- Write an **article** on the development of the application *LiveDots* as well as its possible uses and the implications it may have in other fields (see Annex A) for the ICCE 2020 congress ($28^{th}$ *International Conference on Computers in Education*) organized by APSCE (*Asia-Pacific Society for Computers in Education*). This is a Core B type congress. The intention of the article is to disclose the implications of our work in the academic world to try to have a real impact on the current educational landscape.

## Objectives Modification

On March 15, 2020, the state of alarm was declared in Spain due to the global pandemic caused by COVID-19. One of the measures was the confinement of the population. Due to this social situation that has arisen during the development of our project, there have been objectives that we have not been able to achieve due to lack of means. The main consequence of this situation has been not being able to collaborate with ONCE as we had planned. This has made us modify or eliminate the following objectives:

- **Real-time translation from braille to printed music.** While the translation from printed music to braille is univocal and is reflected in an international manual (Krolick, 1996), the reverse translation, from braille to printed music, is not univocal . Therefore, in order to implement this translation, direct hand-to-hand collaboration with an expert in the field is necessary, since it is necessary to evaluate case by case how to carry out said translation. Due to the lack of such an expert, we have removed this functionality from our project, leaving it as a possible extension in future work.
- **Testing the application in a braille line.** In order to achieve this objective, it was essential to have a braille line display. We were either going to do these tests directly at the ONCE headquarters, or to ask them to borrow a braille line to be able to test it. None of these options could be carried out due to the confinement situation. However, this functionality is implemented despite not having tested it.
- **Testing the application with visually impaired users and evaluating it.** Again for the same reason, we were unable to collaborate in person with visually impaired users.

We have transformed the effort that we were going to dedicate to these objectives to add functionality to the program and carry out a different experiment:

- **Real-time modification of printed sheet music.** We have added this functionality to our program. It allows to drag notes on the printed staff changing their tone and you can see this modification in real time in Braille.

- **Score playback.** We have added the ability to play a score.
- **Test the application with sighted users and make an evaluation of it.** We have carried out a study that includes a *blindfolded* part (without using visualization) with sighted users to check the usability of the application developed from the point of view of both end users (sighted and blind) and to check if an application with real-time modification like the one we have developed can be useful to integrate the blind student into the music classroom.

## Structure of this Document

This document is divided into 7 chapters, each one dedicated to a theme. This section is framed in the first introductory chapter in which the motivation and objectives of the project have been defined.

Chapter 2 collects all the initial study of the state of the art that gives us background on the subject of the project. It is divided into different sections, the first three explain how blind people use a computer, the use of accessible applications in teaching and the braille music notation, while the last two make a study of different musical applications and edition of scores created for the blind around accessibility and musical applications aimed mainly at sighted audiences, respectively.

In Chapter 3 we explain everything related to the implementation of the *LiveDots* application, which has been the longest part of this project. The first two sections generally outline the project management and the technologies used. The third section tells in detail how the development of the application has been carried out and gives details on relevant parts of the implementation. It also talks about all the accessibility related features.

Chapter 4 compiles the experiment that we have carried out to test the developed application. The different sections describe the objective of the experiment, the participants, the experimental design, the instruments used to carry out the experiment, the results obtained and the discussion of these results.

Chapter 5 collects the narration of the contribution made by each author of this project and their point of view on it.

In Chapter 6 we give an overview of the future work that inspires this project. Finally, Chapter 7 explains in detail the conclusions obtained after carrying out this project.

# Capítulo 2

# Estado del Arte

## 2.1. Uso de computadores por personas invidentes

El uso tradicional de un ordenador por una persona vidente se basa principalmente en la percepción visual de la información. Un avance muy importante en la tecnología es la adaptación de los sistemas informáticos para que sean accesibles a las personas ciegas o con discapacidad visual. Una de las cuestiones más importantes a resolver es cómo comunicar al invidente la información que se muestra habitualmente en la pantalla. Esto, hoy en día, se resuelve principalmente de dos formas.

La primera es con un *revisor de pantalla*, que consiste en una aplicación que reproduce por medio de audio lo que se muestra en la pantalla. El revisor tendrá que ofrecer la información relevante para el usuario y no toda la información disponible en la pantalla para que la comunicación sea eficiente. Para ello, dependiendo del revisor y de las aplicaciones en uso, se suele decir por audio el elemento que está seleccionado en ese momento y la información relevante a ese elemento.

Existen varios revisores de pantalla diferentes, entre ellos está JAWS (Freedom Scientific, s.f.) que se trata de un revisor de pantalla implementado para el sistema operativo Windows, que funciona a través de scripts que capturan y modifican los eventos y acciones realizadas por el sistema. JAWS es una herramienta muy potente y dentro de los revisores de pantalla, es de los más populares pero, como pega tiene que es necesario adquirir una licencia. Una buena opción gratuita es el revisor de pantalla NVDA (NonVisual Destok Access, NV Access, s.f.) es software libre, y es el segundo más utilizado por los usuarios según algunos estudios (WebAIM, s.f.). "ZoomText", s.f., es un software de pago para Windows. Ofrece además del lector de pantalla, un magnificador. El magnificador es una herramienta que amplía el tamaño de lo que se muestra en la pantalla, como si fuera una lupa, lo que hace muy útil para personas con baja visión. Window-Eyes (GW Micro, s.f.) es un lector de pantalla de pago que en su momento fue bastante utilizado pero que ha cesado su venta y, por tanto, ha quedado obsoleto. Por último, "VoiceOver", s.f. es el lector de pantalla que utiliza el sistema operativo iOS y está integrado por defecto, sin coste adicional en todos los ordenadores

MAC (también en las tabletas, teléfonos móviles y resto de productos de la marca Apple).

También se han desarrollado lectores de pantalla para los dispositivos móviles, lo más utilizados son VoiceOver que como ya hemos visto está directamente integrado en los dispositivos de Apple y "Google TalkBack", s.f. que es una aplicación de descarga gratuita para el sistema operativo Android.

La segunda forma es mediante una *línea braille* (figura 2.1). Se trata de un componente hardware que se conecta al ordenador e imprime, a tiempo real, caracteres braille que se corresponden con lo que se muestra por pantalla. Las líneas braille pueden variar según su modelo, pero suele ser habitual que ofrezcan la posibilidad de interactuar mediante un sistema similar al de un teclado para introducir información en el ordenador.


Figura 2.1: Línea braille. Modelo Humanware Brailliant BI 40

Además del revisor de pantalla y la línea braille hay otras formas de adaptar la tecnología al servicio de personas invidentes o con discapacidad visual. Las técnicas, conocimientos y recursos de esta adaptación son tema de estudio de la *tiflotecnología*. En España, hay una organización solidaria que trabaja para la inclusión de las personas ciegas y con discapacidad en la sociedad que es la Organización Nacional de Ciegos Españoles ("ONCE", s.f.). La ONCE tiene un centro dedicado al estudio y desarrollo de la tiflotecnología que es el "CTI. Centro de Tiflotecnología e Innovación de la ONCE", s.f.

Es necesario mencionar que la adaptación de la tecnología es esencial para permitir el uso del computador a las personas invidentes, pero esto no es suficiente ya que tan solo son herramientas que reproducen el contenido disponible en las aplicaciones. Si estas aplicaciones no son accesibles, será muy difícil o incluso imposible que puedan ser usadas por un usuario ciego. Para que una aplicación sea *accesible*, tiene que cumplir ciertos requisitos

que permitan a cualquier usuario, independientemente de su diversidad funcional, utilizar satisfactoriamente la aplicación con su sistema de acceso habitual. Estos requisitos incluyen pero no se restringen a los siguientes (CESyA. Centro Español del Subtitulado y la Audiodescripción, s.f.):

- Diseñar un sistema de navegación a través de la aplicación que no requiera el uso del ratón. Se tiene que poder hacer uso completo de la aplicación a partir del teclado.
- Implementar un sistema de verbalización de los elementos y controles para que un revisor de pantalla informe adecuadamente de las acciones que se realizan en la aplicación y de la información relevante que esta ofrece.
- Permitir modificar el tamaño de los elementos y textos para personas con visibilidad reducida.
- Escoger colores con alto contraste.
- Añadir una guía de ayuda al usuario indicando la forma de usar la aplicación.

La adaptación de la tecnología para lograr un uso accesible es un avance enorme hacia la inclusión. Mediante el desarrollo de aplicaciones accesibles que hagan uso tanto de la línea braille como de un revisor de pantalla para la muestra del contenido, se consigue que las personas ciegas puedan hacer uso de las mismas aplicaciones que las personas videntes. Como veremos a continuación, la tecnología, una vez adaptada se puede utilizar como herramienta para potenciar y mejorar la inclusión en otras áreas importantes como puede ser la enseñanza.

## 2.2. Docencia con aplicaciones accesibles

La tecnología que se desarrolla para adaptar el uso de computadores a personas ciegas o con discapacidad visual es muy útil para que alguien que antes no podía usar el ordenador o los dispositivos móviles, pueda acceder a ellas. Sin embargo, puede ser mucho más que una mera adaptación de la tecnología. Se puede lograr potenciar y mejorar la inclusión en muchas áreas diferentes con estas herramientas. Una de las áreas más interesantes es la educación. El uso de dispositivos y aplicaciones accesibles en las aulas puede mejorar inmensamente las condiciones del alumno invidente. Esta tecnología, orientada a la enseñanza puede ayudar con la traducción de textos escritos en tinta a braille, la lectura de contenido digital, incluso, con el aprendizaje del braille. A continuación, se mencionan diferentes aplicaciones accesibles que se utilizan por todo el mundo para mejorar la enseñanza de los alumnos invidentes.

La “Perkins School for the Blind”, s.f. en Massachusetts es una de las escuelas líderes en la enseñanza de alumnos ciegos en el mundo. Ahí, a parte de dedicarse a la enseñanza de alumnos en el día a día, desarrollan programas de estudio y herramientas tecnológicas para facilitar y mejorar la educación de las personas con discapacidad visual. En su página podemos encontrar enlaces a muchas herramientas y artículos escritos diariamente sobre la actualidad tecnológica que son útiles para que asociaciones o individuos de cualquier parte del planeta puedan hacer uso de los numerosos recursos disponibles. Por ejemplo, podemos

informarnos de la última iniciativa de "ObjectiveEd", s.f. en colaboración con Microsoft para desarrollar una inteligencia artificial accesible que pretende sustituir al profesor enseñando Braille a un estudiante (Eyewire News, s.f.). Este proyecto pretende solucionar la necesidad de un profesor especializado para cada alumno, dándole a cada estudiante las herramientas adecuadas para recibir un aprendizaje específico al mismo tiempo que autónomo. Este proyecto está aún en desarrollo pero, mientras tanto, hay otras opciones a la hora de enseñar el lenguaje Braille.

En España, la ONCE hace una labor similar a la de la Perkins School, dedicándose a buscar formas de incluir a los alumnos invidentes en las aulas y de facilitar su aprendizaje con herramientas accesibles. En la "Web de Educación de la ONCE", s.f. se recopila y se informa sobre las diferentes aplicaciones accesibles orientadas a la educación. Entre ellas está uno de los últimos proyectos de la ONCE llamado Braitico (ONCE, s.f.). Este consiste en un método de enseñanza para que los más pequeños aprendan la lectura y escritura en braille. Con este método se utilizan herramientas tecnológicas a través del ordenador para su mayor eficacia. El ofrecer una solución digital a la enseñanza mejora la facilidad y comodidad a la hora de distribuir el contenido. Además permite corregir, mejorar y actualizar el programa ofreciendo la mejor versión posible.

Aparte de tener aplicaciones que enseñen el sistema Braille, para lograr una enseñanza verdaderamente inclusiva, es interesante el desarrollo y uso de aplicaciones que fomenten la integración de los alumnos con ceguera o discapacidad visual dentro de las aulas. Esto consiste en proporcionar las herramientas necesarias para que un profesor sin conocimiento del sistema Braille pueda ofrecer a una persona invidente todo el contenido necesario para el aprendizaje. También, en el sentido contrario, es necesario que la persona invidente pueda comunicar información a la persona vidente. Podemos poner el ejemplo de un profesor preparando un examen para una clase en la que hay un alumno invidente. Este profesor no conoce el lenguaje braille y, por tanto, no puede proporcionar una copia del examen a su alumno invidente.

Buscando soluciones a este problema, encontramos software de traducción a Braille o editores de texto en Braille para permitir la comunicación de información escrita en ambos sentidos. Existen programas e incluso sitios web que realizan la traducción de textos a braille de forma inmediata como "Braille Translator", s.f. y "Traductor braille : Traduce de Braille a Texto", s.f. En el área de aplicaciones de escritorio que permiten la edición y traducción de texto Braille ("Braille Translation Software - Index Braille", s.f.), podemos encontrar diversos programas que se utilizan por todo el mundo como Ebrai, Euler o NatBraille (figura 2.2). Este último, por ejemplo, es una aplicación de software libre que fue financiada por el Ministerio de Educación Francés. Como parece lógico un editor capaz de traducir a Braille es una herramienta muy útil para la integración de un alumno invidente en el aula pero no es suficiente, por ejemplo, para traducir símbolos matemáticos y ecuaciones a braille.

Es en este ámbito donde encontramos aplicaciones más especificas como Edico o Lambda. Edico ("CTI. Editor Científico ONCE", s.f.) es un proyecto realizado en colaboración entre la UCM y la ONCE que consiste en un editor matemático accesible. Es un programa muy potente que permite escribir y traducir entre lenguaje científico y Braille en los campos de matemáticas, física y química. Esta funcionalidad permite que un alumno ciego pueda realizar los ejercicios o recibir las lecciones de manera sencilla. No solo permite traducir entre un lenguaje y otro cómodamente, sino que además lo hace en tiempo real, generando

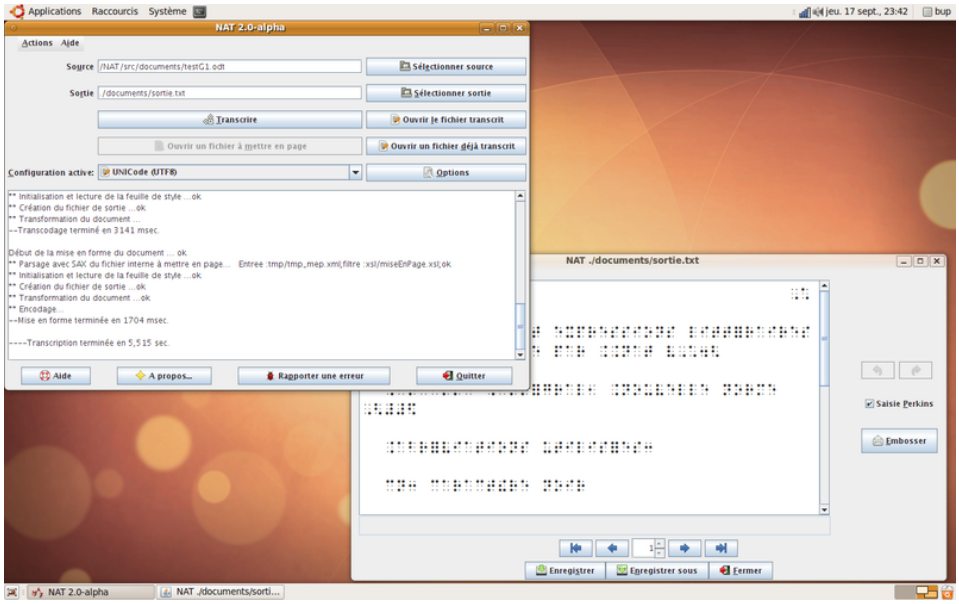



Figura 2.2: Captura de pantalla del editor de Braille NatBraille ("NatBraille : un transcripteur Braille libre", s.f.).

una interacción entre el profesor y el alumno esencial para el aprendizaje. También existen calculadoras accesibles ("Accessible calculators - ATWiki", s.f.) que permiten su uso por una persona invidente como la Talking Texas Instruments Scientific Calculator, desarrollada por Living Aids o, incluso, alguna aplicación de calculadora accesible como "Talking Calculator on the AppStore", s.f. que ofrece una opción mucho más económica aunque menos potente y cómoda.

Esto solo son algunas de las herramientas que hay para ofrecer un aprendizaje accesible para aquellos que lo necesitan. Como conclusión que aunque hay disponibles diversas aplicaciones muy útiles, normalmente no son sencillas de encontrar y en la mayor parte de los casos, requieren una licencia de pago. Por tanto, algunas de estas herramientas no son totalmente accesibles puesto que ofrecen una solución pero a una coste alto y en consecuencia no es una solución para todo el mundo. Esto se está intentando cambiar con la iniciativa de organizaciones como Perkins o la ONCE pero aún así queda un largo camino por recorrer hasta una educación completamente accesible.

## 2.3. Músicografía Braille

Cuando pensamos en una partitura musical, normalmente, pensamos en un papel con varios pentagramas y sobre estos, diferentes símbolos musicales y anotaciones que describen la composición musical. Sin embargo, este no es un formato que pueda interpretar una persona invidente. Para ello, existen diferentes sistemas de representación musical que hacen uso de elementos que puedan ser interpretados de forma táctil.

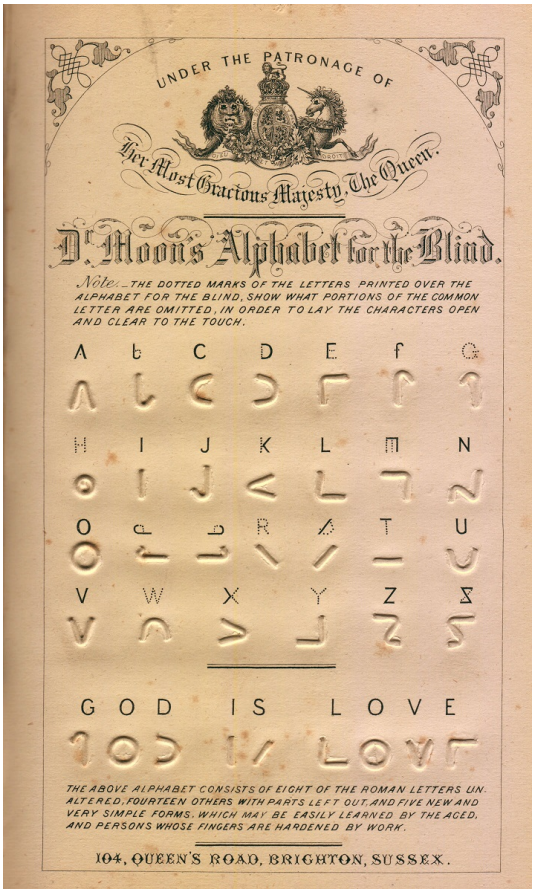



Figura 2.3: Representación de las letras en Moon Type.

Los símbolos que se han diseñado alrededor del mundo para permitir la lectura por medio del tacto son muy diferentes ("Una Breve Historia de los Sistemas de Escritura Táctil para Lectores con Ceguera e Discapacidades Visuales", s.f.). Desde imprimir las letras del alfabeto en relieve (utilizado en el sistema Boston Line Type, figura 2.4) hasta la codificación por medio de puntos del sistema Braille. Pasando, también, por formatos como el Moon Type (figura 2.3) que utiliza símbolos con cierta similitud a las letras, lo que facilita el aprendizaje a las personas que han perdido la vista tras una edad avanzada y están familiarizadas con la forma de las letras. Dicho esto, con el objetivo de universalizar y estandarizar un código para todo el mundo, el sistema Braille ha resultado la elección más popular y la que hoy en día se utiliza en la mayor parte del planeta.

Louis Braille, mientras estudiaba en el Instituto Real para Jóvenes Ciegos en Francia, diseñó un sistema basado en puntos para que las personas ciegas pudieran leer y escribir de forma rápida y eficiente. Además, era un gran aficionado a la música y se aseguró de que su sistema fuera lo suficientemente flexible para permitir la representación de partituras para cualquier instrumento. Este sistema utiliza cajetines de seis puntos para representar todos los elementos necesarios (Figura 2.5). Sin embargo, el sistema propuesto por Louis Braille

Figura 2.4: Boston Type.

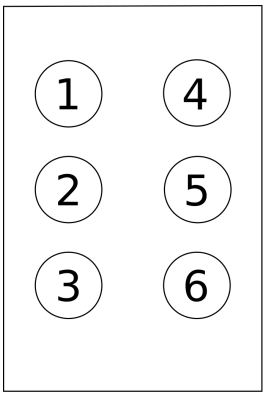



Figura 2.5: Cajetín Braille.

no es el único que utiliza la representación de puntos.

Uno de los problemas que pudiera tener el sistema Braille es el reducido tamaño de las celdas. La representación sobre seis puntos conlleva la ambigüedad de los símbolos y la utilización de una gran cantidad de cajetines para algunas figuras musicales. Con la intención de diseñar un sistema que mejorase la capacidad de representar música, Gabriel Abreu amplió y modificó a su conveniencia el sistema Braille para acabar creando su propio sistema con ocho puntos (Burgos-Bordonau, s.f.). Este es conocido como *sistema Abreu* (figura 2.6) y permite mayor número de combinaciones por cada cajetín y, por tanto, una estructura menos compleja. Además de este hubo otro formato importante desarrollado en España contemporáneamente con los de Louis Braille y Gabriel Abreu (Campos-Arcaraz, 2017). Este es el sitema Llorens (figura 2.6) que consiste en representar los símbolos musicales a partir de líneas que cambian su significado según la posición y orientación. A pesar de que, tanto el sitema Abreu como el sistema Llorens, se utilizaron en España durante bastante tiempo, el sistema Braille acabó imponiéndose debido a su popularidad y universalización.

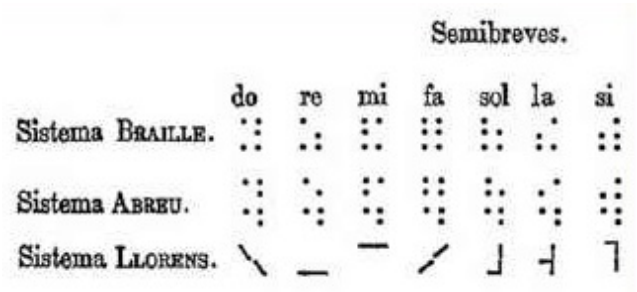



Figura 2.6: Comparación de la representación de las siete notas musicales con duración semibreve (redonda) en los sistemas Braille, Abreu y Llorens.

El sistema Braille se extendió mundialmente, pero aún así había regiones del planeta con sus propias formas para representar música. Además, el sistema Braille se modificaba y adaptaba según la zona donde se utilizara y esto llevaba a una inconsistencia en la forma de

representar partituras en Braille entre los diferentes países. Tomando de base el sistema de seis puntos elaborado por Louis Braille, Bettye Krolick recopiló y escribió el *Nuevo Manual Internacional de Musicografía Braille* (Krolick, 1996). Este manual recoge las reglas de la representación musical en braille para la estandarización y universalización de las partituras en este formato. Un manual internacional permite, entre otras cosas, la elaboración de sistemas informáticos que hagan uso de este sistema de musicografía para que se puedan utilizar por todo el mundo, y permite que todas las personas trabajen sobre el mismo formato para una mayor eficiencia del desarrollo.

En esta línea, se han realizado numerosos estudios para computarizar las partituras en Braille y también se han desarrollado diversos programas que interpretan, escriben o manipulan estas partituras. Cabe destacar estudios como *A Transcription System from MusicXML Format to Braille Music Notation* (Gotoh y col., 2008) donde proporcionan una forma de convertir una partitura en formato MusicXML a una partitura en Braille. Uno de los puntos más interesantes de este estudio es que interpretan la partitura Braille en un formato de árbol y de esta forma se puede lograr una correspondencia con la estructura de árbol de las partituras en formato MusicXML.

## 2.4. Editores musicales enfocados a personas invidentes

Una aplicación musical necesita poder almacenar la información de una partitura (por ejemplo, de una nota necesitamos su duración y su tono). Para ello, es necesario un formato de notación musical que permita este almacenamiento y edición de partituras.

La mayoría de aplicaciones musicales utilizan MIDI o MusicXML (o ambos) como formatos de notación musical. Estos formatos se diferencian de los formatos de audio (como mp3, por ejemplo) en que no están destinados a la reproducción del archivo musical (no en primera instancia, al menos), sino a su almacenamiento y modificación (esto último no es posible con un archivo de tipo mp3), es decir, están destinados a la edición musical.

MIDI fue creado en 1983 mientras que MusicXML se publicó en 2004. El primero es altamente compatible con una gran cantidad de software musical y está diseñado para facilitar la conexión de instrumentos digitales y ordenadores que se comuniquen entre sí ("El protocolo y el formato MIDI", s.f.). Sin embargo, no ofrece mucha información en el diseño de partituras (no señala agrupamientos de notas ni mecánicas de dinámica como el staccato o el picado, entre otras cosas). Por otro lado, MusicXML, a pesar de tener menos compatibilidades, permite una representación musical mucho más exacta ("Musicología digital - Fundación Juan March", s.f.), por lo que, aún siendo mucho más moderno, es un estándar y los programas de edición musical más utilizados (como Sibelius o Finale) soportan también este formato.

Las aplicaciones musicales accesibles suelen hacer uso de MusicXML como formato notacional, ya que esto permite utilizar las partituras provenientes de otras aplicaciones. Existen distintos servicios musicales relacionados con la accesibilidad, como son BME, FreeDots o DancingDots .

### 2.4.1. Braille Music Editor

Una aplicación muy interesante de cara al desarrollo de nuestro proyecto es Braille Music Editor (BME) (Giuseppe Paccini, s.f.). Como su propio nombre indica, es un editor musical en braille. Esta aplicación permite importar y exportar archivos tanto en MIDI como en MusicXML ("Braille Music Editor (BME)", s.f.). BME es un software de pago (con una versión de prueba de 30 días) de origen italiano que permite la edición de partituras con múltiples pistas para, posteriormente, reproducirlas con distintos instrumentos o exportarlas como archivos MIDI o MusicXML. Actualmente se encuentra en su segunda versión (BME2), que es la que permite esta entrada y salida de archivos con formato MusicXML.

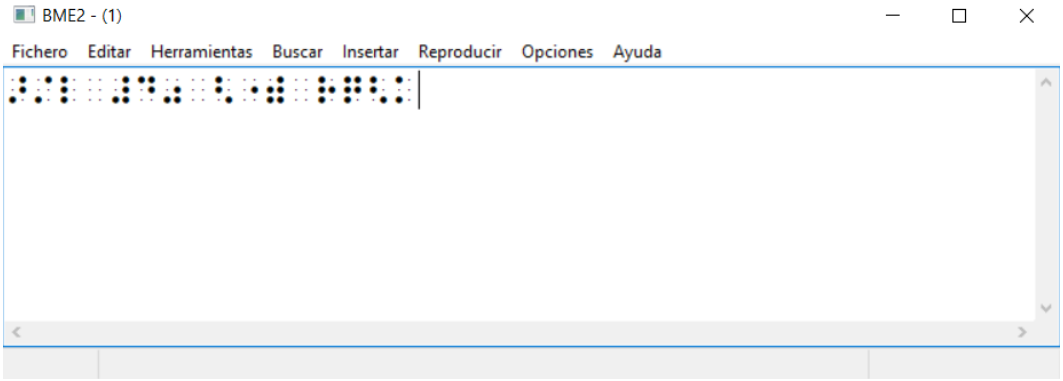



Figura 2.7: Braille Music Editor

Este programa consta de una ventana en la que el texto está escrito en braille (figura 2.7). Se puede cargar un archivo de notación musical, el cual aparecerá en la ventana traducido a braille. A partir de ahí (o sin ningún archivo cargado previamente), el usuario puede editar el fichero escribiendo caracteres en braille. Una vez editada la partitura, BME permite la reproducción de toda la partitura o de parte de la misma y tiene la posibilidad de cambiar alguna configuración (por ejemplo, se puede añadir un metrónomo o se puede cambiar el instrumento con el que se reproduce la partitura eligiendo entre un abanico de más de 100 elementos, entre los que se encuentran todos los sonidos de instrumentos MIDI estándar). Además, se puede exportar la partitura en otros formatos o guardarlo en el propio formato del programa (braille music markup language, .bmml), el cual es una extensión del formato MusicXML que soporta partituras en braille.

Para que la aplicación sea accesible, Braille Music Editor cuenta con distintas propiedades:

- Una síntesis de voz propia para leer la partitura en braille, aunque también puede hacer uso del revisor de pantalla JAWS mediante el uso de unos scripts que permiten que este revisor pueda identificar las notas y sus características de forma precisa. La síntesis de voz dispone de distintos niveles de profundidad (elegibles por el usuario). Se tiene la opción de que diga la combinación de puntos braille del carácter actual (aquel en el que se encuentra situado el cursor) o, por el contrario, que lo que diga sea el elemento musical correspondiente al carácter actual. Sin embargo, mientras se escribe,

siempre utilizará la opción de decir los puntos braille del carácter escrito ya que, sin contexto, un carácter braille puede significar distintas cosas dentro de la notación musical (incluso, hay elementos musicales que necesitan de más de un único carácter para ser representados, por lo que, hasta que el elemento no está completamente escrito, es imposible saber qué elemento musical están representando los caracteres). Sin embargo, cuando una partitura está escrita al completo en braille, su representación sí es unívoca, lo que permite al sintetizador de voz utilizar los elementos musicales en lugar de la combinación de puntos braille en su monólogo.

- Se puede utilizar la navegación con el teclado, de manera que podemos recorrer los menús de la aplicación con distintas teclas. Esta propiedad de accesibilidad es de especial importancia en el caso de las personas invidentes, ya que, al no poder alcanzar las distintas opciones mediante el uso del ratón, esta posibilidad permite el uso de todas las características de la aplicación mediante el uso del teclado (ayudado por el revisor de pantalla, el cual va diciendo el lugar en el que está localizado actualmente el foco de la aplicación).
- Permite la salida del texto por una línea braille, lo que da la posibilidad a los usuarios de que, además de escuchar mediante el revisor de pantalla el texto escrito en braille, puedan utilizar su línea braille para la lectura del mismo.
- La entrada del texto es en braille de 6 puntos, lo que permite escribir las notas, acordes o partituras de la misma forma que serán leídas posteriormente. Para ello, se utilizan las teclas $f$, $d$, $s$, $j$, $k$, y $l$, que se corresponden con los puntos 1, 2, 3, 4, 5 y 6 de un cajetín braille, respectivamente. De esta manera, si se quiere escribir el carácter cuyos puntos correspondientes son el 1, el 4 y el 6, se deberá pulsar simultáneamente las teclas $f$, $j$ y $l$.
- En caso de disponer de impresora braille, BME da la opción de imprimir el fichero actual. Además, para ello, permite configurar algunos parámetros.

Braille Music Editor nos resulta realmente interesante de cara a la accesibilidad porque nuestra aplicación necesitará tener algunas de sus propiedades: será necesario un script para JAWS que permita que este revisor de pantalla lea elementos musicales en lugar de los caracteres con los que los representamos, la navegación por la aplicación debe ser posible a través de teclado y el texto braille debe ser legible con el uso de una línea braille.

Tanto la escritura en braille como la posibilidad de imprimir el texto braille son dos características muy interesantes de cara a nuestra aplicación; sin embargo, están fuera del alcance de este trabajo.

### 2.4.2. FreeDots

Freedots es una aplicación que, dado un archivo con formato MusicXML, devuelve un archivo con la traducción a braille de la partitura en MusicXML (figura 2.8). Además, permite la reproducción de la partitura actual. A diferencia de Braille Music Editor, se trata de un proyecto de software libre que podemos descargar desde el github de Mario Lang (Repain y col., s.f.). Además, la aplicación permite visualizar la partitura en braille y tienen

la intención de que se pueda editar desde el mismo texto braille, aunque, desgraciadamente, llevan más de 3 años sin actualizar la aplicación. A partir de esta aplicación, Nicolas Froment creó un sitio web ("MusicXML to Braille converter", s.f.) en el que convertir un MusicXML a braille sin necesidad de descargarse una aplicación en el ordenador.

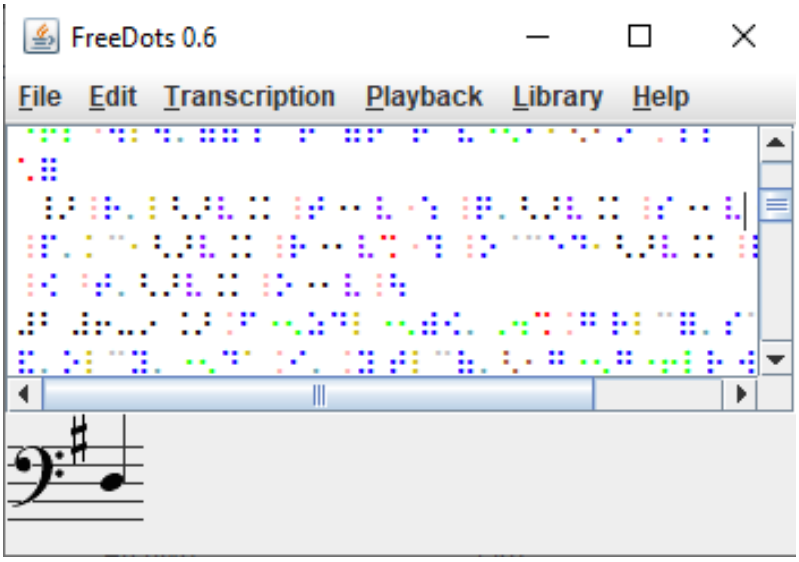



Figura 2.8: Freedots

Freedots es una muy buena aproximación a la idea de nuestra aplicación, ya que importa las partituras en MusicXML y las traduce a braille, aunque no permite la edición de la partitura en tiempo real.

### 2.4.3. Dancing Dots

Dancing Dots ("Braille Music Software for Blind: Magnified Music for Low Vision by Dancing Dots", s.f.) es una plataforma que ofrece aplicaciones para estudiantes con discapacidad visual y para sus profesores. Tienen distintos programas orientados a hacer la música más accesible.

Uno de los programas que ofrecen es el *Lime Lighter* (figura 2.9, una aplicación para que personas con visibilidad reducida puedan leer partituras. Lime Lighter muestra las partituras con un gran tamaño y permite que la partitura contraste del fondo de diversas maneras, algo que puede facilitar su lectura (por ejemplo, se puede mostrar cada una de las siete notas en distinto color). También permite el uso de un sistema OCR para escanear partituras y el uso de la aplicación mediante un pedal inalámbrico, lo que permite tener las manos libres al recorrer la partitura (aunque se puede configurar la opción de que la partitura se recorra automáticamente a la velocidad que se indique). Dispone de versión de prueba y una licencia cuesta unos 80 dólares al mes.

Además, tienen la aplicación *Lime Aloud*, que trabaja haciendo uso del revisor de pan-

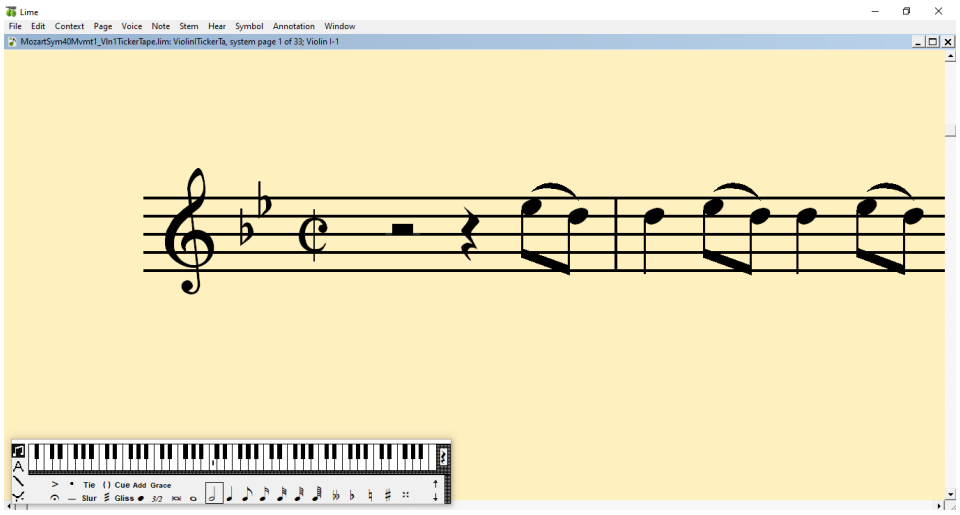



Figura 2.9: The Lime Lighter

talla JAWS. Su función es permitir la creación musical a personas con discapacidad visual haciendo uso del editor Lime.

Otro programa del que disponen es *GOODFEEL*, que permite escanear y editar una partitura para hacer su traducción a braille. Una persona invidente tendrá acceso a la partitura mediante su verbalización (o mediante su reproducción). Este software permite el uso de archivos MusicXML como entrada para la traducción. También da la opción de usar distintos revisores de pantalla, como JAWS o NVDA. Dispone de una versión de prueba de 15 días o de la opción de hacerse con una licencia del programa.

Además, disponen de cursos donde aprender musicografía braille y de otros productos y servicios. Por tanto, Dancing Dots es una plataforma orientada a la accesibilidad con diversas aplicaciones que favorecen el aprendizaje musical a personas con discapacidad visual.

## 2.5. Otras aplicaciones musicales y de edición de partituras

A la hora de afrontar el desarrollo de una aplicación tan grande como puede llegar a ser la que tenemos entre manos, el primer paso es plantearnos qué software hay ya hecho que podamos usar para implementar ciertas partes de la aplicación. Nuestro objetivo es que el producto final sea útil y fiable, para ello no es necesario, ni recomendable, hacer cada parte de nuestra aplicación desde cero.

Estudiaremos en primer lugar aplicaciones ya existentes con un propósito similar al de nuestra aplicación. Después hablaremos de distintos códigos relacionados con las partituras musicales y la notación braille.

### 2.5.1. Audacity

Audacity es un editor de audio de código libre y gratuito (Dannenberg y Mazzoni, s.f.). Permite grabar, reproducir, importar y exportar datos en varios formatos como WAV, AIFF y MP3. Trabaja con un sistema de pistas, como se puede ver en la figura 2.10. A pesar de ser una aplicación accesible, tiene ciertas carencias. Procedamos a estudiar su accesibilidad.

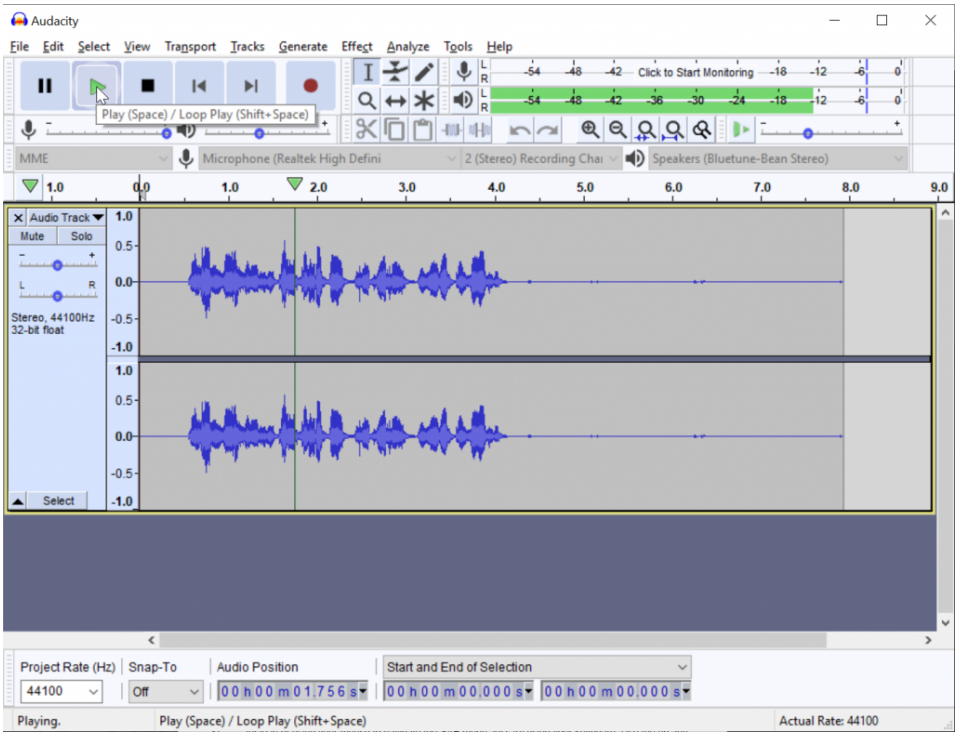



Figura 2.10: Reproducción de una pista de audio con Audacity

Tiene un alto número de atajos de teclado, que son personalizables. Sin embargo hay partes de la aplicación que no son completamente accesibles usando el teclado (Audacity Team, s.f.), lo cual priva de funcionalidad a usuarios con discapacidad visual, que usan este elemento como medio de interacción con la aplicación. Las partes de Audacity que no son completamente accesibles incluyen:

- Recortes: Un recorte es un fragmento de audio que puede ser manipulado de manera independiente. Una pista puede contener uno o más recortes organizados de manera paralela, secuencial o cualquier combinación de los mismos. En Audacity no hay forma de moverse a través de estos fragmentos usando el teclado. Por ejemplo, no hay un atajo de teclado para moverse al principio del siguiente fragmento.

- Pista de tiempo: Es una pista que sirve para controlar la velocidad de reproducción de las pistas de audio. Solo puede manejarse mediante el ratón.

- Barra de Herramientas de Herramientas: Contiene distintas herramientas para selección, ajuste del volumen, zoom y desplazamiento temporal de audio. De todas ellas, solo la herramienta de selección puede manipularse mediante el teclado.

Siguiendo con la accesibilidad de la aplicación, veamos qué lectores de pantalla soporta y cómo de bien funcionan. En Windows trabaja con NVDA, Windows-Eyes y JAWS, sin embargo sigue habiendo partes de la aplicación que tampoco serán accesibles para el revisor de pantalla. Además de los recortes y pistas de tiempo mencionadas anteriormente, añadimos a la lista de elementos no accesibles las pistas de etiquetas (es una pista adicional que contienen etiquetas para momentos y/o lapsos de tiempo) y la barra de Herramientas de medidores (muestra la amplitud del audio que se está grabando o reproduciendo).

A pesar de ser un editor muy completo para un vidente, deja fuera mucha de su funcionalidad y practicidad a la hora de usar la aplicación de forma accesible. Una visión general de esta aplicación nos ilustra la necesidad de desarrollar software pensando en la accesibilidad desde el comienzo del desarrollo, en lugar de tratar la accesibilidad a posteriori.

### 2.5.2. MuseScore

MuseScore (figura 2.11) es un programa de edición de partituras con notación musical (Schweer, s.f.). Tiene licencia GNU GPLv2 y es gratuito. Permite componer partituras para distintos instrumentos y agrupaciones. Trabaja con los siguientes formatos originales: el formato por defecto es **.mscz*, está comprimido para ocupar menos espacio en disco. Su versión descomprimida es **.mscx* y se usa para depurar o guardar en un sistema de versiones. Para copias de seguridad dispone de los formatos *.*.mscz* y *.*.mscx*.

Para permitir la compatibilidad con otro software musical MuseScore puede importar y exportar archivos MusicXML y MIDI. También es capaz de importar archivos de formato nativo de otros programas de notación musical como MuseData, Capella, Bagpipe Music Writer, BB, Overture, Guitar Pro y Power Tab Edition. En cuanto al audio, MuseScore puede crear audio estéreo de la partitura los formatos: WAV, MP3, OGG VORBIS, FLAC.

Desde su versión 3.3 la mayoría de sus características son accesibles, es decir, pueden accederse tanto mediante un lector de partituras y como mediante el editor. MuseScore viene con soporte para el lector de pantalla NVDA. También se pueden instalar scripts para habilitar el soporte para JAWS. Los lectores de pantalla soportados permiten leer la partitura nota por nota. Actualmente, el resto de lectores de pantalla no están soportados y solo leen menús y diálogos (Comunidad de desarrolladores de MuseScore, s.f.).

Tiene integrada la aplicación de código libre Audiveris que le permite importar archivos en formato pdf.

MuseScore es un editor de partituras muy completo para música en tinta y es accesible en su mayor parte, sin embargo, no soporta partituras en Braille.

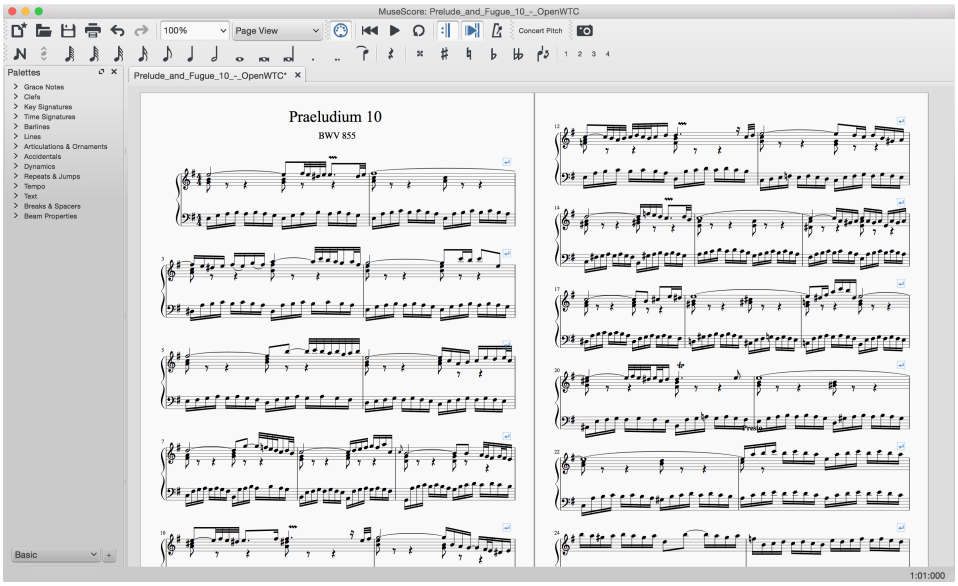



Figura 2.11: Partitura de piano visualizada en MuseScore

### 2.5.3. Sibelius

Sibelius es el programa de edición de partituras más vendido del mundo. Permite crear, editar y exportar música además de reproducirla. Está desarrollado por la empresa estadounidense Avid Technology (Avid Technology, s.f.).

Sibelius dispone de tres versiones distintas: Sibelius First, Sibelius y Sibelius Ultimate. La primera es una versión limitada de las otras dos y es la única gratuita. Las versiones de pago cuestan desde 9 y 19 euros al mes según la versión. Las versiones de pago soportan MIDI y MusicXml y son capaces de exportar el audio en formato WAV, AIFF y MP3.

Para poder usar Sibelius de forma correcta con un revisor de pantalla es necesario usar una extensión. La primera versión fue Sibelius Speaking de Dancing Dots, que dejó de suministrarse en 2009. Actualmente se usa Sibelius Access que es un conjunto de scripts JAWS y plug-ins de ManuScript (Rugman, s.f.).

Además de el hecho de ser una aplicación de pago, Sibelius también tiene el contra de no soportar partituras en Braille.

### 2.5.4. Finale

Finale (figura 2.12) es un editor de partituras que permite escribir, reproducir, imprimir y publicar partituras de música (MakeMusic, s.f.). Fue creado por la empresa MakeMusic. Es un programa de pago y la licencia completa cuesta 600$. También dispone de una versión con funcionalidad reducida, Finale PrintMusic, sin embargo dejó de actualizarse en 2014 y

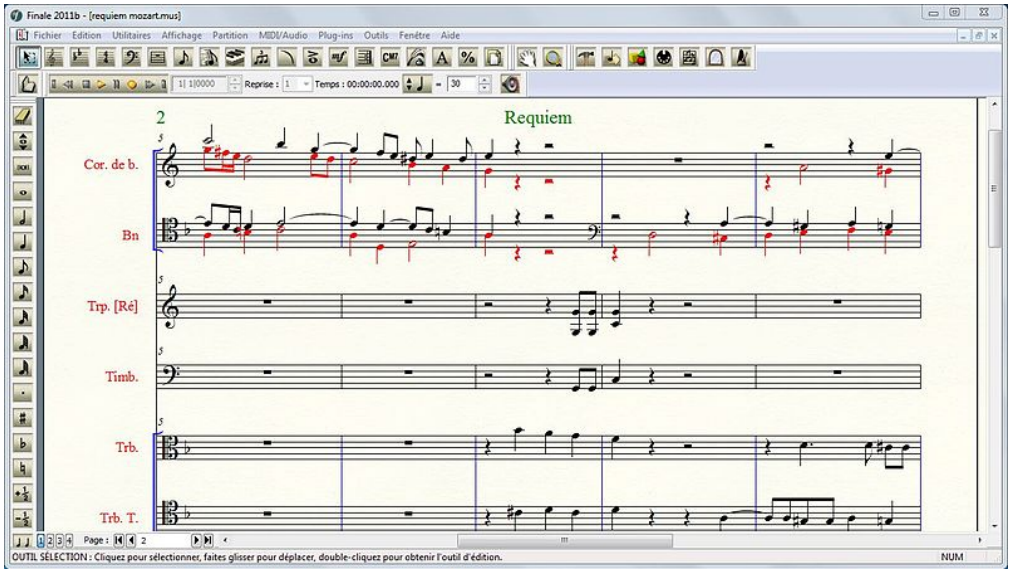



Figura 2.12: Partitura para piano visualizada en Finale 2011b

solo está disponible para Windows. Su precio es de 120$.

Permite importar archivos en formato MusicXML y MIDI. El audio puede ser exportado en formato AIFF o MP3 a partir del MIDI asociado al documento. La partitura puede exportarse en EPUB (para visualización en dispositivos electrónicos), MusicXML o SmartMusic.

Su competidor directo es Sibelius y entre los dos son responsables de la mayoría de las partituras en tinta publicadas hoy en día. La principal diferencia radica en que Finale se centra en maximizar la funcionalidad del programa a cambio de sacrificar cierta facilidad de uso mientras que, en comparación, Sibelius prioriza la facilidad de uso a cambio de no ofrecer tanta funcionalidad (DanLisMusic, s.f.).

### 2.5.5. Manufaktura

Manufaktura.Controls es una biblioteca de notación musical de código abierto para web, escritorio y móvil. Ofrece controles para el grabado de música en ASP.NET MVC y muchas plataformas diferentes (Salamon, s.f.).

Los controles Manufaktura son compatibles con la especificación de SMuFL (*Standard Music Font Layout*). SMuFL es una especificación de código abierto que proporciona una forma estándar de asignar a cada uno de los miles de símbolos musicales requeridos por la notación musical convencional un código en el Área de Uso Privado en el Plano Básico Multilingüe de Unicode para una única fuente independiente del formato (W3C Music Notation Community Group, s.f.). Esto permite cambiar la fuente musical.

Manufaktura proporciona métodos para importar partituras desde MusicXML, representarlas gráficamente como partituras en tinta, hacer pequeñas modificaciones de forma gráfica en el pentagrama y reproducir partituras vía MIDI.

Manufaktura soporta gran variedad de plataformas:

- Escritorio de Windows: WPF, UWP, Avalonia y WinForms
- Otro escritorio (Mac, Linux): Avalonia
- Web (servidor): ASP.Net MVC, ASP.Net Core
- Web (cliente): Blazor
- Móvil (Android, IoS, Windows): Xamarin Forms con Skia Sharp, Xamarin Forms con renderizador nativo (sólo Android)
- Plataformas Legacy: Silverlight, Windows 8, Windows Phone 8 Silverlight

### 2.5.6. vdaron/MusicXml.Net

MusicXml.Net de vdaron es una biblioteca que proporciona métodos que permiten analizar archivos MusicXML y pasarlos a una estructura de árbol de MusicXML. Tiene sus repositorios públicos en GitHub (Daron, s.f.). Permite organizar jerárquicamente la información de un archivo MusicXML para poder después trabajar con ella.

### 2.5.7. Audiveris

Audiveris es una aplicación de código libre que permite transcribir música en formato MusicXML. Tiene sus repositorios públicos en GitHub (Bitteur, s.f.). La entrada es mediante imagen y soporta varios formatos diferentes incluyendo archivos JPG, PNG, TIFF y PDF. Este programa es por tanto un sistema de OCR (Reconocimiento Óptico de Caracteres). En el programa, puede seleccionar entre diferentes modelos de visualización como lógico o físico, o una combinación de los dos.

# Capítulo 3

# Metodología de trabajo

## 3.1. Metodología de trabajo / Gestión del proyecto

Nuestro equipo se ha organizado de manera *descentralizada democrática*. Así, la comunicación ha sido horizontal y las decisiones se han tomado por consenso (no ha habido un jefe permanente).

### 3.1.1. Metodología de desarrollo software

La metodología de desarrollo de software utilizada ha sido la metodología *Scrum*. Nos hemos basado en iteraciones que daban lugar a una aplicación que funcionaba, la cual era ampliada en cada iteración.

Scrum tiene distintas ventajas, de las cuales destacan que la planificación es flexible a cambios (algo muy favorable debido a las circunstancias ocasionadas por el estado de alarma decretado a causa del COVID-19), que el tiempo de desarrollo es predecible (por estar basado en iteraciones, que son más sencillas de calcular).

Esta metodología nos pareció la más adecuada porque nos facilitaba la comunicación con el cliente (los directores de este Trabajo de Fin de Grado) para saber cuál era el siguiente avance a realizar.

Al comienzo de cada iteración, planificábamos cuál era el objetivo de la iteración (por ejemplo, mostrar la partitura en tinta). Diariamente, teníamos una reunión para situar los avances del día anterior y concretar el trabajo a realizar. Al final de la iteración, revisábamos el desarrollo realizado y, posteriormente, se lo entregábamos al cliente para comenzar la siguiente iteración.

### 3.1.2. Control de versiones

Para el control de versiones hemos usado *Git*, que es un sistema distribuido. Esto nos permitía realizar de forma paralela distintos avances de una iteración (en distintas ramas) para, una vez conseguido finalizar cada avance, incluirlo en la rama principal del proyecto.

Además, al tener cada integrante una réplica local, nos permitía realizar de diversas formas un mismo objetivo de forma coetánea para, posteriormente, seleccionar aquel que considerásemos más favorable.

### 3.1.3. Planificación temporal

En la figura 3.1, podemos ver el diagrama de Gantt del proyecto. Se utiliza un código de colores, el cual se basa en tareas y subtareas y en la modificación de objetivos explicada en la sección 1.2.2.

El color azul se corresponde con las tareas que habían sido planificadas antes de comenzar a desarrollar el proyecto y que se han completado. En amarillo aparecen las subtareas de estas tareas realizadas. En color rojo aparecen las tareas que se planificaron pero no fueron realizadas (como se explica en la modificación de objetivos, sección 1.2.2). Por último, el color verde se utiliza para aquellas tareas que no fueron planificadas inicialmente pero sí se desarrollaron finalmente en el proyecto.

## 3.2. Tecnologías usadas

Para elaborar nuestro proyecto hemos utilizado diversas herramientas de software. Tras una primera discusión con nuestros tutores decidimos que desarrollaríamos el proyecto en el lenguaje C# dado que facilita la implementación de aplicaciones accesibles y es compatible con las herramientas que se utilizan hoy en día en accesibilidad.

### 3.2.1. Wpf en Visual Studio

Durante la carrera nos hemos familiarizado mucho con el entorno de desarrollo Visual Studio así que tomamos este como punto de partida para nuestro proyecto. A la hora de comenzar el proyecto, decidimos que este fuera un proyecto de Wpf (Windows Presentation Foundation). Los proyectos Wpf proporcionan una plantilla de proyecto muy útil para desarrollar aplicaciones de escritorio. Separa la parte visual del programa de la parte funcional como se hace también en WinForms o en otros tipos de proyectos parecidos. El lenguaje que se utiliza Wpf para la parte visual es XAML (Extensible Aplication Markup Language) y en la parte funcional (code-behind) se utiliza C#. Además los proyectos Wpf son proyectos que se desarrollan sobre .NET que es un marco de desarrollo software creado por Microsoft para facilitar la compatibilidad de los programas con los distintos sistemas y para permitir

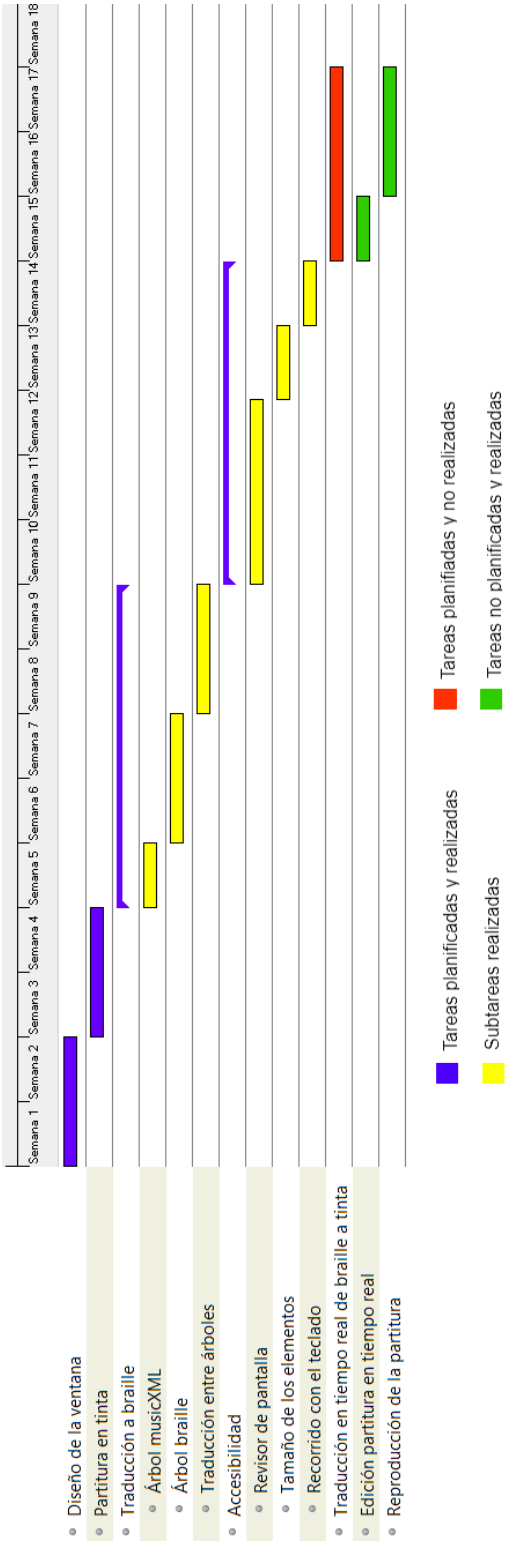



Figura 3.1: Planificación temporal.

una mayor compartición de herramientas software entre los desarrolladores de las distintas plataformas que utilizen .NET.

### 3.2.2. C#

C# fue diseñado como un lenguaje de programación orientada a objetos (como Java y muchas otras) basado en C. Dentro de este tipo de lenguajes, una utilidad muy importante es el polimorfismo de clases. Gracias a este podemos crear una estructura jerárquica con interfaces y diferentes clases en nuestro proyecto. Además hemos realizado un desarrollo modular del programa para permitir un cómodo punto de partida para trabajo futuro. Otra razón importante para modularizar nuestro programa es que crear partes diferenciadas nos permite seguir una buena metodología de proyecto poniendo como objetivos la finalización de módulos completos. Por último, tener diferentes módulos facilita corregir, modificar y mejorar las diferentes diferentes partes de nuestro programa.

El desarrollo del lenguaje C# está ligado a la creación de .NET y por esta razón podemos hacer uso de su facilidad a la hora de utilizar herramientas de software desarrolladas por terceros. Así como en otros lenguajes sería necesario instalar librerías externas (DLL's) en tu proyecto, trabajando sobre .NET podemos hacer uso de paquetes NuGet para lograr el mismo objetivo. Los paquetes NuGet son, en resumen, paquetes de código compilado que se publican en nuget.org para el consumo de cualquier usuario que utilice .NET. Visual Studio en Windows cuenta con un administrador de paquetes NuGet que facilita enormemente la búsqueda, descarga e instalación de dichos paquetes en el proyecto.

### 3.2.3. XAML y sus Componentes

XAML (Extensive Aplicaton Markup Language) es un lenguaje declarativo basado en XML que utiliza una jerarquía por nodos para definir los diferentes objetos o componentes que se presentarán en la pantalla de un proyecto en Wpf. El nodo raíz es la ventana en la que se presenta el programa y dentro de ella se van incluyendo los diferentes componentes (o controles) que pueden ser botones, textos, layouts, etc. Al igual que en C#, existen paquetes NuGet para XAML para hacer uso de controles diseñados por otros usuarios.

### 3.2.4. Unión entre las dos partes (*Binding*)

Una vez conocemos las dos partes de un proyecto en Wpf, falta por determinar cómo enlazarlas para que la ventana muestre elementos de nuestro código y viceversa, que nuestro código pueda obtener información de la ventana. Para ello hay diferentes formas de comunicación que varían según el uso que se quiera hacer. Las dos más importantes, que utilizamos en nuestro proyecto son el paso de eventos y el *binding* (enlazado). El paso de eventos consiste en lanzar una señal (evento) cuando se realiza una acción pre-definida, esta señal será capturada por un controlador que está esperando y el controlador ejecuta la función deseada para dicho suceso. En resumen, los eventos nos permite realizar acciones en respuesta a

sucesos en la ventana. En nuestro caso, por ejemplo, nos permite controlar lo que sucede cuando se selecciona alguna opción del menú.

En cuanto al *binding* (o enlazado) podemos decir que nos permite la comunicación de información entre la parte visual y el código. Es posible obtener el valor en la ventana de una variable declarada en el código e, incluso, es posible hacerlo en el otro sentido sin hacer uso de ninguna herramienta adicional. El problema surge en que la consulta de esa información se hace de forma estática. Esto provoca que cada vez que la información cambie, es necesario realizar una nueva consulta y esto puede complicar el programa. Como solución, se ha desarrollado la utilidad del *binding*. El *binding* se hace sobre variables que tienen lo que se denomina como propiedades de dependencia que indican cuando el estado de una variable cambia. Una variable enlazada (*binded*) a otra con propiedades de dependencia, se actualizara automáticamente tras el cambio de estado de esta segunda. El *binding* se puede realizar en los dos sentidos, por separado o, incluso, a la vez. En nuestro proyecto esta funcionalidad es útil, por ejemplo, a la hora de modificar el tamaño de los elementos.

# Capítulo 4

# Desarrollo de la aplicación *LiveDots*

## 4.1. Diseño inicial

Después del análisis realizado sobre aplicaciones de edición de partituras y de traducción a braille pasamos a pensar en un primer diseño para nuestra prueba de concepto. Hemos identificado los componentes que más se repiten en otras aplicaciones similares y hemos escogido los que nos parecían útiles para darle funcionalidad completa a nuestra aplicación.

### Panel visualización partitura en tinta

El objetivo principal de nuestra aplicación es facilitar la interacción entre vidente e invidente, por tanto es necesario incluir funcionalidades y componentes que les permitan a ambos comprender la partitura e interactuar con la aplicación. En todos los diseños de editores o visualizadores de partituras para videntes que hemos analizado, la representación gráfica de la partitura es el componente principal ocupando la mayor parte de la pantalla. Para representar la partitura de forma gráfica se usa un pentagrama, que es una notación musical internacional. En nuestro diseño, decidimos incluir este panel. Sin embargo, en lugar de ocupar la totalidad del espacio de trabajo, decidimos que el tamaño fuese aproximadamente de la mitad de la pantalla para poder incorporar los siguientes elementos.

### Salida por línea braille

Este es un componente no gráfico de la aplicación. El código braille correspondiente a la partitura que se está visualizando en pantalla, se muestra simultáneamente a través de una línea braille. Esto permite a personas invidentes o con discapacidad visual alta ser

capaces de leer la partitura de la forma que les resulta más natural. Aunque también se pueda escuchar la lectura de la partitura mediante el revisor de pantalla, este audio puede resultar engorroso y poco intuitivo a la hora de comprender la música.

### Panel visualización braille

Otra característica importante que hemos encontrado en aplicaciones musicales adaptadas a personas con discapacidad visual es la visualización del braille por pantalla expresada en forma de cajetines braille. Tenemos un ejemplo de esto en Braille Music Editor (Giuseppe Paccini, s.f.). Hemos incluido este componente porque facilita la interacción entre vidente e invidente. Por ejemplo, un profesor vidente que sabe braille puede ver en tiempo real lo mismo que el alumno está leyendo mediante la línea braille. También puede ser útil para personas con discapacidad visual que tengan dificultades a la hora de leer un pentagrama en tinta. Los cajetines braille tienen una estructura muy simple y se pueden reconocer mejor que ciertos símbolos musicales en tinta. Espacialmente este componente debe ser suficientemente grande para permitir una lectura fluida y cómoda, será nuestro segundo componente principal.

### Menú

Por último decidimos incorporar un menú que recogiese las funcionalidades principales de la aplicación, como abrir un archivo MusicXML. Se puede recorrer mediante el ratón o mediante el teclado para que, usando un revisor de pantalla adecuado, sea totalmente accesible.

### Disposición

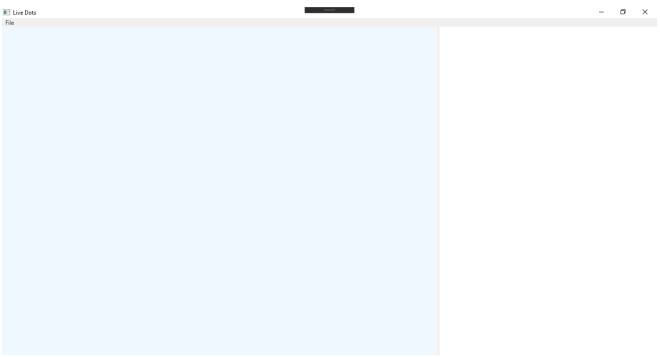



Figura 4.1: Primer diseño de LiveDots

Una vez escogidos los componentes principales de la parte gráfica (menú, partitura en tinta y partitura en braille) pasamos a su ordenación espacial. Situamos el menú en una barra en la parte superior de la ventana, por convenio. Dividimos el resto de la pantalla

de forma vertical en dos espacios donde irán las representaciones de la partitura en tinta a la izquierda y en braille a la derecha (figura 4.1). El espacio designado a la partitura en tinta es ligeramente mayor, ya que un pentagrama ocupa más espacio que una secuencia de cajetines braille.

## 4.2. Visualización de la partitura en tinta

Una vez elegido y programado el diseño general de nuestra aplicación, el objetivo era, dado un archivo en formato MusicXML, obtener la partitura en tinta correspondiente.

Una posibilidad habría sido desarrollar la implementación por nosotros mismos. Esto habría requerido la implementación gráfica de un pentagrama, de las distintas claves, de las distintas notas y sus duraciones y de las alteraciones, entre otras muchas cosas. Considerando que no era el objeto de este trabajo, decidimos buscar código que ya implementara esta funcionalidad.

Encontramos la biblioteca Manufaktura (sección 2.5.5), que es de código abierto. Esta biblioteca permite importar archivos en formato MusicXML y mostrar la partitura en tinta correspondiente. De esta manera, decidimos introducir esta biblioteca a nuestro proyecto.

Así, en el panel izquierdo de nuestra aplicación, introducimos un control de tipo *ManufakturaControls:NoteViewer*, el cual permite visualizar la partitura. Este control tiene distintos atributos (como los márgenes o la posición horizontal y vertical), siendo el más importante *XMLSource*, que es un atributo de tipo *string* que contiene el texto MusicXML del que pretendemos obtener la partitura en tinta.

Con unas primeras pruebas (pasando el texto MusicXML directamente al atributo *XMLSource*) vimos que la partitura se mostraba perfectamente. Después, queríamos que el control de tipo *ManufakturaControls:NoteViewer* mostrara una partitura en tinta de un MusicXML que pudiéramos seleccionar desde la aplicación. Para ello, creamos un botón de *abrir en el menú*, el cual, al ser pulsado, permite seleccionar un archivo de formato MusicXML que tengamos en nuestro ordenador. Una vez seleccionado el archivo, la variable *sourceXml* toma el valor del texto del archivo MusicXML seleccionado. Después, hacemos un binding (cuyo funcionamiento hemos visto en la sección 3.2.4) entre esta variable y el atributo *XMLSource*. Así, tenemos que, cada vez que se abra un archivo MusicXML en la aplicación, la variable *sourceXml* se actualiza con el texto de dicho archivo y el atributo *XMLSource*, al estar *binded* a esta variable, toma también su valor, y por tanto se actualiza la partitura en tinta que se muestra por pantalla a la correspondiente al archivo abierto. En la figura 4.2 se observa el resultado de la visualización de la partitura en tinta.

Además, este control de la biblioteca Manufaktura ( *ManufakturaControls:NoteViewer*), permite la edición de la partitura. Para ello, dispone de un atributo llamado *InnerScore*, el cual almacena el archivo MusicXML. Cuando se modifica una nota en la partitura en tinta (mediante el uso del ratón, haciendo que esta se desplace en verticalmente), se modifica ese atributo, de manera que la partitura en tinta queda modificada. Además, a partir del atributo *InnerScore*, podemos obtener el texto MusicXML en formato *string* asociado a la partitura en tinta modificada. En principio, no íbamos a hacer uso de esta funcionalidad

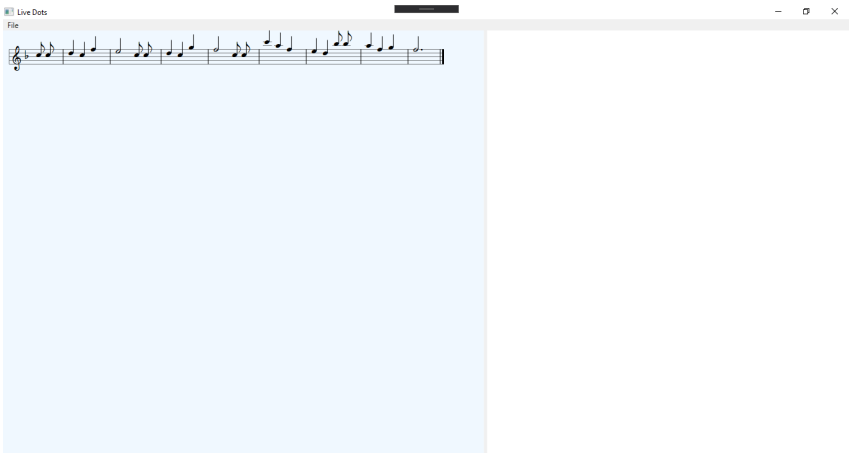



Figura 4.2: Vista de la partitura

pero, como explicamos en la sección 4.8, al final sí lo hicimos.

### 4.2.1. Visualización del texto MusicXML

Por otro lado, para empezar a usar la aplicación, hicimos que el texto MusicXML del archivo abierto se mostrara a la derecha de la partitura, en el panel que posteriormente correspondería al texto braille asociado a la partitura. Para ello, creamos un *TextBox* (el cual configuramos como no editable) cuyo valor también asociamos mediante un *binding* a la variable *sourceXml*.

De esta manera, al abrir un archivo MusicXML, se mostraba, en el panel derecho de la aplicación, el texto MusicXML de dicho archivo. Esto ocurría al mismo tiempo que se mostraba la partitura en tinta, por lo que teníamos ambas representaciones a la vez, tal y como se observa en la figura 4.3.

## 4.3. Traducción de MusicXML a Braille

Una vez conseguido que se muestre la partitura en tinta en el panel de la izquierda y el texto MusicXML correspondiente en el panel de la derecha, el objetivo era lograr que, en lugar del texto MusicXML, en el panel derecho se mostrara el texto en braille correspondiente a la partitura en tinta. Para ello, debíamos pasar del texto MusicXML asociado a la partitura al texto braille y necesitábamos poder representar el texto MusicXML y el texto braille de manera que pudiéramos hacer la traducción de uno a otro. En la figura 4.4, podemos ver un esquema de esta traducción.

Lo primero fue estudiar la representación de una partitura en MusicXML y en braille. En MusicXML, al tratarse de un formato basado en XML, tenemos una estructura en forma

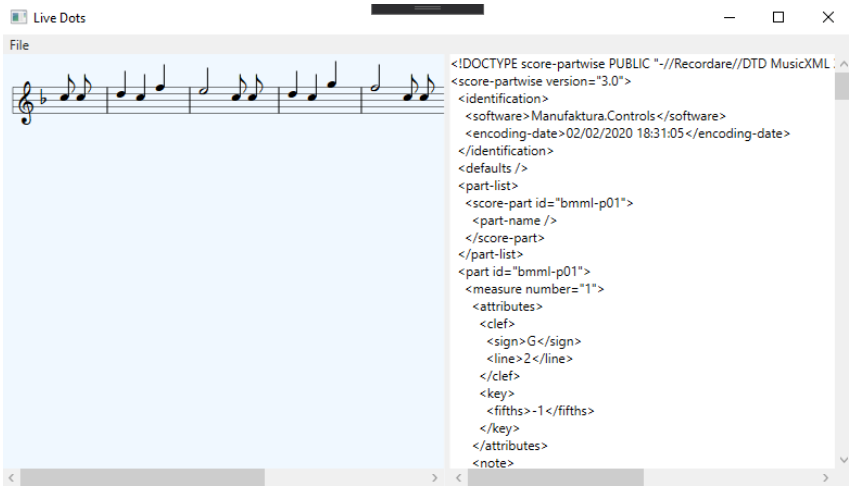



Figura 4.3: Vista de la partitura y el texto MusicXML asociado

de árbol. El caso del braille es más complejo, ya que no se distingue una estructura clara con facilidad. Sin embargo, tras un tiempo de investigación, encontramos *A Transcription System from MusicXML Format to Braille Music Notation* Gotoh y col., 2008, un paper japonés en el que habían realizado un estudio teórico para realizar esa traducción de MusicXML a braille, en el que se explicaba la estructura (también en forma de árbol) de la partitura en braille.

Así, nuestra idea fue crear ambos árboles, de manera que podríamos pasar del texto MusicXML a árbol MusicXML, de este árbol al árbol braille y, por último, del árbol braille al texto braille. Además, estas representaciones permiten también ser utilizadas para el recorrido inverso, de manera que se podrían utilizar para programar el paso de texto braille a texto MusicXML (y, con esto, a la partitura).

### 4.3.1. Árbol MusicXML

Para el diseño del árbol, nuestra idea inicial era crear una clase para cada tipo de elemento de un texto MusicXML. Así, tendríamos una clase *MusicxmlScore*, la cual representaría la partitura al completo. Para ello, contendría una lista de *MusicxmlPart*. Cada *MusicxmlPart* estaría compuesto por una lista de *MusicxmlMeasure*, que son los compases de esa parte. Cada uno de los compases tendría por un lado un *MusicxmlAtribute* y por el otro las distintas *MusicxmlElement*. El *MusicxmlAtribute* contendría las divisiones, el tono, el tempo y la clave. Un *MusicxmlElement* puede ser un *MusicxmlNote* o un *backward* o *forward*. Por último, cada *MusicxmlNote* contendría información sobe su tipo, su duración, su alteración, su tono, su voz, su mano (izquierda o derecha, para el piano, por ejemplo) y si es o no un silencio. En la figura 4.5, podemos ver un esquema básico de la estructura en forma de árbol que pretendíamos obtener a partir del texto MusicXML.

Sin embargo, encontramos que en el github de Vincent Daron había una biblioteca con

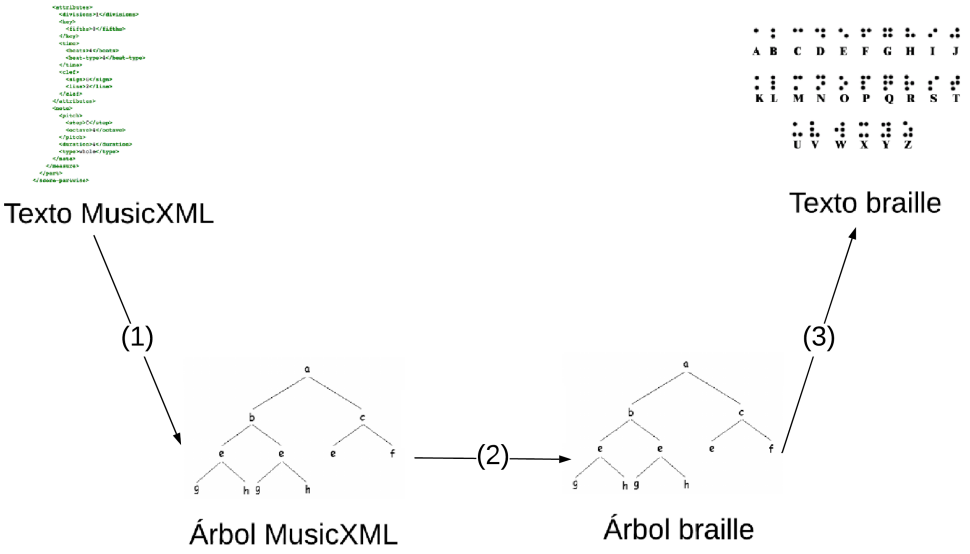



Figura 4.4: Esquema de traducción de MusicXML a braille

liencia libre llamada MusicXML.Net Daron, s.f. Esta biblioteca permitía, a partir de un texto musicXML, crear el árbol MusicXML asociado. El árbol lo devuelve en una variable que podemos recorrer de la misma manera que si hubiéramos creado las clases antes mencionadas, es decir, siguiendo la estructura árborea que pretendíamos diseñar.

De esta manera, decidimos incluir la biblioteca a nuestro proyecto, algo que nos facilitó el trabajo por la alta cantidad de clases que nos evitó crear y porque ya incluía el paso desde el texto en MusicXML al árbol MusicXML, es decir, nos proporcionaba el trabajo correspondiente a la flecha (1) de la figura 4.4.

### 4.3.2. Árbol Braille

Este árbol no es igual que el árbol MusicXML (algo que habría facilitado enormemente el trabajo). En este caso, sí creamos las clases necesarias para representar la estructura musical en braille.

Comenzamos creando la clase *BrailleScore*, que contiene una lista de *BraillePart*. Esta clase contiene a su vez una lista de *BrailleMeasure*, y cada uno de ellos contiene su *BrailleAttribute* y sus *BrailleStaff* (que, como se ha señalado, distinguen una mano de la otra en instrumentos como el piano). Cada *BrailleStaff* está compuesto por una lista de *BrailleVoices*, los cuales contienen las *BrailleNote*. Cada *BrailleNote* conteiene información sobe su tipo, su duración, su alteración, su tono, su voz, su mano (izquierda o derecha, para el piano, por ejemplo) y si es o no un silencio.

Podemos ver en la figura 4.6 cómo es la estructura básica del árbol braille. Además, comparando dicha figura con la figura 4.5, observamos las diferencias existentes entre la

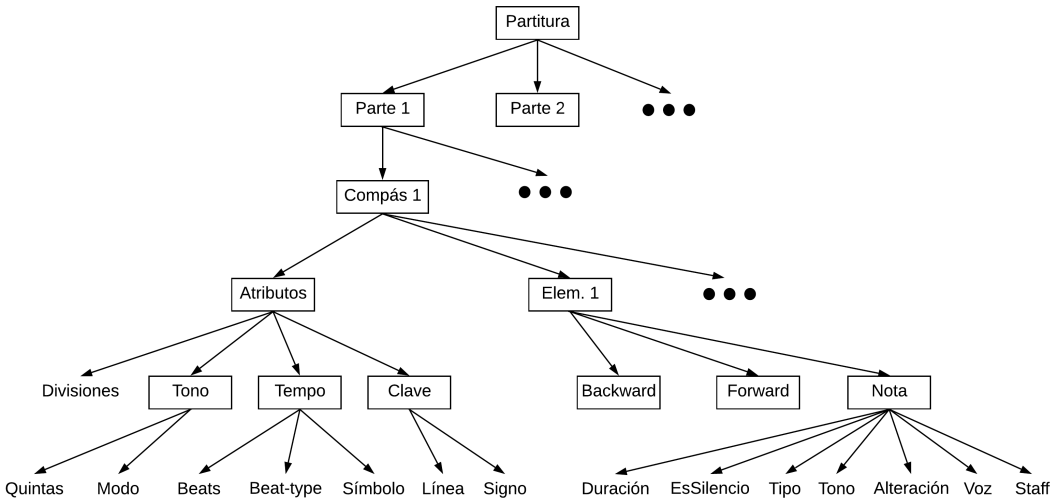



Figura 4.5: Árbol de MusicXML

estructura de la representación musical en formato MusicXML y en braille.

### 4.3.3. Traducción de Árbol MusicXML a Árbol Braille

Una vez se dispone de una forma de representación de la estructura del texto MusicXML y otra forma para la estructura del texto braille, debíamos conseguir ser capaces de pasar de la primera a la segunda, es decir, el objetivo en este momento era completar la flecha (2) de la figura 4.4. Así, cada vez que se modifique un MusicXML, al abrirlo con nuestro programa será capaz de traducir ese MusicXML para que (una vez implementado el paso de árbol braille a texto braille) se pueda visualizar la partitura en braille.

Para ello, creamos una clase llamada *Converter*. Su método principal (al que se llama cuando se abre una nueva partitura) es *treeXml2Braille*, el cual transforma un árbol MusicXML (obtenido por la traducción del texto MusicXML mediante la biblioteca MusicXML.Net) en un árbol braille. Este método comienza recorriendo las partes de la partitura del árbol MusicXML y, para cada parte, recorre los compases. Si nos fijamos en los árboles de MusicXml (figura 4.5) y de braille (figura 4.6), por lo que la idea es recorrer cada compás para ir haciendo la traducción y, finalmente, añadir el compás entero traducido al árbol braille.

La traducción de cada compás la realizamos llamando al método auxiliar *Xml2attribute*. Este método copia el tono, la clave y el tempo del compás del árbol MusicXML para ponerlo en el árbol braille (no así las divisiones, que en el árbol braille no aparecen). Además, si el compás tiene armadura, se copia para poder ajustar las alteraciones de las notas y, si no la tiene, se copia la última armadura.

Una vez traducidos los atributos, procedemos a recorrer los elementos del compás. En

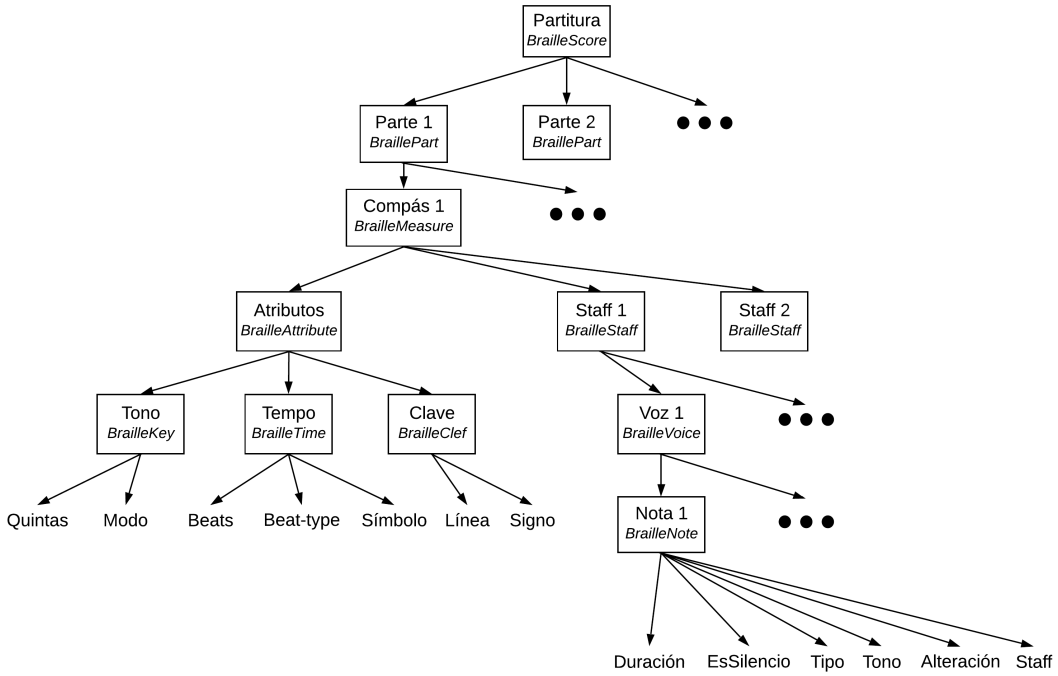



Figura 4.6: Árbol de braille

el caso de que el elemento sea un backup, a la variable *temp* (que representa el tiempo actual de la partitura, comenzando en cero y aumentando siempre según la duración de cada nota) le restamos el valor indicado en dicho backup y, en el caso de que el elemento sea un forward, le aumentamos el valor indicado en dicho forward. Estos valores de backup y forward permiten tocar notas al mismo tiempo que otras, ya que se puede volver a una posición temporal correspondiente a una nota para añadir otra que sea tocada al mismo tiempo.

La última posibilidad es que el elemento sea una nota. En ese caso, primero calculamos sus alteraciones teniendo en cuenta la armadura del compás mediante la función *GiveAccidental* (de la clase *BrailleNote*). Después, en el staff marcado por la nota, seleccionamos la voz correspondiente (atributos que pertenecen a la nota en el árbol MusicXML pero que en el árbol braille también son nodos por encima de ella) y añadimos la nota. De esta manera, finalmente nuestro compás del árbol braille tiene staffs, que están compuestos por voces, las cuales contienen las notas. Por último, señalamos que la última nota es esta (ya que si una nota y la anterior difieren lo suficiente, hay que señalar en braille la octava de la segunda) y sumamos a la variable *temp* la duración de la nota.

Así, una vez terminado cada compás del árbol MusicXML, lo añadimos a nuestro árbol braille. Se consigue así pasar de una estructura a otra. Además, destaca que el diseño de esta traducción es bastante modular (dentro de lo posible), haciendo uso de distintas funciones auxiliares. Esto permite hacer comprobaciones y mejoras con una mayor facilidad, algo reseñable teniendo en cuenta lo confusa que puede resultar la traducción entre estos dos

árboles.

### 4.3.4. Traducción de Árbol Braille a Braille

Una vez tenemos el árbol de braille, queremos traducirlo a un texto en cajetines braille que podamos mostrar al usuario (paso (3) de la figura 4.4).

En primer lugar, nos planteamos cómo representar cajetines braille. Necesitábamos un método que nos permitiese tanto una salida por pantalla legible como un formato compatible con línea braille. Tras un primer proceso de investigación, descartamos la idea de diseñar un sistema de representación nosotros mismos por dos motivos. En primer lugar podría ser complicado lograr compatibilidades con distintas líneas braille y, debido a que es un elemento del cual íbamos a tener disponibilidad limitada (finalmente nula), decidimos optar por una solución más sencilla. En segundo lugar éste es un problema lo suficientemente general como para que alguien se haya enfrentado ya a él y desarrollado una solución.

**Fuente *edico_es_br6.tt*: de código ASCII a cajetín braille**

Efectivamente, la representación de los cajetines braille suele hacerse mediante un archivo de fuente TrueType, que no es más que una fuente de texto. Nosotros hemos usado la fuente *edico_es_br6.tt*, la cuál hemos obtenido gracias a la ONCE. Esta fuente se diseñó para el proyecto Edico que, como ya hemos comentado, es un programa de notación matemática en braille. Al usar la fuente nos encontramos con dos problemas; en primer lugar la fuente asocia cajetines braille a distintos caracteres ASCII, pero al no saber braille, no estábamos familiarizados con dicha correspondencia, y en segundo lugar, al ser una fuente desarrollada para notación matemática, tenía algunas diferencias con la notación braille estándar en lo que a caracteres numéricos se refiere.

***DicBraille*: de strings de números a código ASCII**

Solucionamos ambos problemas desarrollando un diccionario (*DicBraille.cs*) que define una correspondencia entre los caracteres ASCII correspondientes a los cajetines braille mediante la fuente TrueType y un string que represente qué puntos están presentes en dicho cajetín. Por ejemplo, la entrada `dic.Add("1245", "g");` del diccionario significa que al string "1245" le asociamos el código ASCII que representa el carácter "g". A su vez dicho código ASCII se representa en la fuente *edico_es_br6.tt* como el cajetín braille que tiene los puntos 1, 2, 4 y 5 resaltados (figura 4.7).

De este modo, trabajamos con cadenas de texto de números, formato que nos resulta más intuitivo para representar cajetines braille. Para realizar la traducción de cadena de números al carácter correspondiente al cajetín braille que representa dichos números usamos el método *Num2Braille* implementado en la clase *Translator* que crea un objeto *DicBraille* y, dada una lista de strings de números representando una partitura en braille, los traduce uno a uno usando dicho diccionario.

***ParseBraille*: de árbol braille a lista de strings de números**

Ahora queremos pasar de árbol braille a lista de strings de números. Para ello recorremos

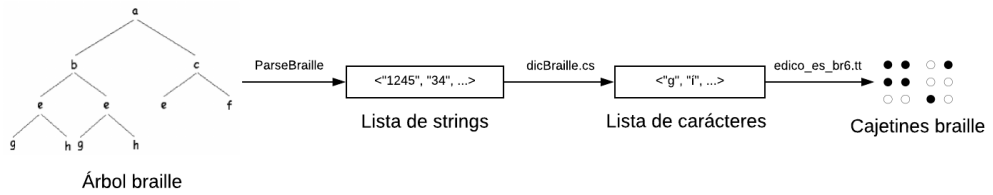



Figura 4.7: Esquema de traducción de árbol de braille a braille

el árbol braille llamando a la función *ParseBraille*. Esta función pertenece a la interfaz *BrailleRepresentation* que implementan todos aquellos elementos que componen el árbol braille.

Para realizar la traducción llamamos al método *ParseBraille* del objeto *BrailleScore*, ya que es el que recoge todo el árbol braille por ser la clase más alta en la jerarquía. Recordamos que un elemento de la clase *BrailleScore* tiene una lista, *Parts*, de elementos de clase *BraillePart* que representan las distintas partes de un pentagrama (figura 4.6). Así, el método *ParseBraille* de un elemento *BrailleScore* llama a su vez al método *ParseBraille* de cada uno de los elementos de la clase *BraillePart* que tiene en la lista *Parts*. De forma similar, se va llamando al método *ParseBraille* de cada uno de los elementos del árbol haciendo un recorrido en profundidad del mismo.

Las hojas son los elementos que tienen representación directa en braille. Al llegar a ellos, devuelven una lista de strings de números que representa el código braille correspondiente. También los elementos no hoja pueden añadir elementos a la lista de strings de números para indicar distintos elementos de notación musical. Por ejemplo, la función *ParseBraille* de un elemento *BrailleMeasure* (que representa un compás) después de llamar a los métodos *ParseBraille* de sus atributos añade un cajetín braille en blanco (sin puntos resaltados) que representa el cambio de compás. Otro ejemplo sería la función *ParseBraille* de un elemento de la clase *BrailleScore* que, tras añadir a la lista de strings de números los códigos devueltos por las funciones *ParseBraille* de los elementos de la lista *Parts*, añade unos cajetines braille que representan el final de partitura.

## 4.4. Mejora de diseño

Después de terminar la traducción de texto MusicXML a texto Braille, nuestra aplicación mostraba el texto braille en el panel de la derecha. Dado que en el diseño inicial no se mostraba el texto braille, no sabíamos exactamente cómo sería visualmente. Por ello, decidimos plantear un posible cambio de diseño, ya que, como en ese momento se visualizaban todos los componentes visuales de la aplicación, tenía sentido decidir la distribución final de la misma.

De esta manera, decidimos cambiar la distribución del *panel de visualización de partitura en tinta* y del *panel de visualización braille*. El primero, que estaba situado a la izquierda,

paso a ocupar la parte superior, y el segundo, que estaba situado en la parte derecha, paso a ocupar la parte inferior. Por tanto, en lugar de estar separados verticalmente, ahora los paneles están separados horizontalmente.

Este cambio de diseño resulta favorable ya que la visualización de la partitura es mayor (evitamos el desplazamiento horizontal). También mejora la representación del texto braille, ya que una línea de cuarenta caracteres se puede visualizar sin necesidad de hacer desplazamiento horizontal. Este aprovechamiento del espacio de la aplicación lo podemos ver en la figura 4.8. Vemos también que el menú se mantiene en la posición superior de la aplicación.

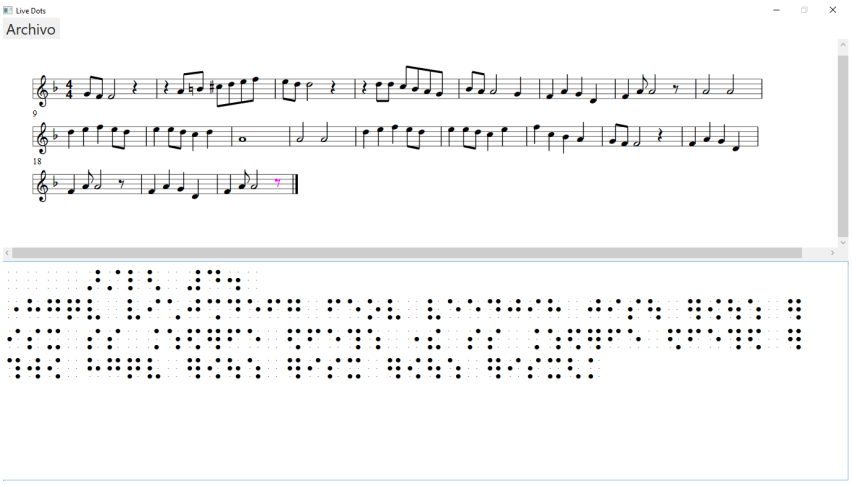



Figura 4.8: Diseño actualizado de LiveDots

## 4.5. Accesibilidad

Hacer que la aplicación sea accesible era uno de los objetivos principales de nuestro proyecto. Para ello, principalmente, tuvimos que asegurarnos de que la aplicación se pudiera utilizar completamente a través del teclado y que un revisor de pantalla informara adecuadamente de lo que sucedía dentro de la aplicación. Además para completar la accesibilidad de la aplicación se añadieron algunas funcionalidades como pueden ser cambiar el tamaño de los elementos, cambiar el contraste de colores de la aplicación o una guía de uso de la aplicación.

### 4.5.1. Revisor de Pantalla

Comenzamos a implementar la accesibilidad por el revisor de pantalla. Para que este funcionara correctamente tuvimos que implementar funcionalidades que leyeran tanto los textos del menú, como el contenido de las partituras. Dado que la partitura en tinta y la partitura traducida a braille representan el mismo contenido, decidimos que la información que leería el revisor de pantalla sería únicamente la de una de ellas, la partitura en braille.

#### 4.5.1.1. Elección del Revisor

El primer paso fue elegir el revisor de pantalla para el que íbamos a diseñar la aplicación. En nuestro caso, la decisión estaba entre JAWS y NVDA. La principal ventaja de NVDA es que no requiere licencia de pago para su uso. Esto nos facilitaría el testeo en nuestros ordenadores personales. Sin embargo, el uso de JAWS está más extendido y, además, mejor documentado. Tras valorar la decisión, nos decantamos por orientar el programa hacia JAWS. Esta elección tiene como inconveniente que para probar el correcto funcionamiento de la aplicación tendríamos que hacer uso de una versión de prueba de JAWS que te permite un uso de 40 minutos cada vez que reinicias el ordenador. A pesar de esto, nos pareció que el resultado final sería mejor al elegir JAWS.

#### 4.5.1.2. Configuración de JAWS

El programa de JAWS hace uso de Scripts para implementar sus funcionalidades. Los Scripts son documentos que contienen funciones en el lenguaje JavaScript. En este caso, JAWS utiliza una versión adaptada con la extensión de fichero *.jss* y utiliza su propio compilador para este tipo de ficheros.

Una vez el revisor de pantalla se está ejecutando, se van llamando a diferentes Scripts que JAWS tiene configurados que ejecutarán las funcionalidades necesarias. JAWS, en realidad, no utiliza los ficheros de Script directamente si no que hace uso de una versión en binario de estos (con la extensión *.jsb*) que es generada por su compilador. El revisor de pantalla tiene implementados *.jss* (y sus asociados *.jsb*) para las aplicaciones y funcionalidades habituales en un ordenador. Estas son funcionalidades generales que funcionan para cualquier aplicación, el problema aparece cuando se desea que JAWS funcione de forma diferente a la que tiene programada.

A la hora de que JAWS lea los menús de nuestra aplicación todo lo que está ya programado nos resultó muy útil ya que, por defecto, JAWS lee lo que es necesario leer de un menú. Sin embargo, para leer la partitura tuvimos que diseñar una forma tanto de recorrerla como de leerla que tuviera sentido para el usuario. Para hacer estos cambios sobre el uso habitual de JAWS fue necesario implementar un Script asociado a nuestro programa que indica a JAWS las acciones a tomar.

JAWS está diseñado para que cuando una aplicación reciba el foco en la pantalla, se ejecute el Script con el mismo nombre que la aplicación. Por tanto, nosotros nos tuvimos que asegurarnos que el Script tuviera el mismo nombre que el de nuestra aplicación. Además, es necesario que el Script *LiveDots.jss* (junto con su asociado *LiveDots.jsb*) se copie junto con el resto de Scripts de JAWS. Para lograr esto, creamos una función de configuración (*JawsSettings.CheckJawsInstalled*) en nuestro programa que se encarga de mover el *.jss* implementado y el *.jsb* compilado asociado a la carpeta donde el programa JAWS almacena habitualmente todos sus Scripts.

#### 4.5.1.3. Comunicación con la App

Preparando un esquema para conseguir que el revisor leyera los elementos de la partitura, nos encontramos con dos puntos importantes. El primer punto que tuvimos en cuenta fue el recorrido de la partitura (explicado en la sección 4.5.1.6). Esto no tiene una solución obvia ya que cada elemento musical se representa con, posiblemente, más de un cajetín braille. La solución que diseñamos fue un recorrido de la partitura braille saltando por el número de cajetines que ocupa cada elemento en vez de leer todos los cajetines uno a uno. De esta forma, al pulsar la flecha para avanzar, en vez de leer el siguiente cajetín que puede pertenecer al mismo elemento musical, se salta todos los cajetines del elemento y lee el siguiente elemento.

Por defecto JAWS lee los cajetines como el carácter que se utiliza para representarlo. Entonces, el otro punto importante a tratar fue, cómo conseguir que JAWS dijera el nombre de los elementos musicales en vez de lo que dice por defecto (explicado en la sección 4.5.1.5). Para dar solución a este problema, necesitábamos comunicar el Script de JAWS con nuestra aplicación para que esta le proporcionara el nombre de los diferentes elementos. Además, para resolver el recorrido mencionado antes, pensamos en un primer momento que también podríamos indicarle a JAWS desde la aplicación el número de cajetines a saltar cada vez que se avanzaba el cursor. Por tanto, tuvimos que estudiar cómo realizar esta comunicación entre el Script de JAWS y nuestra aplicación.

La forma intuitiva de resolverlo fue creando una instancia de un objeto de nuestra aplicación en el Script. Este objeto tiene funciones que, al llamarlas, se encargan de proporcionar la información necesaria. Sin embargo, dado que los Scripts y nuestra aplicación utilizan lenguajes de programación diferentes, esta comunicación no se podía hacer de forma directa. Para instanciar un objeto de nuestra aplicación desde una aplicación externa, fue necesario hacer uso de un objeto COM (Component Object Model, ver figura 4.9). Este objeto, se registra en el ordenador para que el Script conozca su existencia y pueda hacer uso de él. De esta forma, desde el Script, únicamente hay que instanciar el objeto COM y llamar a sus funciones. Para registrar el objeto COM en el ordenador, implementamos una función que se llama dentro de la función de configuración antes mencionada (*JawsSettings.CheckJawsInstalled*) para que se haga automáticamente. Además, decidimos que en vez de controlar las llamadas a nuestra aplicación directamente desde el objeto COM, haríamos una clase (*BrailleMusicViewer*) que se encargara de saber qué es lo que el revisor de pantalla tiene que decir según la posición en la que está el cursor.

#### 4.5.1.4. Implementación del Script

Para lograr que nuestro Script tenga la funcionalidad que deseamos, el manual de JAWS indica que una de las formas consiste en sobrescribir las funciones que están definidas por defecto en JAWS. La documentación de JAWS proporciona el nombre y una breve descripción de todas las funciones definidas por defecto. Para sobrescribir una de estas funciones, hay que crear una nueva definición de la función, con el mismo nombre, en el Script asociado a nuestra aplicación (*LiveDots.jss*). Así, siempre que JAWS vaya a llamar a una función, mientras nuestra aplicación tenga el foco, llamará a la nueva definición. En el caso de que la función no esté sobrescrita en el Script, se llama a la definición por defecto de la función.

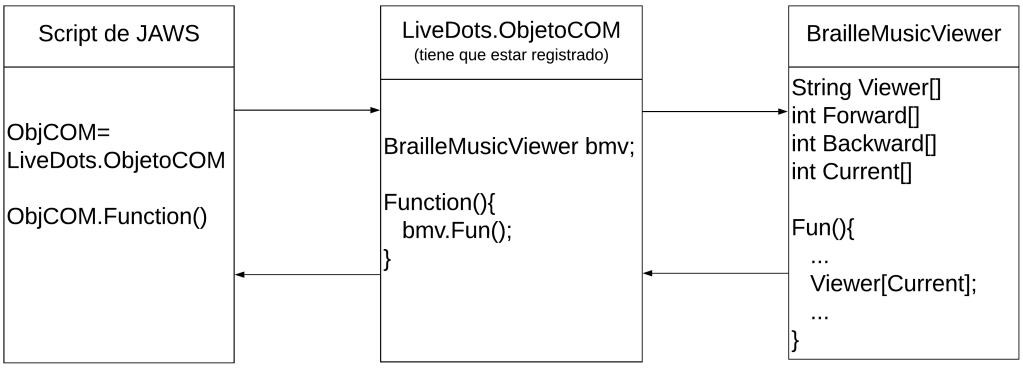



Figura 4.9: Esquema del proceso de comunicación entre el Script de JAWS y el programa LiveDots a través de un objeto COM

Dada esta información, buscamos la función más adecuada a sobrescribir y resultó ser la función *SayCharacter*. Esta función se se llama cada vez que se quiere leer un carácter. En nuestro caso, cada cajetín está representado por un carácter. Entonces, sobrescribimos la función *SayCharacter* para que, en vez de leer el carácter, leyera el nombre del elemento musical al que pertenece este carácter (explicado en la sección 4.5.1.5). En la misma función, después de leer el nombre del elemento, se avanza el cursor hasta el final del elemento. La información del nombre del elemento y el número de caracteres que tiene que avanzar viene proporcionada por nuestra aplicación. Hay que tener cuidado al sobrescribir las funciones de JAWS ya que, al hacerlo, se altera el funcionamiento habitual dentro de la aplicación en ejecución. Por tanto, está nueva funcionalidad solo la realizamos cuando el panel de visualización braille está seleccionado. Para obtener esta información también se hace una consulta a nuestra aplicación.

Controlar el movimiento del cursor desde el Script de JAWS resultó ser una mala idea. Esto fue porque, en el mismo instante que se llamaba a la función *SayCharacter*, se realizaba el movimiento del cursor con alguna otra funcionalidad (probablemente del propio sistema de Windows). Esto generaba muchas dificultades para obtener correctamente la posición actualizada del cursor y que avanzase la cantidad de cajetines deseada. Por tanto, decidimos buscar otra solución para el recorrido de los elementos y, únicamente, mantener la función *SayCharacter* sobrescrita para la lectura correcta del elemento. La nueva forma para recorrer los elementos se explica en la sección 4.5.1.6.

#### 4.5.1.5. Lectura de Elementos

La clase *BrailleMusicViewer* se encarga de saber el nombre del elemento que tiene que decir según la posición actual y cuánto hay que mover el cursor dependiendo del elemento que se esté saltando. Para poder tener toda esta información, guardamos tres listas y un entero con la posición actual (ver figura 4.10 con un ejemplo). La primera lista (*Viewer*) contiene lo que el revisor tiene que decir al leer el elemento. Dado que cada elemento ocupa, posiblemente, varios cajetines, guardamos el nombre del elemento una vez por cada cajetín

que ocupa. De esta forma, podemos averiguar fácilmente qué elemento de la lista *Viewer* es necesario leer según la posición del cursor. La segunda lista (*Forward*) contiene el número de cajetines que ocupa el elemento en esa posición hasta el siguiente elemento. Esto nos interesa porque no siempre será necesario saltar el elemento entero y con una simple consulta a la lista podemos averiguar cuánto hay que moverse adelante. La tercera y última lista (*Backward*) contiene información parecida a *Forward* pero, esta vez, cuánto tiene que saltar hacia detrás para llegar al final del elemento anterior. Hay que tener en cuenta que si consideramos los extremos donde podría situarse el cursor, el número de posiciones puede ser una más que el número de cajetines. Para que esto no cause problemas, añadimos un cero al final de la lista *Forward* y otro al comienzo de la lista *Backward*, así indicamos que no se salte al llegar a los extremos.

| Partitura |  | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Braille | ⠨ | ⠄ | ⠸ | ⠀ | ⠨ | ⠸ | ⠨ | ⠨ | |
| Viewer | Si bemol redonda | Si bemol redonda | Si bemol redonda | Espacio | Sol blanca | Sol blanca | Fa blanca | Fa blanca | |
| Forward | 3 | 2 | 1 | 1 | 2 | 1 | 2 | 1 | 0 |
| Backward | 0 | 1 | 2 | 3 | 1 | 1 | 2 | 1 | 2 |

Figura 4.10: Ejemplo con el contendido de las distintas listas de *BrailleMusicViewer* (no se incluye la traducción de la armadura por simplificar).

Una vez organizamos el contenido de cada lista, tuvimos que implementar varias funciones para generar las listas correctamente según la partitura que estuviera abierta. Las dos funciones principales son *AddElement* y *ParseText*.

*AddElement* se encarga de introducir los elementos dentro del *BrailleMusicViewer*. Es necesario pasarle el nombre del elemento y el tamaño del mismo. Introduce a *Viewer* el nombre del elemento el número de veces correspondiente al tamaño. Además, calcula los valores que hay que añadir a *Forward* y *Backward* para que el número a saltar en esas listas sea correcto.

*ParseText* es una función que tienen que implementar todos los elementos musicales en braille para indicar su nombre y el número de cajetines que ocupa para después llamar a la función *AddElement* con esos valores. Esta función necesita como atributo el *BrailleMusicViewer* donde se quiere añadir el nuevo elemento. Para el funcionamiento correcto, tuvimos que implementar todas las funciones *ParseText* para cada elemento donde, dependiendo de los valores del objeto actual, este tendría un nombre diferente o un tamaño diferente. Esta función sigue una idea muy parecida a la anteriormente mencionada *ParseBraille* pero, en vez de preocuparse de su representación en braille, se interesa por cómo se va a leer cada elemento. Así, al realizar una llamada a *ParseText* con el elemento raíz (*BrailleScore*) como atributo, se recorre el árbol con todos los elementos de braille introduciendo la información necesaria para que se lean correctamente.

#### 4.5.1.6. Recorrido de Elementos

Como ha sido mencionado antes, decidimos cambiar el recorrido de los elementos para que se hiciera desde nuestra aplicación en vez que desde JAWS. Esto, nos dimos cuenta, tiene sentido en el contexto de que es posible controlar el cursor directamente desde el *TextBox* (panel de visualización braille) de nuestra aplicación. Y, además, es interesante minimizar los cambios sobre la funcionalidad habitual de JAWS sobrescribiendo lo menos posible sus funciones.

La solución consistió en implementar una función que controlara el evento que se lanza cuando en el *TextBox* se mueve el cursor. En esta función, tenemos que comprobar si el movimiento se ha realizado hacia delante o hacia detrás. Esto lo hacemos comparando la posición actual del cursor con la posición que tenemos nosotros guardada en el *BrailleMusicViewer*. Una vez sabemos si hay que mover hacia delante o hacia detrás, movemos el cursor adecuadamente haciendo uso de las listas *Forward* y *Backward* que habíamos diseñado con este objetivo precisamente. Por último, actualizamos la posición actual en *BrailleMusicViewer*. Esta solución, nos permite incluso actualizar la posición actual en el caso de que se seleccione con el ratón, resolviendo así un posible problema añadido.

#### 4.5.1.7. Salto de Línea y Reorganización

Una vez el revisor de pantalla leía correctamente los elementos musicales, nos teníamos que preocupar por el resto de funcionalidades necesarias para que nuestra aplicación fuera accesible. Entre ellas, estaba dentro de nuestro plan, implementar la salida por línea braille. Para obtener una salida por línea braille tendríamos que realizar una división de las líneas por el tamaño de la línea braille. Esto además permitiría ver más cómodamente la partitura braille en la pantalla. Sin embargo, significaría varios cambios en nuestra aplicación. El primero en la forma en la que escribimos el braille por la pantalla y el segundo en la forma en la que el revisor lee los elementos.

Empezando por la forma en la que se muestra el braille, decidimos que la longitud de línea braille que tomaríamos de referencia sería 40 (la longitud más habitual de líneas braille). Además, pensamos que nuestra partitura podría saltar de línea, únicamente, entre elementos para que no partiera elementos por la mitad, lo que creemos que es más cómodo para su lectura. Con esto en mente, decidimos que la solución resultaba en crear una clase (*BrailleText*) que controlara el número de cajetines introducidos en la línea y metiendo un salto de línea en caso de que se llegara al tamaño máximo de línea. Esta clase pasaría a contener el texto que se representa en braille. Este cambio llevó a una modificación sobre la clase (*ParseBraille*) para que en vez de devolver la forma de escribir los elementos en braille directamente, utilizara una función (*AddText*) de *BrailleText* que introdujera un salto de línea si era necesario dependiendo del tamaño y la posición del elemento.

Pasando a la forma con la que el revisor lee los diferentes elementos, era necesario que también se indicara cuándo se ha llegado al final de línea ya que esto requeriría avanzar una vez más el cursor para pasar a la siguiente línea. Para resolver esto, hicimos algo parecido que en el caso del texto. Introdujimos el *BrailleMusicViewer* dentro de la clase *BrailleText* para que esta también pudiera hacer uso del tamaño de línea. De esta forma, igualmente,

había que cambiar las funciones *ParseText* para que utilizaran una función *AddViewer* que introdujera un elemento de final de línea en caso de haber llegado al tamaño máximo. Sin embargo, a parte de este cambio, se podían seguir utilizando el resto de funcionalidades del BrailleMusicViewer.

Al realizar este pequeño cambio, decidimos hacer un cambio más grande que solucionaba algunos problemas que surgían cuando la línea no tenía el tamaño de línea máximo. Esto ocurre cuando un elemento es más grande que el espacio restante de la línea. El cambio que hicimos fue sobre el recorrido del árbol. En vez de hacer el recorrido una vez para obtener el texto que se muestra con *ParseBraille* y otra para obtener lo que se dice con *ParseText*, se haría una única vez llamando a ambas funciones en cada elemento. Además las funciones *ParseBraille* y *ParseText* solo las implementarían los elementos musicales que necesitan representación (las hojas del árbol braille). De esta forma, ahorramos en el coste por el recorrido para cada partitura y solucionamos los pequeños problemas que estábamos teniendo. Ahora, cuando se realiza un salto de línea porque el elemento no cabe en la línea en la función *ParseBraille*, directamente se introduce también en el *BrailleMusicViewer* la información para que lea el final de línea. Después, al llamar a *ParseText*, puesto que el salto de línea ya se ha introducido, no hay que preocuparse por el tamaño de línea. Por último para poder realizar este recorrido del árbol que llama a las funciones *ParseBraille* y *ParseText* en las hojas, creamos la función *Parse* que tienen que implementar todos los elementos braille. La función *Parse* en las hojas, simplemente, llama a las dos funciones otras dos funciones en orden y en los elementos que no son hojas se encarga de realizar el recorrido del árbol correctamente.

#### 4.5.1.8. Detalles de Lectura

Una vez solucionada la lectura de todos los elementos por medio del revisor de pantalla arreglamos un par de detalles. El primero es que, al cambiar el foco de la aplicación al *TextBox* donde mostramos la partitura en braille, no se leía el elemento donde se hubiera seleccionado (habitualmente el primero). Esto es porque en esa acción no se ejecuta la función *SayCharacter* de JAWS y, por tanto, no pasa por nuestra función sobrescrita. Para arreglar esto, buscamos la función que se llamaba en este caso para sobrescribirla y encontramos que al menos una de las funciones que se llamaban era *FocusChangedEventEx*. De nuevo tuvimos que tener cuidado para no sobrescribir demasiado la funcionalidad habitual de JAWS pero, sin muchas más complicaciones, arreglamos este detalle.

El segundo consistía en un detalle más específico y, es que, cuando se cambiaba el foco dentro de la aplicación al *TextBox*, JAWS decía, por lo que nos pareció entender, "type index" tras leer correctamente el elemento. Esto lo decía incluso habiendo sobrescrito la función que se llamaba al cambiar el foco de la aplicación. Dado que no encontramos ninguna otra función que fuese la causante de este mensaje encontramos otra manera de solucionarlo. Utilizamos la instrucción *SpeechOff* de JAWS para bloquear la lectura del revisor de pantalla. Esto puede ser peligroso y no recomendable ya que puedes evitar la lectura de otros mensajes importantes, sin embargo, la lectura se vuelve a activar (*SpeechOn*) en el momento en el que se realiza cualquier otra acción. Dado que, en nuestro caso, después de cambiar el foco al *TextBox* y de leer el elemento ahí presente, no hay que comunicar ningún otro mensaje antes de realizar otra acción, podemos hacer uso de esta solución sin estropear

el resto de funcionalidades.

Con estos dos detalles y tras varias pruebas de uso, dimos por acabada la parte relacionada con el revisor de pantalla y continuamos con otros aspectos importantes de la accesibilidad.

### 4.5.2. Recorrido por Teclado y Atajos

Recordemos que otro de los aspectos más importantes de la accesibilidad es el de permitir hacer uso completo de la aplicación a través del teclado. La forma en la que están implementados los controles de Wpf ya permite un recorrido a través de ellos por medio del tabulador. No solo esto, si no que tiene funcionalidades para facilitar el movimiento a través de los menús por medio del teclado. Gracias a esto, nosotros únicamente tuvimos que asegurarnos de que los elementos se recorrían de una forma cómoda. Para asegurarnos de esto, tuvimos que tener cuidado con qué elementos eran seleccionables y cuáles no. Por ejemplo, no nos interesaba permitir la selección del panel de visualización de partitura en tinta puesto que no es modificable por teclado y únicamente entorpecería el recorrido de la aplicación por medio del teclado.

Además del recorrido de la aplicación por medio del teclado pensamos en diseñar algunos atajos de teclado para permitir un acceso fácil a las funcionalidades más importantes de la aplicación. Para hacer esto, fue necesario crear objetos en nuestro proyecto que implementaran la interfaz *ICommand*. A estos elementos les indicamos la combinación de teclas necesaria para activarlos y las acciones que deben ejecutar al activarse. También fue necesario añadir estos elementos a la lista de comandos que se pueden introducir en nuestra ventana principal. De esta forma, mientras la ventana principal de la aplicación tenga el foco, al realizar la combinación de teclas adecuada se ejecutará la acción deseada.

### 4.5.3. Tamaño de los Elementos

Otro aspecto importante que queríamos que permitiera nuestra aplicación era modificar el tamaño de los diferentes elementos. Con esta funcionalidad, permitiríamos que las personas con visibilidad reducida pudieran adaptar el tamaño de los diferentes elementos de la aplicación a su gusto. La forma en la que implementamos esto fue a través de *Binding*. El *Binding*, como se ha explicado en el apartado de tecnologías usadas (sección 3.2.4), es una herramienta muy potente. En este caso, lo hemos utilizado para que el tamaño asociado a los elementos, en vez de ser un número constante, sea una variable que varía dinámicamente.

En la aplicación hay tres partes principales, el menú, la partitura en tinta y la partitura en braille. Para modificar el tamaño de los menús, la propiedad que cambiamos es el tamaño de fuente de la letra. En la partitura en braille lo mismo, dado que está en formato de texto modificamos el tamaño de la fuente. En la partitura en tinta modificamos la propidedad de *ZoomFactor* que viene proporcionada por el control de Manufaktura para cambiar el tamaño de la partitura. Para cada una de estas partes guardamos una variable (*FontSize*, *BrailleSize* y *ScoreZoomFactor*, respectivamente) a la que se realiza un *Binding* desde la

ventana. Además implementamos funciones que varían el valor de estas variables y creamos opciones dentro del menú para variar cada uno de estos tamaños. Por último, añadimos un atajo de teclado que incrementa todos los elementos a la vez y otro que decrementa de la misma forma.

### 4.5.4. Resto de Elementos de Accesibilidad

Para completar la accesibilidad de nuestra aplicación nos faltaban algunos detalles. Uno de ellos era el contraste de colores de nuestra aplicación. Aunque nuestra aplicación no utiliza muchos colores y los elementos se representan en un básico negro sobre blanco, tenemos que tener en cuenta que personas con diferentes discapacidades visuales pueden necesitar otras gamas de colores para distinguir los elementos correctamente. Sin embargo, al ser una aplicación Wpf y teniendo cuidado de no modificar la configuración por defecto, nuestra aplicación hereda correctamente el contraste de colores establecido por el sistema Windows. Esto quiere decir que si en el panel de control, el usuario decide invertir los colores, nuestra aplicación también se verá con los colores invertidos.

Un último detalle, pero muy importante, es la elaboración de una guía de uso. Esto es útil para cualquier usuario pero, especialmente, para los usuarios invidentes. En esta guía indicamos los diferentes elementos que hay en la aplicación y las acciones que se pueden realizar dentro de esta. De esta forma, un usuario nuevo, puede tener una referencia para no tener que explorar la aplicación antes de poder hacer uso de ella. La guía se adjunta en un archivo de texto (*.txt*) con la aplicación pero, además, la hemos metido dentro de la aplicación. En el menú se puede seleccionar la opción de abrir la guía de uso que abrirá una nueva ventana con la guía de uso. Es importante mencionar que JAWS leerá correctamente esta guía, ya que, el uso habitual de JAWS soporta la lectura de texto y en nuestro Script hemos tenido cuidado de no sobrescribir esta funcionalidad. Otra cosa relevante de la ventana de la guía es el hecho de que los atajos de teclado que hemos implementado no funcionarán ya que, estos están asociados a la ventana principal de la aplicación y no a la de la guía. Para arreglar esto, hicimos algunas modificaciones sobre los comandos para que se incluyera el atajo que cambia de tamaño el texto en la ventana de la guía de uso.

## 4.6. Modificación de los objetivos iniciales

En esta etapa del desarrollo tenemos una aplicación que lee una partitura en formato MusicXML y la muestra por pantalla tanto en tinta como en braille siendo compatible con salida por línea braille. Todo esto con código libre y modular para facilitar la ampliación y reutilización del trabajo hecho en este proyecto.

Llegados a este punto del proyecto, nuestra idea inicial era desarrollar la traducción inversa de braille a tinta. Esto nos permitiría, con el código que ya teníamos desarrollado, cargar partituras en formato braille y mostrarlas tanto por pantalla (en tinta y en braille) como por línea braille. Además de sentar las bases para poder ampliar el programa en el futuro añadiendo la opción de modificar el braille y que los cambios se mostrasen en tiempo real. Sin embargo, tras un estudio teórico sobre la traducción de braille a tinta, nos encontramos

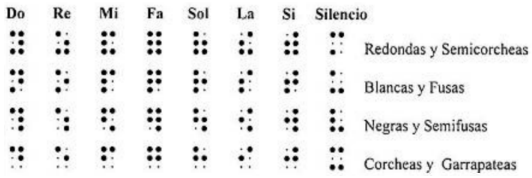



Figura 4.11: Correspondencia de notas musicales en braille

con el problema de que esta traducción no es unívoca: hay distintas combinaciones de cajetines braille que pueden tener distintos significados según el contexto en el que estén escritos. Por ejemplo, la figura musical (signo que representa gráficamente la duración musical de un sonido) no queda identificada unívocamente por un cajetín braille (figura 4.11) sino que necesita un análisis del contexto. Por tanto para poder hacer esta traducción necesitábamos hacer un estudio pormenorizado del código braille musical y su traducción dependiente del contexto.

Para desempeñar esta tarea contábamos inicialmente con la colaboración de la ONCE, sin embargo, debido a la situación de confinamiento en que nos hallábamos en este punto del proyecto (y con la previsión de que esta situación no cambiaría en un futuro cercano) parecía poco probable que pudiésemos colaborar con ellos mano a mano, algo que resultaba fundamental para poder llevar a cabo este objetivo.

Una vez nos dimos cuenta de que a causa del confinamiento no íbamos a tener el asesoramiento necesario para poder realizar esta traducción correctamente decidimos cambiar los objetivos iniciales del proyecto. En lugar de hacer la traducción inversa, de braille a MusicXML, decidimos incluir otras funcionalidades a nuestra aplicación: reproducción de la melodía y modificación en tiempo real de la partitura en tinta.

## 4.7. Reproducción de la melodía

Dada la reelaboración de los objetivos iniciales, una de las funcionalidades que decidimos incluir en el proyecto como nuevo objetivo fue la reproducción de la partitura.

En primer lugar investigamos sobre cómo pasar la partitura de MusicXML a MIDI ya que, al ser ambos formatos muy populares, parecía factible encontrar una librería que hiciese esta traducción. Sin embargo no fue así. Tuvimos muchas dificultades para encontrar una librería de código abierto con este fin y no podíamos invertir personas en desarrollar este módulo desde cero ya que hubiese supuesto eliminar otras tareas (ver figura 3.1).

Por tanto, pasamos a buscar una solución a partir del código desarrollado hasta el momento para la aplicación, buscando algún punto a lo largo de la traducción de MusicXML a braille a partir del cuál fuese más sencillo pasar la partitura a MIDI (u otro formato musical). Así, encontramos la solución de usar la librería *Manufaktura.Controls* (Salamon, s.f.) que habíamos empleado ya para representar una partitura en tinta a partir de un archivo MusicXML. Esta librería tiene métodos para generar un archivo MIDI a partir de clases

internas que representan la partitura y que se crean a partir de un archivo MusicXML.

En primer lugar, pasamos de un string (*SourceXml*) que contiene el archivo MusicXML en su totalidad a un objeto *Manufaktura.Controls.Model.Score* que representa una partitura mediante el método *Manufaktura.Controls.Linq.ToScore*. A partir de dicha representación de la partitura usamos el constructor *Manufaktura.Controls.Desktop.Audio.MidiTaskScorePlayer* para crear un objeto de tipo *MidiTaskScorePlayer*, que es un reproductor para una partitura en particular. La clase *MidiTaskScorePlayer* hereda de la clase abstracta *ChannelSelectingTaskScorePlayer* que a su vez hereda de la clase abstracta *TaskScorePlayer* que a su vez hereda de la clase abstracta *ScorePlayer*, como vemos en la figura 4.12. Un *ScorePlayer* proporciona los métodos *Play, Stop* y *Pause* para controlar el reproductor y tiene un atributo *State* que nos indica qué está haciendo el reproductor en cada momento, sus posibles valores son *Idle* (inactivo), *Paused* (pausado) y *Playing* (reproduciendo).

Con estos controles tendríamos las herramientas suficientes para hacer un reproductor con opción de parar y pausar una melodía. Sin embargo, cuando probamos un reproductor creado como acabamos de describir, descubrimos ciertas deficiencias en la librería de *Manufaktura.Controls* ya que la reproducción no funcionaba correctamente: se solapaban melodías, había notas que no sonaban, había que detener una partitura manualmente para poder volver a reproducirla sin tener que reiniciar el programa, etc. Tuvimos entonces que meternos en el código de esta librería y estudiar clase por clase qué estaba fallando, ya que la librería viene sin documentación. Finalmente descubrimos que todos los problemas del reproductor se debían a dos factores:

1. Cuando se acaba de reproducir una partitura, el estado permanecía en *Playing*, cuando debería pasar a ser *Idle*. Esto impedía distinguir cuando podíamos reproducir de nuevo la partitura y cuando no; no podíamos gestionar cuando permitíamos pulsar el botón de reproducir para evitar solapamientos en el audio.
2. Al reproducir una partitura se usa un iterador para recorrer las estructuras internas que forman el archivo de audio. Al pausar y volver a reproducir una partitura, se mueve el iterador antes de volver a reproducir de forma que en ocasiones se pierden notas en el audio.

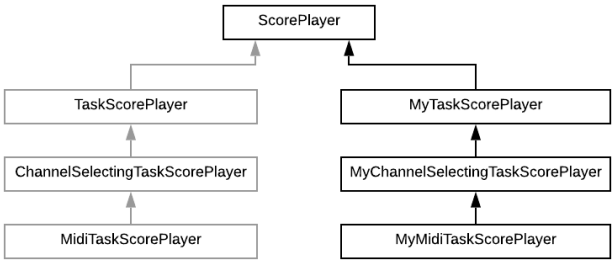



Figura 4.12: Diagrama de algunas clases de *Manufaktura.Controls*

Gracias a que el código de *Manufaktura.Controls* tiene licencia MIT, hemos podido modificar sus clases para corregir estos bugs. Las funciones *PlayInternal, Play, Pause* y

*Stop* que había que modificar estaban en la clase abstacta *ScorePlayer*. Recordemos que habíamos usado una instancia de *MidiTaskScorePlayer* como reproductor. Creamos unas clases análogas que modificamos para solucionar los problemas de código, para diferenciarlas les añadimos el prefijo *My* (figura 4.12).

Finalmente, para reproducir una partitura, hacemos el mismo proceso que hemos explicado antes con las clases de la librería *Manufaktura.Controls* con el cambio de que usamos como reproductor una instancia de *MyMidiTaskScorePlayer* en lugar de una de *Manufaktura.Controls.Desktop.Audio.MidiTaskScorePlayer*.

## 4.8. Modificación de la partitura en tinta en tiempo real

Otra de las nuevas funcionalidades que añadimos como objetivo fue poder modificar la partitura en tiempo real. Ya existen editores de partituras que te permiten crear y modificar una partitura y después exportarla y traducirla con otro programa a braille. El problema de este sistema es que es lento y no es interactivo. Por tanto la parte más importante era en nuestra aplicación esta funcionalidad funcionase en tiempo real.

Recapitulando, necesitamos primero permitir de algún modo que el usuario interactúe con la partitura en tinta y segundo comunicar el componente de la partitura en tinta con el componente braille en tiempo real para que las modificaciones hechas se reflejen instantáneamente.

Para que el usuario modifique la partitura vamos a usar únicamente la componente gráfica. La acción no se podrá realizar sin usar el ratón porque no está dirigida a usuarios invidentes. Un usuario invidente modificaría la partitura en braille (acción que se realizaría únicamente interactuando con el teclado o línea braille) y esta acción necesita la implementación de la traducción inversa (de braille a tinta) para funcionar.

Repasemos brevemente el funcionamiento de la representación en tinta. En la clase *MainWindow* tenemos un atributo *string SourceXml* que tiene propiedad de dependencia y cuyo valor es el contenido del archivo MusicXML cargado en cada momento y se modifica cuando cargamos una nueva partitura. En la parte gráfica tenemos un control de tipo *ManufakturaControls:NoteViewer* cuyo atributo *XmlSource* (su valor es el contenido del archivo MusicXML que se representa en el componente) está *binded* a nuestra variable *SourceXml*. De esta forma cuando cambiamos el contenido de la variable *SourceXml* cambia la representación gráfica reflejando los cambios.

El control *ManufakturaControls:NoteViewer* se puede configurar para que permita al usuario arrastrar las notas con el ratón cambiándolas de altura (modificación del tono) de forma que esos cambios se reflejan en su atributo *XmlSource*. Por tanto, ahora queremos que los cambios en *XmlSource* se reflejen en tiempo real en *SourceXml*, para ello hacemos el *binding* entre estas variables bidireccional.

Por otra parte, el texto braille (tanto el que se muestra por pantalla como el que sale por línea braille, es el mismo) está *binded* a nuestra variable *Braille* que es un string con propiedad de dependencia que almacena la partitura en braille. Recordamos que para obtener el

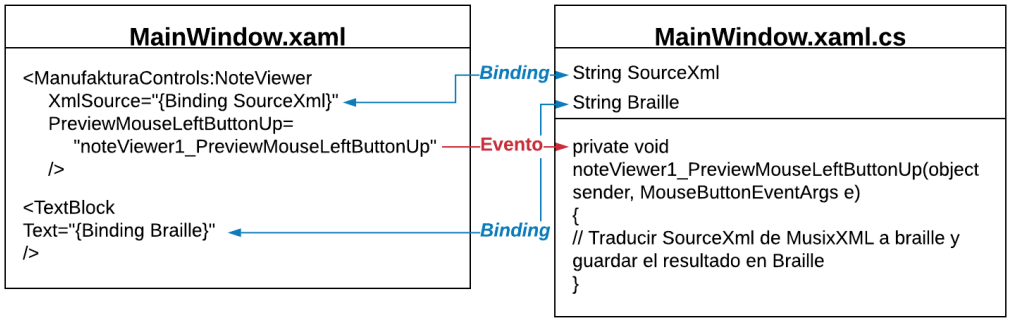



Figura 4.13: Diagrama de clases con los principales mecanismos de comunicación entre la parte gráfica y el modelo

texto de la partitura en braille realizamos la traducción desde MusicXML pasando por las estructuras de árbol de MusicXML y árbol de braille. Para realizar esta traducción cada vez que se modifique *SourceXml* usamos paso de eventos. Desde el control *ManufakturaControls:NoteViewer* lanzamos un evento *noteViewer1_PreviewMouseLeftButtonUp* mediante el atributo *PreviewMouseLeftButtonUp* cada vez que se levante el botón izquierdo del ratón, ya que es la última acción que sucede al arrastrar una nota. Resumimos las interacciones entre los elementos en la figura 4.13.

## 4.9. Ampliación de las características musicales

En una primera iteración del desarrollo implementamos un modelo musical básico: un pentagrama con clave, armadura, tempo, notas y silencios. La cantidad de características musicales representables en un pentagrama en tinta compone una lista mucho más extensa. Para concluir nuestra aplicación decidimos hacer una segunda iteración y añadir características a la representación.

Hemos incluido las alteraciones (sostenidos y bemoles) entre las características representables. Para ello hemos añadido los atributos correspondientes a lo largo de todo el modelo (figura 4.4).

Primero extraemos la información necesaria del árbol de MusicXML y se la añadimos a nuestro elemento *BrailleNote* del árbol de braille. Después modificamos las funciones *ParseBraille* y *ParseText* de la clase *BrailleNote* para que la información sobre la alteración que acompaña o no a la nota se represente correctamente en braille y sea leída correctamente por el revisor de pantalla.

## 4.10. Creación del ejecutable

Al finalizar con la programación del proyecto fue necesario obtener un fichero con el cuál se pudiera compartir el programa para poder realizar la experimentación. Esto en muchas ocasiones puede resultar muy sencillo porque el propio entorno de Visual Studio en el que este programa ha sido desarrollado, ofrece la opción de publicar la aplicación en formato *.exe*. Sin embargo, en seguida nos dimos cuenta que esta funcionalidad no nos funcionaba porque nuestro programa requiere permisos de administrador para realizar la copia de algunos archivos y la herramienta no permite publicar aplicaciones que exijan permisos de administrador. Esto fue un problema con el que no contábamos y para el cual realizamos la búsqueda de diferentes opciones para lograr resolverlo.

El problema finalmente lo resolvimos creando un proyecto de instalador para nuestra aplicación que resulta ser otra de las formas habituales de publicar aplicaciones. En este caso, utilizamos la extensión de Visual Studio para la creación de SetUp Projects y creamos nuestro proyecto de instalador que llamamos *LiveDotsInstaller*. La forma que tienen estos proyectos de funcionar es que instalan los archivos que se deseen en el ordenador cliente que ejecute el instalador. Dentro de estos archivos que instalan puedes elegir incluir la salida de otro proyecto que hayas creado junto con todas las librerías de las que depende. En nuestro caso, al incluir la salida del proyecto LiveDots en el instalador LiveDotsInstaller logramos que un usuario cliente pueda instalar el archivo que ejecuta la aplicación LiveDots. Además, se añadirán automáticamente dentro de la misma carpeta todas las librerías que utiliza nuestro proyecto. De esta forma, el instalador solicitará permisos de administrador y copiará los archivos necesarios para la ejecución correcta de la aplicación sin necesidad de que la propia aplicación lo haga.

Además de la aplicación y las librerías que esta necesita, se pueden añadir otros archivos al instalador. Esto fue muy conveniente para instalar la fuente de braille que utiliza el programa y el objeto COM que se utiliza para la comunicación entre el programa y el revisor de pantalla. Para instalar la fuente de braille con el proyecto de instalador, únicamente fue necesario añadir la fuente al proyecto como fichero *.ttf* e indicar al instalador que instalara este fichero como fuente en el ordenador cliente. Un proceso análogo se hizo para la instalación del objeto COM. Con esto, logramos tener un archivo que podríamos distribuir con facilidad para realizar la experimentación.

# Capítulo 5

# Experimentación

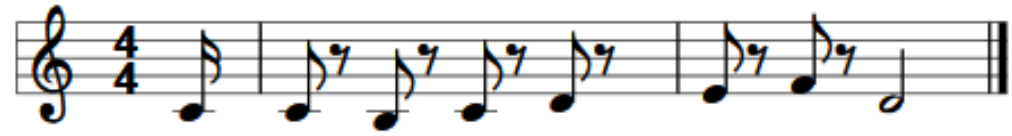
## 5.1. Pruebas internas del programa

Una vez acabado el programa y durante el desarrollo del mismo fuimos realizando diferentes pruebas para comprobar el correcto funcionamiento. Las partes más importantes que tuvimos que probar fueron la traducción a braille, la lectura de los elementos con el revisor de pantalla y la accesibilidad del programa. Para comprobar que la traducción a braille era correcta fuimos contrastando diferentes partituras de ejemplo que vienen en el manual internacional de musicografía braille (figura 5.1). Para la lectura de los elementos con el revisor de pantalla simplemente utilizamos la partitura en tinta como referencia para comparar lo que ponía con lo que decía. Por último, para comprobar que la aplicación era accesible realizamos diferentes recorridos por todos los elementos de la aplicación comprobando que se pudieran acceder con el teclado y que el revisor de pantalla informaba adecuadamente. Además, comprobamos que no se produjeran errores al ejecutar las diferentes funcionalidades de la aplicación.

## 5.2. Experimento

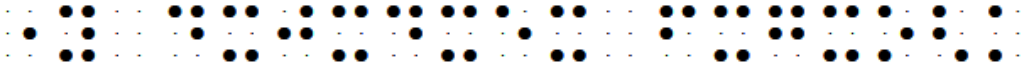
Para comprobar si la aplicación *LiveDots* puede ayudar a incluir estudiantes ciegos en las clases de música realizamos un experimento en el que presentamos *LiveDots* a distintos usuarios para que la probasen. Nuestra idea inicial era llevar la aplicación a la ONCE para que usuarios ciegos la probasen y nos diesen su opinión sobre usabilidad y experiencia de uso. Después de esto, también íbamos a llevar la aplicación a profesores de musica videntes y alumnos de música ciegos para obtener *feedback* de los usuarios finales. Sin embargo debido a la situación de confinamiento causada por el COVID-19, no pudimos llevar estas pruebas a término.

En su lugar realizamos un experimento *blindfolded* (a ciegas) con personas videntes. Este



Partitura de prueba en tinta.

Partitura de prueba en braille del Manual Internacional.



Partitura de prueba en braille de la aplicación LiveDots.

Figura 5.1: Ejemplo de prueba realizada para la traducción de braille.

tipo de experimento consiste en no poder usar el sentido de la vista, para ello el participante puede o bien taparse los ojos o bien apagar la pantalla del ordenador. De este forma se simula la interacción de un usuario ciego.

De esta forma podemos comprobar que la aplicación es usable para un usuario invidente, es decir, comprobamos que la aplicación es accesible en el sentido de que puedes realizar todas las acciones posibles usando únicamente el teclado (y atajos de teclado) y el audio de un revisor de pantalla. Sin embargo es importante remarcar que este tipo de experimento no refleja la experiencia de uso de un usuario invidente debido a la distinta filosofía que tienen a la hora de interactuar con el ordenador.

## 5.3. Objetivo del experimento

El objetivo principal de este experimento era comprobar si la aplicación *LiveDots* es útil para incluir a un alumno ciego en un aula de música. Pensamos que ser capaz de traducir partituras a braille en tiempo real aumentaría la interacción entre profesor y alumno, mejorando así la participación del alumno. Para verificar esta hipótesis, formulamos las tres siguientes **preguntas de investigación** que queremos responder con los resultados del experimento:

*RQ1. ¿Puede la aplicación LiveDots ayudar a incluir a un alumno ciego en una clase de música?*

Además, para comprobar si la aplicación es útil, es importante verificar también que es utilizable. Por tanto, otro objetivo de este test es verificar la usabilidad para ambos usuarios objetivos: profesores de música (videntes) y estudiantes de música (invidentes).

Por una parte, es un requisito que una persona ciega pueda usar la aplicación *LiveDots* sin intervención o ayuda de otra persona. Los atajos de teclado y el revisor de pantalla deben ser prácticos y fáciles de usar. Para verificarlo, planteamos la siguiente pregunta de investigación:

*RQ2. ¿Puede una persona ciega usar la aplicación LiveDots sin ninguna ayuda externa?*

Por otra parte, un profesor debe poder usar *LiveDots* incluso sin tener ningún conocimiento sobre braille o braille musical. Las acciones de cargar una partitura y modificarla deben ser sencillas. Por tanto la última pregunta de investigación es:

*RQ3. ¿Puede una persona vidente con conocimientos musicales que no sabe braille musical usar la aplicación LiveDots?*

## 5.4. Participantes

El estudio contó con 7 participantes. Todos ellos eran personas videntes sin conocimientos sobre braille o braille musical y con distintos conocimientos básicos sobre notación musical. Cada uno de ellos probó la aplicación en dos escenarios distintos: primero *blindfolded* (simulando el punto de vista del alumno ciego) y después pudiendo ver la pantalla (simulando el punto de vista del profesor vidente). Por último, los participantes rellenaron una encuesta sobre *LiveDots*.

## 5.5. Diseño experimental

Diseñamos el experimento para tratar de responder a las tres preguntas de investigación. Primero usaron la aplicación y después nos dieron *feedback* a través de un cuestionario (figura 5.2).

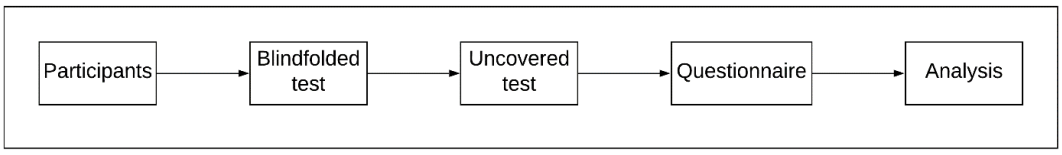



Figura 5.2: Diseño experimental

La parte blindfolded del experimento verificaba si la aplicación era utilizable por personas ciegas. Era posible que algunos de los participantes no pudieran utilizar el teclado sin verlo, por lo que había un supervisor que les ayudaba a pulsar las teclas que querían y se aseguraba de que los participantes tuvieran los ojos vendados mientras utilizaban la aplicación. En esta primera parte los participantes debían abrir una partitura, reproducirla, detenerla, mover el cursor por la partitura braille para escuchar la lectura de los elementos musicales a través del revisor de pantalla y finalmente cerrar la aplicación. Utilizaron el lector de pantalla JAWS mientras tenían los ojos vendados (figura 5.2).

Al terminar la primera parte *blindfolded*, los participantes descubrieron sus ojos y probaron *LiveDots* desde el punto de vista de un usuario vidente. Las tareas visuales eran: abrir una partitura, modificarla, guardar los cambios e identificar esos cambios en la partitura en braille usando el lector de pantalla.

La primera parte del experimento comprobó si los usuarios ciegos pueden usar la aplicación LiveDots sin ayuda externa (*RQ2*) y la segunda parte si las personas videntes sin conocimiento sobre braille musical pueden utilizar la aplicación (*RQ3*). Además, probar la aplicación desde el punto de vista de ambos usuarios finales permite a los participantes evaluar si LiveDots es útil para incluir a un estudiante ciego en una clase de música (*RQ1*).

Por último, hicieron un cuestionario con cuatro partes, cada una relacionada con una de las preguntas de la investigación. La duración de la prueba fue de unos 30 minutos, incluyendo el uso de la aplicación y cuestionario.

## 5.6. Material e instrumentos

Para realizar el experimento se facilitó a los participantes la aplicación *LiveDots* que usaron junto al revisor de pantalla JAWS.

### 5.6.1. Cuestionario

El cuestionario tenía dos preguntas iniciales de sí/no. La primera era si consideraban que usar aplicaciones informáticas para conseguir partituras en braille facilita la tarea en comparación a la forma tradicional de imprimirlas en papel perforado, y la segunda era si el participante conocía previamente alguna aplicación de traducción o lectura de partituras en braille. Nuestra hipótesis era que las respuestas serían sí y no, respectivamente. Esto resaltaría la necesidad de una aplicación como LiveDots para hacer esta traducción y lectura, justificando así de forma afirmativa la primera pregunta de investigación (*RQ1*).

Después de estas dos preguntas, el cuestionario constaba de cuatro partes diferentes. Las tres primeras valoradas en escala Likert-5, siendo 1 la peor puntuación para la aplicación y 5 la mejor. La tabla 5.1 describe estas tres partes.

Por último, había una pregunta abierta para sugerir mejoras de la aplicación *LiveDots*.

Cuadro 5.1: Resumen del cuestionario

| **Variable medida** | **Instrumento** | **Tipo** | **Cómo se calcula** | **Número de preguntas** | **Rango** |
|---|---|---|---|---|---|
| Utilidad de *LiveDots* | Test blindfolded y visual | Escala Likert-5 | Media de los valores de las preguntas | 5 | 1-5 |
| Usabilidad de *LiveDots* para usuario invidente | Test blindfolded | Escala Likert-5 | Media de los valores de las preguntas | 5 | 1-5 |
| Usabilidad de *LiveDots* para usuario vidente | Test visual | Escala Likert-5 | Media de los valores de las preguntas | 5 | 1-5 |

## 5.7. Resultados

Tras realizar el experimento, obtuvimos 7 respuestas al cuestionario. La baja tasa de respuesta se debe, una vez más, a la inesperada situación de pandemia mundial que no nos permitió realizar el experimento en las instalaciones de la ONCE ni reunir a más personas para que probasen la aplicación. Dicho esto, logramos obtener y analizar los resultados de la mejor manera posible. Para las 5 preguntas de escala Likert-5 calculamos la media de las respuestas obtenidas en cada pregunta y los resultados son los que se muestran en la figura 5.3.

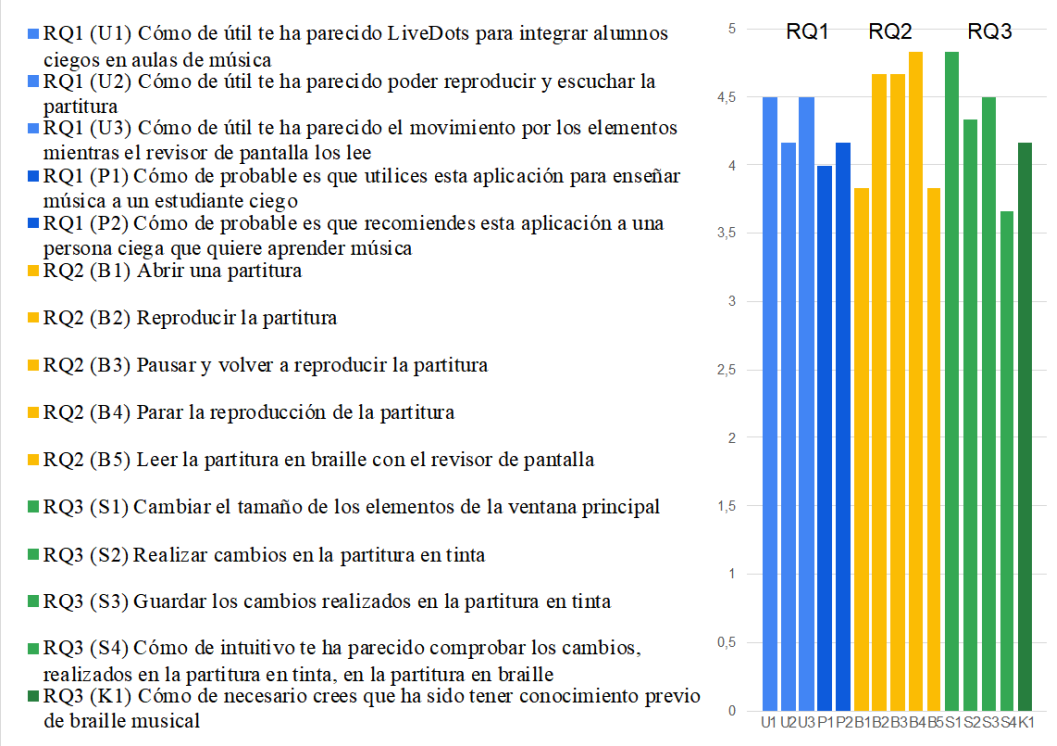



Figura 5.3: Resultados de las preguntas Likert-5

En la tabla se distinguen en por colores tres secciones diferentes. Cada sección está pensada para ayudar a responder una de las preguntas de investigación. La primera sección (correspondiente a *RQ1*) contiene las preguntas relacionadas con la utilidad de la aplicación (U1-U3) y con la probabilidad de que los participantes usen o recomienden la aplicación (P1-P2). La segunda sección (correspondiente a *RQ2*) pregunta con qué facilidad los participantes hicieron las distintas tareas *blindfolded* (con los ojos vendados) (B1-B5). La tercera sección (correspondientes a *RQ3*) se corresponde con la parte visual de la prueba (sin los ojos vendados) y pregunta cómo de fácil encontraron los participantes las diferentes tareas (S1-S4) y si se requería conocimiento previo sobre braille musical (K1). Para cada grupo de preguntas relacionadas, calculamos la media aritmética de las respuestas medias y la representamos en un gráfico de caja (figura 5.4). Por último, para las preguntas de sí/no

obtuvimos que el 100 % de los participantes piensa que el uso de aplicaciones informáticas hace más sencilla la obtención de una partitura braille en comparación con los métodos tradicionales. Y el 85,7 % de los participantes no conocen ninguna aplicación informática para la traducción y/o lectura de partituras en Braille.

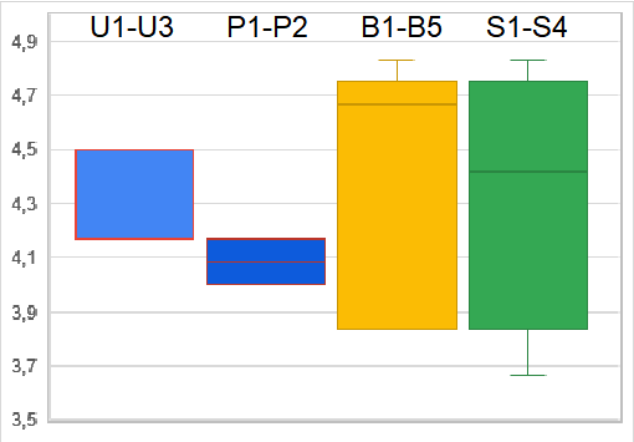



Figura 5.4: Diagrama de cajas de las distintas secciones estudiadas

## 5.8. Discusión de los resultados

A lo largo de esta sección, trataremos de responder a las preguntas de investigación expuestas en la sección 5.3 usando los resultados expuestos en el apartado anterior. En primer lugar, observamos la necesidad de una aplicación como *LiveDots* en el hecho de que aunque todos los participantes (100 %) piensan que el uso de aplicaciones informáticas facilita la tarea de obtener partituras en braille, casi ninguno (14 %) conoce una aplicación para ello. En cuanto a probar la utilidad de la aplicación *LiveDots* (*RQ1*) es necesario probar primero su usabilidad (*RQ2, RQ3*), por tanto, responderemos a las preguntas de investigación en ese orden.

**RQ2. ¿Puede una persona ciega usar la aplicación LiveDots sin ninguna ayuda externa?**

Sí. Los resultados de la prueba *blindfolded* muestran que la aplicación puede ser utilizada fácilmente usando únicamente el teclado con JAWS como lector de pantalla (4.36/5 en la usabilidad para invidentes B1-B5).

**RQ3. ¿Puede una persona vidente con conocimientos musicales que no sabe braille musical usar la aplicación LiveDots?**

Sí. Los resultados de la prueba visual muestran que los participantes sin ningún conocimiento de braille musical podrían utilizar la aplicación sin ningún problema (4.33/5 en usabilidad para videntes S1-S4). De hecho, la mayoría de los participantes (85 %) piensa que se necesitan poco o ningún conocimiento sobre braille musical para utilizar la aplicación.

**RQ1. ¿Puede la aplicación LiveDots ayudar a incluir a un alumno ciego en una clase de música?**

Sí. Después de realizar la prueba *blindfolded* y la visual, los participantes piensan que la aplicación y sus funcionalidades pueden ser útiles para integrar a un estudiante ciego en una clase de música (4,38/5 en utilidad U1-U3) y es probable que usen o recomienden la aplicación (4,08/5 en P1-P2). Además, *RQ2* y *RQ3* muestran que la aplicación también es utilizable, lo cual es un requisito necesario para que sea útil.

# Capítulo 6

# Aportación individual

## 6.1. Óscar Díaz Ribagorda

El proyecto comenzó con la propuesta de la ONCE para realizar un programa que ayudara con la inclusión de alumnos invidentes en aulas de música. Empezamos con una reunión con algunos encargados de la ONCE que nos indicaron el objetivo e interés de una aplicación de estas características. Tras esto nos pusimos a investigar todos los diferentes recursos que nos pudieran ayudar a encontrar una solución. Yo me encargué de estudiar cómo se adaptan los ordenadores a través de la línea braille y el revisor de pantalla para servir como herramienta a las personas invidentes. Esta investigación, junto con la ayuda de nuestros tutores y las sugerencias de la ONCE nos ayudó a escoger a JAWS como revisor de pantalla y el lenguaje de programación C# como lenguaje de desarrollo. Además de esto, investigué que soluciones se han dado a problemas parecidos (sin estar necesariamente relacionados con la música) centrándome en qué programas informáticos se utilizan en la educación de alumnos ciegos. Varios de los diseños e ideas de concepto nos sirvieron para inspirar nuestra solución. Una de las aplicaciones que más nos ha inspirado en el diseño es el editor de braille científico EDICO. Por último, me informé de cómo se representa la música en braille y busqué diferentes recursos, como el manual internacional de musicografía braille (Krolick, 1996), para que mis compañeros y yo pudiéramos utilizar durante el desarrollo de la aplicación. Durante toda esta búsqueda de recursos fuimos recolectando cualquier fuente que nos pudiera ser de ayuda más adelante y hubo una publicación que encontré (*A Transcription System from MusicXML Format to Braille Music Notation*, Gotoh y col., 2008) que resultó ser una gran contribución al desarrollo del programa.

Tras el estudio del estado del arte, nos propusimos empezar a programar cuanto antes e ir logrando versiones cada vez más funcionales de la aplicación. De esta forma, evitábamos el riesgo de organizar un proyecto que tuviera un tamaño fuera de nuestras posibilidades. Las primeras funcionalidades del programa las fuimos añadiendo de forma colectiva pues, en esta etapa, estábamos entrando en contacto con el lenguaje de programación C# (en particular con los proyectos Wpf) que ninguno habíamos utilizado con anterioridad. En cuanto nos

empezamos a familiarizar con el entorno y pusimos toda la información y propuestas sobre la mesa, logramos entre los tres tener un primer diseño conceptual de la aplicación sobre el cual poder distribuir el trabajo.

La primera gran funcionalidad que logramos implementar fue la representación de partituras en tinta a partir de ficheros en formato MusicXml. Para esto hicimos uso de una librería de código libre (Manufaktura) que encontramos durante nuestro estudio del estado del arte (Salamon, s.f.). Durante esta etapa ya teníamos pensado que el braille se iba a mostrar a la vez en la pantalla pero dado que no habíamos implementado aún la traducción, decidimos mostrar el documento MusicXml donde debería ir el braille.

En la segunda etapa nos centramos en la traducción de MusicXml a braille. En este momento ya teníamos suficientemente organizado el proyecto como para poder distribuir las diferentes tareas aunque continuáramos ayudándonos y poniendo en común todos los avances. En mi caso, fue en este momento cuando se me ocurrió que podríamos utilizar las ideas que se proponían en la publicación de Gotoh y col., 2008 para implementar la traducción de MusicXml a braille en C#. Para ello, propuse implementar un árbol diferente para cada una de las partes y programar una función que convirtiera de un árbol al otro. Mi compañero Lucas de Torre y yo nos encargamos de implementar esta función de traducción entre los árboles. Esto fue importante hacerlo entre dos personas porque hacía falta que hubiera coherencia entre los atributos de los diferentes objetos de los árboles y la forma en la que convertíamos los objetos de un árbol a otro. Con esta dinámica logramos que el programa convirtiera correctamente de un archivo MusicXml a un árbol de elementos braille. A partir de aquí, era necesario que cada elemento braille supiera cómo se escribía para poder mostrarlo por pantalla y se daría por finalizada la traducción de partitura en tinta a braille.

Con la traducción a braille lograda solo nos quedaba una gran funcionalidad por cumplir, esto consistía en la accesibilidad de la aplicación. Dentro de la accesibilidad de la aplicación queríamos incluir la lectura de los elementos musicales en braille por el revisor de pantalla. Esta parte resultó ser la más complicada de todas ya que tuvimos que entender bien el funcionamiento de JAWS y cómo conectar nuestra aplicación con este revisor de pantalla para que leyera los elementos musicales que representaba el braille mostrado. Yo me encargué de entender bien el funcionamiento y la documentación de JAWS. En este proceso, realicé pruebas para ver que funciones de JAWS podríamos sobrescribir para lograr el comportamiento que queríamos con el revisor de pantalla. Encontré que la función *SayCharacter* sería la más adecuada y me encargué de programar el Script para que JAWS funcionara como queríamos con nuestra aplicación. Con esto y con las partes de las que se encargaron mis compañeros logramos llegar a una solución que diferencia a nuestra aplicación de la mayoría del resto de aplicaciones accesibles. Además del revisor de pantalla, acabamos entre todos la implementación del resto de funcionalidades que hacen que nuestra aplicación se accesible.

Una vez tuvimos una aplicación con las funcionalidades que queríamos, nos centramos en añadir y comprobar diferentes partes de la aplicación que resultarían en una solución más completa. En mi caso, utilicé las diferentes herramientas que nos proporcionaba el lenguaje (Binding) y la librería de Manufaktura (edición) para permitir la modificación en tiempo real de la altura de las notas. Esta funcionalidad se puede ampliar si se amplía las posibilidades de edición de la partitura en tinta puesto que nuestra aplicación ya utiliza una traducción a braille de la partitura en tiempo real.

**Artículo *LiveDots: Realtime Interactive Braille Music Translator to Integrate Blind Students into Music Classes***

El desarrollo de este proyecto y la investigación realizada para su concepción y diseño nos pareció, a nuestros tutores y a nosotros, de gran importancia para el avance en el uso de tecnologías para la inclusión de personas ciegas. Con esta idea, decidimos realizar un experimento formal para comprobar nuestras sospechas. En este experimento, preparé junto con mis compañeros el cuestionario que íbamos a utilizar y preparé la aplicación para su fácil distribución. Además, me encargué de realizar la prueba con tres participantes.

Con la información recopilada del experimento decidimos escribir un artículo para el congreso ICCE 2020 en el campo del uso de ordenadores en la educación. A la hora de escribir este artículo, yo me encargué de explicar el desarrollo de la aplicación LiveDots (Apartado 3 del Anexo A). Para ello, realicé un esquema conceptual de la aplicación para organizar correctamente todas las partes importantes. Después expliqué de forma concisa cada parte de la aplicación LiveDots centrándome en los aspectos innovadores de cada una.

Tanto la escritura del artículo como el desarrollo del resto del proyecto han sido una gran experiencia que me ha servido no solo para aprender mucho sino para querer aprender más.

## 6.2. Claudia Guerrero García-Heras

El primer paso del proyecto fue hacer un estudio del estado del arte para familiarizarse con el campo del braille musical, de las aplicaciones accesible y de las aplicaciones musicales. Como primera aportación me encargué de la investigación acerca de las aplicaciones de edición de partituras.

Al analizar estas aplicaciones me di cuenta de que casi ninguna era completamente accesible ya que pierden funcionalidades al usar un revisor de pantalla. Este problema se debe en la mayoría de los casos a que el desarrollo de la aplicación se hace pensando únicamente en los usuarios videntes y se añade la accesibilidad como una característica a posteriori. Al hacer esto es posible que nos encontremos con situaciones que son muy complicadas de transcribir con un revisor. El origen por tanto del problema es un sesgo a la hora de desarrollar software. Por tanto, mi estudio aportó que para poder hacer una aplicación totalmente accesible necesitábamos pensar en desarrollar software tanto para videntes como para invidentes desde el comienzo del desarrollo de la aplicación.

Tras implementar un primer boceto de la aplicación de forma colaborativa, llegamos al problema de cómo representar pentagramas y braille en una aplicación WPF. Yo me encargué de resolver cómo representar cajetines braille por pantalla en un formato legible y cómo saber qué cajetin braille estamos representando sin necesidad de saber braille. Para ello realicé en primer lugar un diccionario que nos permitiese usar la fuente TrueType de cajetines braille *edico_es_br6.tt* que nos proporcionó la ONCE. Era necesario este paso intermedio porque esta fuente estaba pensada para EDICO, que es un editor científico y las correspondencias entre caracteres ASCII y símbolos de la fuente estaba orientada a

representar símbolos científicos. El diccionario desarrollado establecía una correspondencia entre cadenas de números que indican los puntos resaltados en un cajetín braille y el caracter ASCII correspondiente a dicho cajetín en la fuente. De esta manera di una solución general al problema.

El siguiente paso fue la traducción de una partitura de MusicXML a braille. En esta etapa me encargué de la traducción de árbol MusicXML a braille. Para conseguir una secuencia de caracteres braille que represente el árbol añadí atributos de representación a cada una de las clases del árbol braille que asignan a cada elemento los cajetines braille correspondientes. Posteriormente un recorrido del árbol en el orden correcto proporciona la secuencia de braille deseada. Para realizar esta tarea era necesario saber qué símbolos braille asignar a cada símbolo, ésta fue la parte que más tiempo llevó. Para realizar esta correspondencia usé el estándar del manual internacional de musicografía braille (Krolick, 1996). Fue en general un proceso de aprendizaje y familiarización con el braille muy necesario ya que no sabía nada al respecto antes de comenzar el proyecto. Para la correcta representación del braille en linea braille es importante hacer los saltos de línea en braille de manera adecuada, lo cual tiene la complicación de que en braille hay elementos que se representan de distinta forma en función de si caben en la línea actual o no. Implementé también un sistema de contadores para solucionar esta cuestión.

En la siguiente etapa trabajamos en configurar el lector de pantalla JAWS para que funcionase correctamente con la aplicación y conseguir que leyese las figuras musicales correspondientes al pasar el cursor por los cajetines braille. Mi aportación en esta etapa fue desarrollar e implementar una solución que nos permitiese que al recorrer los elementos braille con el cursor se leyesen los símbolos musicales correspondientes con el revisor de pantalla. Esta tarea tiene la complicación de que mientras que muchos de los elementos que se representan en tinta con un único símbolo se corresponden con varios cajetines braille y el número de cajetines con los que se representa no solo depende de la figura en sí, sino también del contexto en el que se encuentre. Implementé por tanto un sistema de tres listas para llevar toda la información necesaria como se explica en la sección 4.5.1.5.

Como última etapa de desarrollo de la aplicación, buscamos una forma de reproducir la partitura como funcionalidad añadida. Buscamos una librería para pasar a MIDI desde MusicXML, pero al final encontré una forma más sencilla usando directamente la librería *Manufaktura.Controls* que habíamos usado para representar la partitura en tinta. Creé una primera versión de reproductor con opción de reproducir y parar pero me di cuenta de que las clases por esta librería no funcionaban como era esperable. Una parte importante de mi aportación fue arreglar los *bugs* de las librerías. La librería venía sin documentación por tanto tuve que analizar directamente el código para ver que estaba pasando, encontré un par de *bugs* (ver sección 4.7) y los solucioné cambiando el código de algunas clases para nuestro proyecto, lo que pude hacer gracias a que la librería tiene licencia MIT.

### Artículo *LiveDots: Realtime Interactive Braille Music Translator to Integrate Blind Students into Music Classes*

Este proyecto es una prueba de concepto sobre cómo generalizar la idea de interactividad mediante modificaciones en tiempo real de EDICO a otros campos. Una parte esencial del

proyecto es probar la aplicación y difundir el proyecto para que pueda tener repercusión en más áreas del conocimiento.

Para ello diseñamos un experimento en el que distintos usuarios probasen y valorasen la aplicación. Yo me encargué de supervisar la prueba de pruebas de tres participantes. Por último, tras ver la exposición de los resultados hecha por mis compañeros me encargué de discutir los resultados obtenidos en el experimento y de ver cómo respondían a las preguntas de investigación.

Para divulgar este proyecto, escribimos un artículo para el congreso ICCE 2020 (*International Conference on Computers in Education*) que es un congreso Core B organizadopor APSCE (*Asia-Pacific Society for Computers in Education*). El artículo está centrado en el desarrollo del experimento y la discusión de los resultados obtenidos. Está enfocado a demostrar que, efectivamente, la aplicación LiveDots senta un precedente de que se puede generalizar la idea de interactividad en tiempo real propuesta por el proyecto EDICO a los campos artísticos. Mi principal aportación fue concentrar la información sobre la escena de la enseñanza musical a personas invidentes para poner en contexto el proyecto y demostrar su necesidad y utilidad.

## 6.3. Lucas de Torre Barrio

En el estudio del estado del arte, realizado con la intención de adquirir conocimientos sobre los avances realizados en el uso de la informática para facilitar la accesibilidad (tanto en el campo de la música como en otros campos, como el de las matemáticas), me enfoqué en el estudio de formatos de notación musical y en las aplicaciones musicales relacionadas con la accesibilidad. La intención era ver qué formato usan mayormente las aplicaciones musicales accesibles y qué posibilidades proporcionan estas aplicaciones.

Los formatos musicales más extendidos son MIDI y MusicXML, siendo este último el más utilizado en aplicaciones accesibles. El programa BME permite la edición de partituras braille para después exportar el archivo en formato MusicXML (algo que no hemos implementado en nuestra aplicación, porque habría sido inabarcable). FreeDots es una aplicación que traduce un archivo en formato MusicXML a uno en notación de braille musical, parecido al programa GOODFEEL, y Lime Lighter está diseñado para personas con visibilidad reducida. Así, la idea que tuvimos para nuestra aplicación era utilizar el formato MusicXMLpara importar y exportar partituras e implementar las características de estas aplicaciones (no hemos implementado la de BME por no ser objeto de este trabajo, pero la aplicación ha sido diseñada con la idea de facilitar futuras ampliaciones), con la funcionalidad añadida de la interactividad en tiempo real.

Después, programamos el primer diseño de la aplicación. Esto lo hicimos en grupo (como gran parte del proyecto) porque era nuestro primer contacto desarrollando con WPF. Una vez finalizado un primer diseño, implementamos la representación de la partitura en tinta y del texto MusicXML correspondiente a dicha partitura.

La siguiente parte fue conseguir representar la estructura de una partitura en formato MusicXML y traducirla a notación de braille musical. Mi parte aquí se centró en la imple-

mentación de la traducción de árbol MusicXML a árbol braille, funcionalidad implementada con Óscar Díaz, ya que era una parte extensa y, al principio, algo confusa. Esta traducción consistía en ir recorriendo la estructura en la que almacenábamos la partitura en MusicXML (árbol MusicXML) para ir traduciéndola a la estructura en la que íbamos a almacenar la partitura en braille (árbol braille).

Esta parte de la traducción es, durante el uso de la aplicación, la más costosa computacionalmente, ya que recorre ambos árboles completamente. Por ello, hice una aproximación del coste de esta traducción (lineal en el número de nodos del árbol) ya que, de esta manera, teníamos una idea intuitiva de que el programa no iba a ser demasiado lento, algo que mermaría en gran medida la idea de interactividad en tiempo real. Con esto, teníamos ya una primera base para, a partir de una partitura en MusicXML, conseguir almacenarla con una estructura de notación braille.

Respecto a la parte de accesibilidad, me centré en el uso del teclado en la aplicación. Para ello, tenemos en cuenta la posición actual del cursor, de manera que sabemos si ha avanzado o retrocedido y así utilizamos las listas implementadas por Claudia Guerrero en la clase BrailleMusicViewer. Por otro lado, me encargué del salto de línea, que tenía que implementarse para haber como mucho 40 caracteres en una línea y que no se dividiera un elemento musical (que puede ocupar más de un carácter) en dos líneas, ya que imposibilitaría su correcta lectura.

Por último, me centré en la implementación de los atajos de teclado. Con esto, el recorrido de la aplicación para una persona no es únicamente mediante el uso de el tabulado para avanzar y retroceder, sino que hay acciones (como abrir una partitura) que se pueden realizar con un atajo de teclado.

### Artículo *LiveDots: Realtime Interactive Braille Music Translator to Integrate Blind Students into Music Classes*

Una vez terminado el proyecto, decidimos escribir un artículo sobre el mismo, ya que pensamos que era una buena idea su divulgación. El artículo lo presentamos al $28^{th}$ *International Conference on Computers in Education* (congreso Core B). Centramos el artículo en un experimento que probaba la utilidad y usabilidad de LiveDots, en el cual supervisé la prueba de dos de los participantes.

Mi parte en el artículo se enfocó en la escritura de las preguntas de investigación (que habíamos escogido de forma consensuada) y en la explicación del motivo de haber escogido dichas preguntas.

Además, me centré también en la metodología del experimento, en la cual explicaba el diseño experimental, en el que argumentábamos el motivo de que todos los participantes fueran videntes (dada la situación de pandemia, nos resultó imposible contar con participantes invidentes), qué herramientas iban a tener para probar la aplicación (nuestra aplicación LiveDots y el revisor de pantalla JAWS), y en qué consistía el cuestionario que habíamos desarrollado (el cual había sido planteado de forma que respondiera a las distintas preguntas de investigación).

# Capítulo 7

# Trabajo futuro

Este proyecto pretende tener como resultado un impacto real en el mundo educativo. Por esta razón, una parte importante del trabajo que queda por hacer es desarrollar la prueba de concepto expuesta en este proyecto. Esta ampliación podría conseguir una aplicación con todas las funcionalidades necesarias para maximizar la interactividad entre vidente e invidente. Este trabajo incluiría:

- **Aumentar la variedad de modificaciones realizables en la partitura en tinta.** En la prueba de concepto desarrollada sólo se puede modificar la altura (tono) de una nota ya existente en la partitura. Gracias al experimento, hemos comprobado que esta modificación en tiempo real puede ser de gran utilidad para integrar a alumnos ciegos en clases de música, por tanto, creemos que aumentar la variedad de modificaciones realizables en la partitura en tinta proporcionaría a la herramienta una gran interactividad.
- **Implementar la traducción de braille a música en tinta.** Para lograr la interactividad en ambos sentidos entre vidente e invidente es necesario ser capaz de hacer la traducción inversa, de música en braille a música en tinta.
- **Modificar en tiempo real la partitura en braille.** De igual forma, para lograr la interactividad en ambos sentidos también es necesario permitir al usuario invidente hacer modificaciones en la partitura que el usuario vidente pueda ver en tiempo real en la partitura en tinta.
- **Implementar la traducción de partituras en formato jpg o pdf a formato MusicXML.** Dada la alta cantidad de partituras en formato pdf o jpg que se pueden conseguir, esta implementación facilitaría en gran medida la obtención de partituras en formato braille musical. Esto se debe a que podríamos importar una partitura en uno de estos formatos (pdf o jpg), la cual se traduciría primero a formato MusicXML para obtener a partir de este la partitura en formato de braille musical.

Somos conscientes de que algunos de estos elementos estaban entre los objetivos iniciales del proyecto pero no se pudieron llevar a cabo por la pandemia. Otra funcionalidad sugerida

por participantes del experimento que podría ayudar a mejorar la interactividad es resaltar el grupo de cajetines braille correspondientes a una figura musical cuando la figura es seleccionada en la partitura en tinta para que la persona vidente se familiarice poco a poco con el sistema de notación braille.

Por otra parte el estudio realizado ha tenido ciertas limitaciones. Los instrumentos utilizados han sido limitados. El experimento *blindfolded* con personas videntes da una idea acerca de la respuesta a las preguntas de investigación, sin embargo las limitaciones de tiempo para realizar el experimento y la situación de pandemia no nos permitieron probar la aplicación con personas ciegas. Una pregunta interesante que se ha quedado sin poder contestar en este experimento es si los usuarios finales encuentran útil y fácil de usar la aplicación. El experimento ha evaluado la usabilidad y accesibilidad de la aplicación, con resultados bastante positivos, sin embargo, no ha podido recabar datos acerca de la experiencia de uso al no poder contar con participantes que formasen parte del público objetivo. Las personas invidentes podrían probar la aplicación con más detenimiento, comprobar si se siguen los procedimientos habituales de accesibilidad y si es cómoda de usar.

Como resultado del éxito del estudio, la ONCE realizará un nuevo experimento para llevar esta tecnología a las aulas para verificar que realmente funciona y cumple el propósito para el cual se ha creado: integrar a alumnos ciegos en clases de música videntes. Este será un estudio que involucrará a personas ciegas y profesores de música para comprobar el potencial de la aplicación LiveDots.

# Capítulo 8

# Conclusiones

Hemos visto que, aunque hay herramientas que permiten modificar partituras en tinta o traducir una partitura en tinta a partitura en notación braille, su uso docente es limitado. Sin embargo, existen otros campos donde se ha logrado desarrollar aplicaciones interactivas en tiempo real accesibles, como es el caso de EDICO (editor de matemáticas, física y química desarrollado por la ONCE). De ahí surgió la idea LiveDots, una aplicación que puede facilitar la inclusión de estudiantes de música ciegos en clases de música mediante esta idea de interacción en tiempo real.

Durante el estudio del estado del arte observamos que no había muchas aplicaciones accesibles. Sin embargo, aunque hacer accesible el software complica y alarga ligeramente su diseño, desarrollar aplicaciones accesibles debería ser una práctica más extendida, ya que, con ello, se tiene en cuenta a grupos de gente que, de otra manera, no podrían hacer uso de muchas aplicaciones. En nuestro caso, nunca habíamos desarrollado software accesible antes de este proyecto, y ahora creemos que la accesibilidad debería ser un requisito obligado a la hora de desarrollar software.

Para el desarrollo de LiveDots, nos hemos apoyado en diferentes recursos de código abierto disponibles (como la biblioteca Manufaktura). Con ello, nos hemos dado cuenta de la importancia de este código para lograr programar software que, sin él, no se podría haber desarrollado.

En la prueba realizada (que no fue con usuarios finales debido a la pandemia), tuvimos en cuenta la importancia de que el cuestionario respondiera a nuestras preguntas de investigación. Para ello, tuvimos que ver qué partes de la aplicación iba a usar cada tipo de usuario y cómo las iba a usar. Los resultados obtenidos fueron muy positivos y, como hemos visto en la sección de trabajo futuro (sección 7), gracias a estos resultados la ONCE realizará un nuevo experimento involucrando al público objetivo de la aplicación con la intención de llevar LiveDots a las aulas.

Gracias al experimento, probamos empíricamente que LiveDots podría ser útil para favorecer la inclusión de alumnos invidentes en clases de música. Es decir, según los resultados

obtenidos en el cuestionario (sección 5.6.1), la aplicación favorecería la interacción entre un alumno ciego y un profesor vidente en un aula de música, lo que haría que este alumno estuviera incluido en mayor medida en la clase.

Los resultados de esta prueba mostraron que LiveDots puede ser usado por una persona vidente con conocimientos musicales que no sepa nada de braille musical. Gracias a esto, es posible que un profesor de música utilice la aplicación y genere una partitura en braille únicamente con sus conocimientos de música. Además, los resultados también probaron que una persona invidente puede utilizar la aplicación sin ayuda externa. Con todo esto, se concluye que LiveDots podría ser utilizado en clases de música, sin más herramientas que la propia aplicación, ya que tanto el profesor como el alumno invidente pueden utilizar la aplicación sin necesidad de ser ayudados.

Además, esta parte del proyecto nos hizo llegar a la siguiente conclusión que consideramos fundamental: no solo es importante que el software cuyo principal público objetivo es vidente se amplíe para ser accesible, sino también al revés. Es decir, que también es importante que aplicaciones cuyo público objetivo es gente invidente sean diseñada para poder ser usada fácilmente por gente vidente (que no está acostumbrada al mismo tipo de manejo de teclado y que usan herramientas visuales, como el ratón). Si no hubiéramos tenido en cuenta esto, nuestras pruebas de la aplicación habrían sido realmente complicadas, ya que la forma de utilizar el ordenador que tenemos la gente vidente es muy distinta a la de la gente invidente.

Como conclusión global del experimento, LiveDots es una aplicación interactiva en tiempo real relacionada con el campo de la música cuya utilidad y usabilidad han sido probadas empíricamente.

Por tanto, este proyecto ha demostrado que se puede generalizar la idea del proyecto EDICO de interactividad en tiempo real (traducción entre tinta y braille con modificaciones en tiempo real) a otros campos para lograr la interactividad entre personas videntes e invidentes y lograr así una mejor integración del alumno ciego en el aula. Esta generalización de un campo científico a un campo artístico como es el de la música, abre la veda a toda una serie de aplicaciones que lleven esta filosofía al resto de áreas del conocimiento.

Para difundir la idea de LiveDots, hemos escrito un artículo llamado *LiveDots: Real time Interactive Braille Music Translator to Integrate Blind Students into Music Classes.* El artículo, que está estructurado de forma similar a este trabajo de fin de grado, se centra en mayor medida en la parte experimental. En él se explican en detalle el desarrollo de la aplicación, el objetivo del experimento, el diseño experimental, los resultados y la discusión de la prueba realizada. El artículo se ha enviado al congreso congreso ICCE 2020 ($28^{th}$ *International Conference on Computers in Education*) que es un congreso Core B organizado por APSCE (*Asia-Pacific Society for Computers in Education*).

# Conclusions

Although there are tools that allow us to modify ink music scores and translate ink scores into Braille notation, their educational use is limited. However, in other fields of knowledge accessible real time interactive applications have been developed, such as EDICO (physics, chemistry and mathematics editor developed by ONCE). That is how the idea of LiveDots came up, a real-time interactive application that can facilitate the inclusion of blind music students in music classes using real time interactivity.

During the study of the state of the art we observed that there were not many accessible applications. However, although making accessible software may complicate and slightly lengthen its design phase, developing accessible applications should be a more widespread practice, as this takes into account groups of people who might otherwise not be able to use many applications. In our case, we had never developed accessible software before this project, and now we believe that accessibility should be a mandatory requirement when developing software.

For the development of LiveDots, we have relied on different open source resources (such as the Manufaktura library). This made us realized the importance of open-source code to achieve programming software that, without it, could not have been developed.

In the test conducted (which was not with target users due to the pandemic), we took into account the importance of the questionnaire answering our research questions. To do this, we had to see which parts of the application each type of user would use and how they would use them. The results obtained were very positive and, as we have seen in the future work section (section 7), thanks to these results the ONCE will carry out a new experiment involving the application's target audience with the intention of bringing LiveDots into the classroom.

Thanks to the experiment, we empirically proved that LiveDots can be useful to facilitate the inclusion of blind students in music classes. That is, according to the results obtained in the questionnaire (section 5.6.1), the application would ease the interaction between a blind student and a sighted teacher in a music class, which would include more this student in the class.

The results of this test showed that LiveDots can be used by a sighted person with musical knowledge and without any knowledge on musical Braille. As a result, it is possible for a music teacher to use the application and generate a braille music score with only his

or her music knowledge. In addition, the results also proved that a blind person can use the application without external assistance. All of this concludes that LiveDots can be used in music classes, with no other tools than the application itself, since both the teacher and the blind student can use the application without any external assistance.

Furthermore, this part of the project led us to the following conclusion which we consider fundamental: it is not only important that software whose main target audience is sighted is extended to be accessible, but also the other way around. That is, it is also important that applications whose target audience is blind people are designed to be easily used by sighted people (who are not used to the same type of keyboard management and who use visual tools, such as the mouse). If we hadn't taken this into account, testing the application would have been really complicated, as the way sighted people use the computer is very different from that of blind people.

As an overall conclusion of the experiment, LiveDots is a real-time interactive application related to the field of music which has been empirically proved useful and usable.

Therefore, this project has shown that the EDICO project idea of real-time interactivity (translation between ink and Braille with real-time modifications) can be generalized to other fields in order to achieve interactivity between sighted and blind people and thus achieve a better integration of the blind student in the classroom. This generalisation from a scientific field to an artistic field such as music opens the door to a whole series of applications that will bring this philosophy to other areas of knowledge.

In order to disseminate the idea of LiveDots, we have written an article called *LiveDots: Real time Interactive Braille Music Translator to Integrate Blind Students into Music Classes*. The article, which is structured in a similar way to this final dissertation thesis, is more focused on the experimental part. It explains in detail the development of the application, the objective of the experiment, the experimental design, the results and the discussion of the test carried out. The paper has been sent to the ICCE 2020 conference ($28^{th}$ *International Conference on Computers in Education*) which is a Core B conference organized by APSCE (*Asia-Pacific Society for Computers in Education*).

# Bibliografía


¿Qué es la ONCE? Historia de la Organización, empleo y acción. (s.f.). Recuperado el 25 de abril de 2020, desde https://www.once.es/conocenos

Abramo, J. M. & Pierce, A. E. (2013). An ethnographic case study of music learning at a school for the blind. *Bulletin of the Council for Research in Music Education*, (195), 9-24. https://doi.org/10.5406/bulcouresmusedu.195.0009

Accessibility in the Store - UWP applications — Microsoft Docs. (s.f.). Recuperado el 15 de junio de 2020, desde https://docs.microsoft.com/en-us/windows/uwp/design/accessibility/accessibility-in-the-store

Accessible calculators - ATWiki. (s.f.). Recuperado el 25 de abril de 2020, desde http://atwiki.assistivetech.net/index.php/Accessible%7B%5C_%7Dcalculators

Audacity Team. (s.f.). Opciones de Accesibilidad - Audacity Manual. https://ttmanual.audacityteam.org/man/Accessibility/es

Avid Technology. (s.f.). Sibelius: composition and notation software. Recuperado el 6 de abril de 2020, desde https://www.sibelius.com/helpcenter/article.php?id=444%7B%5C&%7Dsearchid=5143573

Benward, B. & Saker, M. N. (2009). *Music in theory and practice* (8.ª ed., Vol. 1). McGraw-Hill.

Bitteur, H. (s.f.). GitHub - Audiveris/audiveris: Latest generation of Audiveris OMR engine. https://github.com/Audiveris/audiveris

Borges, J. A. & Tomé, D. (2014). Teaching Music to Blind Children: New Strategies for Teaching through Interactive Use of Musibraille Software MicroFenix View project Mapavox View project. *Procedia - Procedia Computer Science*, *27*, 19-27. https://doi.org/10.1016/j.procs.2014.02.004

Braille Music Editor (BME). (s.f.). Recuperado el 6 de mayo de 2020, desde https://www.compartolid.es/braille-music-editor-bme/

Braille Music Software for Blind: Magnified Music for Low Vision by Dancing Dots. (s.f.). Recuperado el 6 de mayo de 2020, desde https://www.dancingdots.com/main/index.htm

Braille Translation Software - Index Braille. (s.f.). Recuperado el 25 de abril de 2020, desde https://www.indexbraille.com/en-us/support/braille-editors

Braille Translator. (s.f.). Recuperado el 15 de junio de 2020, desde https://www.brailletranslator.org/

Buhagiar, M. A. & Tanti, M. B. (2011). *WORKING TOWARD THE INCLUSION OF BLIND STUDENTS IN MALTA: THE CASE OF MATHEMATICS CLASSROOMS (MALTA'DAKİ GÖRME ENGELLİ ÖĞRENCILERIN KATILIMINI SAĞLAMAYA*

*YÖNELIK ÇALIŞMA: MATEMATIK DERSLERI ÖRNEĞİ)* (inf. téc. N.º 1). http://eku.comu.edu.tr/index/7/1/mabuhagiar%7B%5C_%7Dmbtanti.pdf

Burgos-Bordonau, E. (s.f.). Las musicografías de Abreu y Llorens: dos sistemas alternativos a la recepción del Braille en España.

Burkholder, J. P., Grout, D. J. & Palisca, C. V. (2008). *Historia de la Música Occidental*.

Campos-Arcaraz, T. (2017). A Proposal for a Music Writing for the Visually Impaired. Springer, Cham. https://doi.org/10.1007/978-3-319-47337-6_6

Capozzi, A., De Prisco, R., Nasti, M. & Zaccagnino, R. (2012). Musica parlata : AAA methodology to teach music to blind people, En *ASSETS'12 - Proceedings of the 14th International ACM SIGACCESS Conference on Computers and Accessibility*, New York, New York, USA, ACM Press. https://doi.org/10.1145/2384916.2384975

Carenas, J. M., Cabra, A. B., Mata-García, M. G., Gea, P. C. & Hernández, D. H. (2018). Prácticas Edico ¿ Qué es Edico ?, 100-108.

CESyA. Centro Español del Subtitulado y la Audiodescripción. (s.f.). Pautas de accesibilidad — CESyA — Centro Español del Subtitulado y la Audiodescripción. Recuperado el 25 de abril de 2020, desde https://www.cesya.es/comunicacion/pautas

Comunidad de desarrolladores de MuseScore. (s.f.). MuseScore Accessibility. https://musescore.org/en/handbook/3/accessibility

CTI. Centro de Tiflotecnología e Innovación de la ONCE. (s.f.). Recuperado el 15 de junio de 2020, desde http://cidat.once.es/

CTI. Editor Científico ONCE. (s.f.). Recuperado el 15 de junio de 2020, desde http://cidat.once.es/home.cfm?id=2351%7B%5C&%7Dnivel=2

DanLisMusic. (s.f.). Finale and Sibelius User Reviews - Composer's Toolbox. Recuperado el 21 de abril de 2020, desde https://composerstoolbox.com/2018/08/14/finale-sibelius-user-reviews/

Dannenberg, R. & Mazzoni, D. (s.f.). Audacity, editor de audio libre. Recuperado el 10 de junio de 2020, desde https://audacity.es/

Daron, V. (s.f.). GitHub - vdaron/MusicXml.Net: Quick C# parser for MusicXML. https://github.com/vdaron/MusicXml.Net

de Candé, R. (2002). *Nuevo diccionario de la musica*. Grasindo.

El protocolo y el formato MIDI. (s.f.). Recuperado el 6 de mayo de 2020, desde http://www.disca.upv.es/adomenec/IASPA/tema5/Midi.html

Encelle, B., Jessel, N., Mothe, J., Ralalason, B. & Asensio, J. (2009). *BMML: Braille Music Markup Language* (inf. téc.).

Eyewire News. (s.f.). ObjectiveEd Joins Microsoft's AI for Accessibility Program to Develop Braille AI Tutor Platform – Eyewire News. Recuperado el 25 de abril de 2020, desde https://eyewire.news/articles/objectiveed-joins-microsofts-ai-for-accessibility-program-to-develop-braille-ai-tutor-platform/

Frederick, b. W. (2009). *Quality of Experience in Mainstreaming and Full Inclusion of Blind and Visually Impaired High School Instrumental Music Students* (inf. téc.).

Freedom Scientific. (s.f.). JAWS® – Freedom Scientific. Recuperado el 3 de junio de 2020, desde https://www.freedomscientific.com/products/software/jaws/

Giuseppe Paccini, A. (s.f.). Bme2 Veia progetti... Recuperado el 5 de mayo de 2020, desde https://www.veia.it/en/bme2%7B%5C_%7Dproduct

Goldstein, D. (2000). Music pedagogy for the blind. *International Journal of Music Education*, *os-35*(1), 35-39. https://doi.org/10.1177/025576140003500112

Google TalkBack. (s.f.). Recuperado el 15 de junio de 2020, desde https://en.wikipedia.org/wiki/Google%7B%5C_%7DTalkBack

Gotoh, T., Minamikawa-Tachino, R. & Tamura, N. (2008). A web-based Braille translation for digital music scores, En *Proceedings of the 10th international ACM SIGACCESS conference on Computers and accessibility*, ACM.

GW Micro. (s.f.). WindowsEyes-GW Micro. Recuperado el 15 de junio de 2020, desde http://www.gwmicro.com/

Homenda, W. (2008). Breaking accessibility barriers: Computational intelligence in music processing for blind people. *Studies in Computational Intelligence*, *107*, 207-232. https://doi.org/10.1007/978-3-540-77662-8_9

Johnson, S. (2015). Understanding Is Seeing. https://doi.org/10.1093/OXFORDHB/9780199331444.013.7

Kent, D. (2012). *What is Braille?* Enslow Elementary.

Krolick, B. (1996). *New international manual of Braille music notation*. Braille Press Zurich.

Louis Braille - Wikipedia. (s.f.). Recuperado el 15 de junio de 2020, desde https://en.wikipedia.org/wiki/Louis%7B%5C_%7DBraille

MakeMusic. (s.f.). Finale — Music Notation Software That Lets You Create Your Way. https://www.finalemusic.com/

Moon type - Wikipedia. (s.f.). Recuperado el 25 de abril de 2020, desde https://en.wikipedia.org/wiki/Moon%7B%5C_%7Dtype

Musicología digital - Fundación Juan March. (s.f.). Recuperado el 28 de abril de 2020, desde https://www.march.es/bibliotecas/tme/musicologia/musicxml

MusicXML to Braille converter. (s.f.). Recuperado el 6 de mayo de 2020, desde https://musicxml2braille.appspot.com/

Naruedomkul, K. (2013). Aim-Math: An audio-based interactive media for learning mathematics.

NatBraille : un transcripteur Braille libre. (s.f.). Recuperado el 25 de abril de 2020, desde http://natbraille.free.fr/

Nicotra, G. & Quatraro, A. (2008). CONTRAPUNCTUS Project: A new computer solution for braille music fruition, Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-70540-6_45

NV Access. (s.f.). NVDA-NV Access. Recuperado el 15 de junio de 2020, desde https://www.nvaccess.org/about-nvda/

ObjectiveEd. (s.f.). Recuperado el 25 de abril de 2020, desde https://www.objectiveed.com/

ONCE. (s.f.). BRAITICO - Web de Educación de la ONCE. Recuperado el 30 de mayo de 2020, desde https://educacion.once.es/braitico

ONCE. (s.f.). Recuperado el 15 de junio de 2020, desde https://www.once.es/

Organización Nacional de Ciegos Españoles - Wikipedia, la enciclopedia libre. (s.f.). Recuperado el 25 de abril de 2020, desde https://es.wikipedia.org/wiki/Organizaci%7B%5C'%7Bo%7D%7Dn%7B%5C_%7DNacional%7B%5C_%7Dde%7B%5C_%7DCiegos%7B%5C_%7DEspa%7B%5C~%7Bn%7D%7Doles

Perkins School for the Blind. (s.f.). Recuperado el 25 de abril de 2020, desde https://www.perkins.org/

Quaglia, B. W. (2015). Planning for Student Variability: Universal Design for Learning in the Music Theory Classroom and Curriculum. *Music Theory Online*, *21*(1).

Repain, A., Marzin, A., Sacc, C., Royer, J., Lang, M., Kainz, M. S., Froment, N. & Loubatier, X. (s.f.). GitHub - mlang/freedots: MusicXML to Braille Music transcription. Recuperado el 1 de junio de 2020, desde https://github.com/mlang/freedots

Rugman, D. (s.f.). Sibelius Access. http://www.musicaccess.co.uk/sibelius-access/

Salamon, J. (s.f.). Manufaktura Controls - music engraving libraries for .NET. http://musicengravingcontrols.com/en-US/Home/

Schweer, W. (s.f.). MuseScore: software gratuito de composición y notación musical. https://musescore.org/es

Smaligo, M. A. (1998). Resources for Helping Blind Music Students. *Music Educators Journal*, *85*(2), 23-45. https://doi.org/10.2307/3399168

Talking Calculator on the AppStore. (s.f.). Recuperado el 25 de abril de 2020, desde https://apps.apple.com/us/app/talking-calculator/id424464284

Traductor braille : Traduce de Braille a Texto. (s.f.). Recuperado el 15 de junio de 2020, desde http://traductorbraille.com/

Una Breve Historia de los Sistemas de Escritura Táctil para Lectores con Ceguera e Discapacidades Visuales. (s.f.). Recuperado el 25 de abril de 2020, desde http://www.tsbvi.edu/seehear/spring06/history-span.htm

VoiceOver. (s.f.). Recuperado el 15 de junio de 2020, desde https://en.wikipedia.org/wiki/VoiceOver

W3C Music Notation Community Group. (s.f.). GitHub - tisimst/smufl: Standard Music Font Layout. https://github.com/tisimst/smufl

Wallin, N., Merker, B. & Brown, S. (2001). The origins of music.

Web de Educación de la ONCE. (s.f.). Recuperado el 15 de junio de 2020, desde https://educacion.once.es/

WebAIM. (s.f.). WebAIM: Screen Reader User Survey #7 Results. Recuperado el 15 de junio de 2020, desde https://webaim.org/projects/screenreadersurvey7/

ZoomText. (s.f.). Recuperado el 15 de junio de 2020, desde https://www.zoomtext.com/

# Apéndice A

# Artículo *LiveDots*

# LiveDots: Real time Interactive Braille Music Translator to Integrate Blind Students into Music Classes

**Claudia GUERRERO-GARCÍA-HERAS[a*], Óscar DÍAZ-RIBAGORDA[a], Lucas DE-TORRE-BARRIO[a], Borja MANERO[a*] & María GUIJARRO-MATA-GARCÍA[a*]**
[a]*Complutense University of Madrid, Spain*
*claudiaggh@gmail.com, bmanero@ucm.es, mguijarro@ucm.es

**Abstract:** Music is a universal cultural expression; however blind students may have difficulties to follow a sighted music class due to the lack of a common written music language between teacher and student. This problem is also present in other academic fields, like science. An approach to solve it is *EDICO*, a real time interactive Scientific Editor. We want to take the solution of real time interactivity to the music field. This paper presents the development process of a real time interactive solution called LiveDots and an experiment to test it in which we develop a real time interactive solution called LiveDots and test it with blindfolded users to check if it could help integration of a blind student in a music classroom. Results show that LiveDots was usable by both sighted and blindfolded users and that real time interactivity may be useful to integrate blind students in music class. These results open up a new horizon of solutions based on real time interactivity to ease blind and sighted students' integration in the same class.

**Keywords:** Accessibility, blind people, tiflotechnology, inclusive education, education, music, real time interactivity, Braille, Braille music, screen reader, Braille line, computer application.

## 1. Introduction

Braille is the most extended method of tactile reading and writing amongst the blind. It is based on different combinations of embossed dots (Kent, 2012). The extension of Braille to the music field is known as Braille music. It is a transcription technique that allows to represent any conventional music score with accurate notation (de Candé, 2002). Musical Braille not only gives blind students a tool to understand and express music but also shapes how we think and talk about music and, by extension, how we analyse it (Abramo & Pierce, 2013; Johnson, 2015).

However, blind students do not often receive the education needed to understand musical Braille notation and most schoolteachers do not know Braille neither how to facilitate the student's learning. The alternative for blind students in order to improve their musical skills is to attend a school for the blind. In this scenario, blind and non-blind students are taught using different teaching strategies leading to a dichotomy between music Braille and conventional music writing. The lack of familiarity with conventional music writing makes it more difficult for blind students to later join in an environment with sighted musicians, for example in music college or in an orchestra (Goldstein, 2000).

There is a need to integrate blind and non-blind students in the same classroom and teach every one of them the essential knowledge to develop their musical skills (Quaglia, 2015; Buhagiar & Tanti, 2011). In other words, we want blind students to be able to follow a music class with mostly sighted students while learning Braille music notation and understanding the conception of print music.

Some of the conventional strategies and tools used by blind or visually impaired students to facilitate participation in class are: enlarged print notation, fellow class members, parents or teachers reading the scores for them and use of Braille music notation whenever it is possible (Frederick & Moss, 2009; Smaligo, 1998). All these tools need either the help of a person who transcribes the score orally or a teacher who knows musical Braille and teaches it to the student which may not always be possible. This makes the student dependent on people around him.

The advance of technology in recent years has given the chance to ease the integration of blind students. There are projects like Braitico (*ONCE*, n.d.), an inclusive Braille literacy method developed by the *ONCE* (National Organization of the Blind in Spain) intended for children to learn Braille. In the field of music, there are computer programs designed to enable visually impaired people to view and edit music in Braille notation students (Homenda, 2008) like Braille Music Editor (Veia Progetti, n.d.), studies about how to teach Braille music to children using computer applications (Nicotra and Quatraro, 2008; Borges y Tomé, 2014), approaches to teach music to blind studies using talking music instead of Braille (Capozzi, Prisco, Nasti & Zaccagnino, 2012), score translation programs from print music into Braille music and vice versa like FreeDots (Repain et al, n.d.) and there is even a standard to share Braille music notation in the web called Braille Music Markup Language (Encelle, Jessel, Mothe, Ralalason & Asensio 2009).

These applications allow the teacher to create a print score with an editing score program, like MuseScore (Schweer, n.d.), and to translate it into Braille using a translation program so the student can read it. However, interactivity is missing: if the teacher wants to modify an element of the score, first, the changes should be done in the editing score program, then exported and translated into Braille with the translation program and finally the student could see the changes. This is a slow process that does not integrate a blind student into the usual development of a music class.

The problem of integrating blind students in the classroom is present in almost every field in education. There are many different approaches for a solution, like Aim-Math, an interactive-enhanced mathematics learning system for blind and visually impaired students using text-to-speech to read aloud math expressions (Naruedomkul, 2013). However, this approach is not friendly for a class of blind and non-blind students. Another approach is *EDICO* (Scientific Editor *ONCE*) (Carenas, Cabra, Mata-García, Gea & Hernández, 2018), a project promoted by the *ONCE* and developed in cooperation with the University Complutense of Madrid. It is an accessible mathematics, physics and chemistry editor. It translates scientific language in real time from printed writing into Braille and vice versa. This allows blind students to follow a science lesson interacting in real time with a teacher who doesn't know Braille.

After the success of *EDICO*, the *ONCE* wanted to develop similar solutions to integrate blind students in other subjects, like music. This is how the idea of LiveDots was born: a music editor that translates music scores from print music to Braille music in real time. LiveDots is an innovative desktop application which allows users to read a score in Braille and in print at the same time and to modify the print score and see the changes in the Braille score in real time. This would give blind students an interactive music learning environment.

In this paper we are going to study the effectiveness of using real time interactive applications like LiveDots in order to integrate a blind student in a sighted class. This paper is structured as follows: In section 2 we talk about the main objectives of the study and we propose three research questions we want to answer to. Section 3 illustrates the design of the application and gives an overview about its development. Section 4 introduces an experiment we are performing to check usefulness and usability. In section 5 we show the results from the experiment and in section 6 we discuss these results. Finally, in section 7 we describe the conclusions drawn from this project and illustrate the long way to go with the presented and future related projects.

## 2. Objectives

The main objective of this study was to check if the application LiveDots is useful to include a blind student in a music class. We believed that being able to translate scores into Braille in real time would increase the interaction between teacher and blind students improving blind students' participation. To verify this theory, we formulate the following research questions:

*RQ1. Can the application LiveDots help with the inclusion of a blind student in a music class?*

In addition, to test the utility of the application LiveDots, it is important to verify its usability. Therefore, another goal of this test was to verify usability for both target users: music teachers and blind music students.

On the one hand, it is required that a blind person can use the application LiveDots without any assistance for the application to be useful. Thus, the keyboard shortcuts and the screen reader must be practical and easy to use. To check this, we posed the following research questions:

*RQ2. Can a blind person use the application LiveDots without any assistance?*

On the other hand, a teacher should be able to use LiveDots even without knowledge on musical Braille. The actions loading a score and doing score changes have to be uncomplicated. Therefore, the last research question was:

*RQ3. Can a sighted person with musical background who doesn´t know musical Braille use the application LiveDots?*

## 3. Tool Design

### *3.1. Application requirements and characteristics*

Building on the previous requirements (a teacher should be able to use the application without any knowledge about musical Braille and blind students should be able to use the application without any assistance), we decided to implement the following features in LiveDots (Figure 1).

The application uses scores in MusicXML format since it is the most used musical representation format. When it is selected, the score is showed both in print and in Braille. The print score for sighted users is displayed by a stave and musical elements and the Braille score is shown in

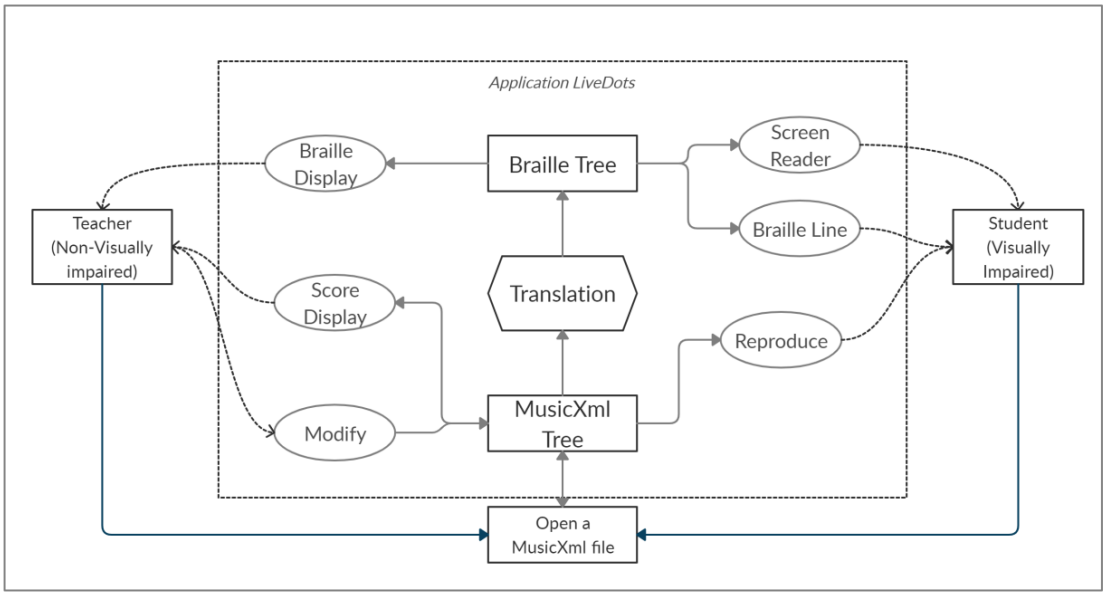



*Figure 1.* LiveDots Application Concept Design

Musical Braille notation to be read with a refreshable Braille display. On one side, a sighted user can edit the print score. These changes are shown in real time in the musical Braille score, so a blind student could read the new score in the refreshable Braille display at the same time it is modified. On the other side, a blind user can use keyboard shortcuts to use the application LiveDots and read the Braille score by a Braille line or using the screen reader: when placing the focus in the musical Braille score displayed on the screen, the screen reader will say the musical elements as you go through them. These features: screen reading of Braille musical elements and real time modification are an innovation of the application LiveDots. Other applications allow you to edit the score (for example, Musescore), but not in real time, or can be used with a screen reader (for example, Braille Music Editor), but they do not say each musical element in the Braille music score when you select them.

Lastly, the application LiveDots can be used by reduced visibility people and color-blinded people. The application allows to regulate the zoom (both in the scores and in the menu) and it is compatible with the Windows Colorblind Mode.

### *3.2 Interface Design*

In the design of the application we took into consideration two main factors: an easy way to move through the application window for the visually impaired user and an intuitive and useful interface for the non-visually impaired. The application´s main window is divided into three main sections (Figure 2):

1. Menu: This menu contains the main actions that can be taken while running the applications. It´s important to include a user's guide for a first-time user. We have also found that having keyboard shortcuts of the functions of the program is very useful for visually impaired users.
2. Print Score: In this area the print score will be displayed. This area will only be used by the non-visually impaired user so it´s important that it is not accessible by keyboard. Here the teacher would be able to see the score and make modifications of pitch if needed.
3. Braille Score: This is the area for the visually impaired user, in our case, the student. It's accessible by keyboard and it will read the elements displayed. Although it is not necessary to display the Braille score for the visually impaired user, we found that by displaying it, the application is a lot more intuitive for the non-visually impaired user and helps with the communication between the student and the teacher.

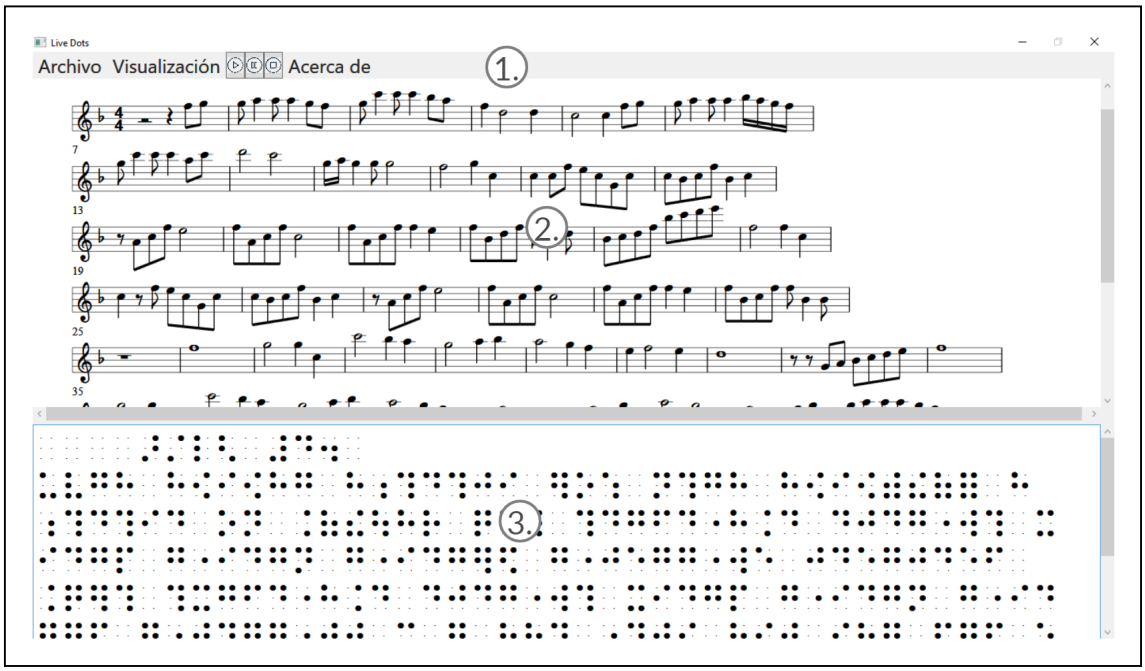



*Figure 2.* LiveDots Application Main Window.

These three sections are stacked vertically since this way the display of the printed score is optimized. The print score and the Braille score are divided by a movable separator in order to allow the desired configuration. It is also possible to change the size of the elements of any of these three sections or change all at the same time using the shortcuts CTRL+ "+" and CTRL+ "-".

### *3.3 Translation MusicXML to Braille*

After doing some research on the matter we found that there is a standard open format for digitalised sheet music. This is a Xml file with the MusicXml format. However, the Braille music is constructed

differently to the print music so a translation between the two is needed. We implemented a solution based on a study made on the matter (Goto, D., Minamikawa-Tachino, T. and Gotoh, N., 2007). In this solution we generate a tree for the print music and a different tree for the Braille music (Figures 3 and 4).

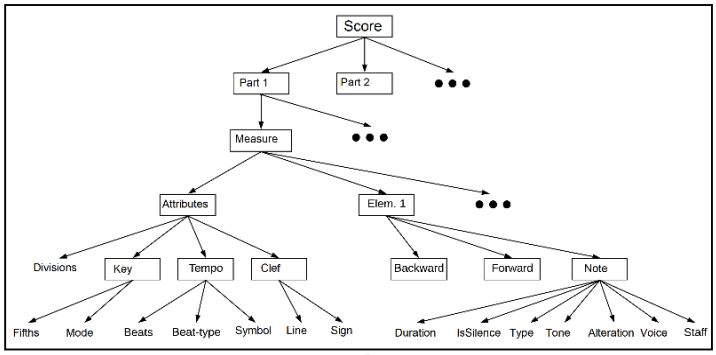



*Figure 3.* MusicXml Tree Structure.

The two trees differ mainly in the parent-child relation of the musical elements and some of the attributes of this elements. The translation is a conversion from the print tree to the Braille tree. For this, we walk through the first tree copying and adapting the information of each musical element into a Braille musical element. Then, we insert the Braille musical element into the Braille tree in the correct position given the Braille tree structure.

Having the Braille score in a tree structure is very useful. Each of the nodes (Braille musical elements) has a function that indicates how to write that element in musical Braille. This way, it is only necessary to walk through the branches and get the correspondent value to create the Braille score and display it. The Braille score is an array of characters that represent the different Braille symbols. In order to display the Braille score, we only need to use a font that assigns those characters to the correspondent Braille symbols. This same structure can be used to output the Braille score through a Braille line so the student can read it.

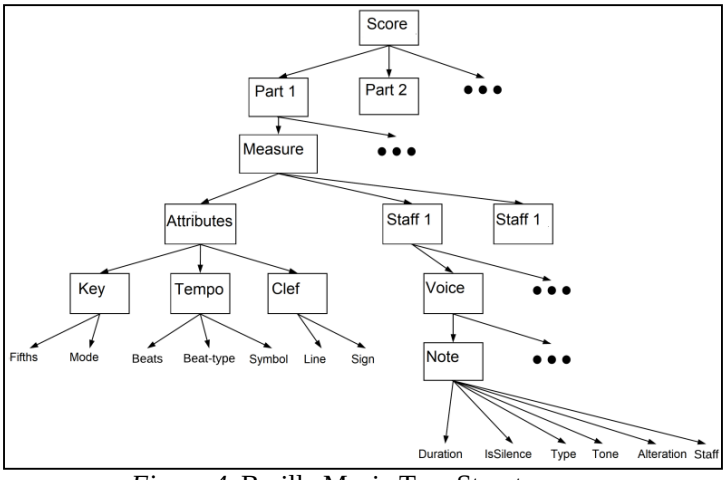



*Figure 4.* Braille Music Tree Structure.

### *3.4 Real time modification*

A combination of tools is used to implement a real time modification on the print score. We use a .NET library (Manufaktura.Controls) to display the print score. This library allows selection and movement of the displayed elements. When an element of the print score is moved, the score attribute that the library uses to display is updated to reflect the changes. In our application we use a mouse click event to communicate when a change to the print score has been made. In the event handler, we read the updated

print score and, through translation, update the Braille score. To communicate that update to the displayed Braille score, we used the binding functionality of C# which updates the displayed Braille score if the bound attribute is modified.

### *3.5 Reading Braille with Screen Reader*

The way the Braille score is represented (with characters) works well for a Braille line output and for a screen display of the Braille. However, a screen reader would read the displayed Braille score as the characters that the symbols represent. This is of no use and even counterproductive for the visually impaired user. For this reason, LiveDots includes the functionality of reading the musical elements that the Braille represents.

In order to do this, in the translation process, an array with the musical elements' names is created. This process is analogue to the creation of the Braille score and each element has a function that indicates the name of the musical element that it represents. This way we can override the behaviour of the screen reader JAWS, developed by Freedom scientific (JAWS – Freedom Scientific, n.d.), to read the elements from this array instead of the character represented.

An important detail must be added. The Braille musical representation of some elements requires, very often, more than one symbol to represent it. To deal with this, we also store the number of symbols that each element uses. Then, we override the movement of the cursor in the text block that represents the Braille score to jump by the size of the musical elements instead of jumping by the characters.

## 4. Methodology

### *4.1 Participants*

The study involved 7 participants. All of them were sighted people without any knowledge on Braille or Braille music. Each of them tested the application twice. They did it first blindfolded and then watching the application. Lastly, they did a questionnaire about LiveDots.

### *4.2 Experimental design*

The experiment was designed in order to test if LiveDots could help to include blind students in a music class. First, the participants checked the application and then they gave us feedback with a questionnaire (Figure 5).

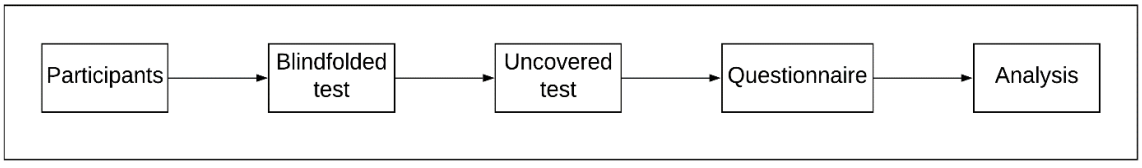



*Figure 5.* Experimental design.

The blindfolded part of the experiment checked if the application was usable by blind people. It was possible that some of the participants couldn't use the keyboard without seeing it, so there was a supervisor to help them press the keys they wanted to and to ensure that the participants were blindfolded while using the application. In this first part the participants had to open a score, play it, stop it, move in the Braille score to listen to the reading of musical elements through the screen reader and finally close the application. They used the screen reader JAWS while they were blindfolded (Figure 5).

On completion the blindfolded part, the participants uncovered their eyes and tested LiveDots from a sighted user point of view. The sighted tasks were to open a score, edit it, save the changes and identify those changes in the Braille music score using the screen reader.

The first part of the experiment tested if blind users were able to use the application LiveDots without any assistance (research question 2.2) and the second part tested if sighted people without any

knowledge on musical Braille were able to use the application (research question 2.3). Furthermore, testing the application as both target users allows the participants to evaluate whether LiveDots would be useful to include blind students in a music class (research question 2.1).
Lastly, they did a questionnaire with four parts, each one related with one of the research questions. The duration of the test was about 30 minutes, including the use of the application and the following questionnaire.

### *4.3 Material and Instruments*

For the experiment, each participant was provided the application LiveDots. In addition, they used the screen reader JAWS.

#### *4.3.1 Questionnaire*

The questionnaire had two initial yes/no questions. The first one was if getting Braille scores with informatic applications is easier than the traditional way of printing it on a piece of paper, and the second one was if the participant knew an application of translation or reading Braille scores different from LiveDots. Our hypothesis was that the answers would be yes and no, respectively. This would highlight the need of an application like LiveDots to do this translation and reading supporting the first research question (2.1).

After these two questions, the questionnaire consisted of four different parts. The first three of them were valued in Likert 5 scale, being 1 the worst scoring for the application LiveDots and 5 the best punctuation. Table 1 describes these three parts.

Finally, there was an open question about what improvements the participant would make to the application LiveDots.

Table 1. *Summary of questionnaires*

| Variable measured | Instrument | Type | How it is calculated | Number of items | Range |
|---|---|---|---|---|---|
| Usefulness of LiveDots | Blindfolded and sighted tests | 5-point Likert scale | Mean of the values of the items | 5 | 1-5 |
| Usability of LiveDots by blind people | Blindfolded test | 5-point Likert scale | Mean of the values of the items | 5 | 1-5 |
| Usability of LiveDots by sighted people | Sighted test | 5-point Likert scale | Mean of the values of the items | 5 | 1-5 |

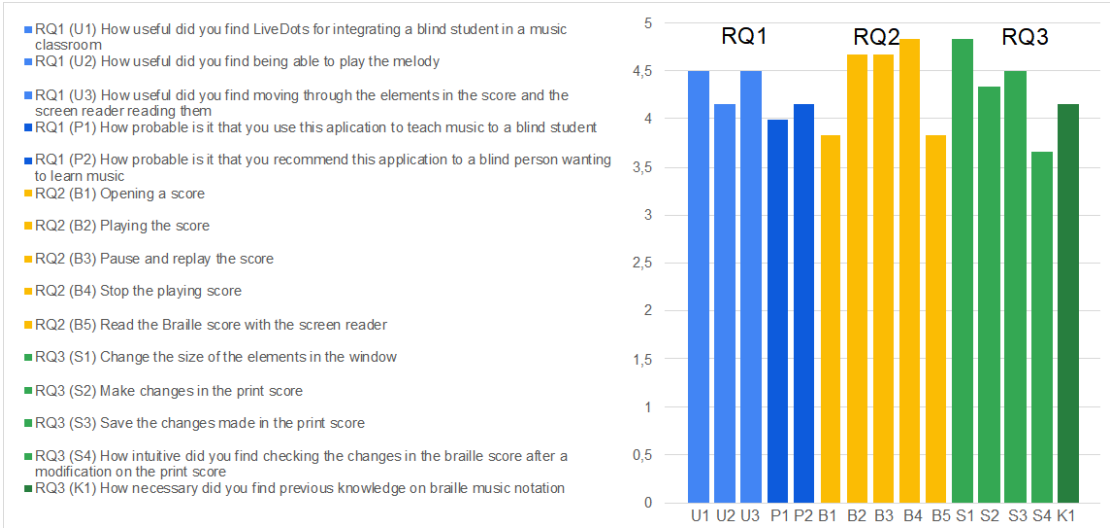



*Figure 6.* Results of the Likert 5 questions.

## 5. Results

After conducting the experiment, we obtained 7 answers for the blindfolded test questionnaire. We obtained a low answer rate due to the unexpected worldwide pandemic situation which did not allow us to conduct the experiment in the ONCE facilities. Having said that, we managed to get and analyse the results the best way possible. For the Likert 5 questions we calculated the mean of the answers obtained for each question and the results are the shown in the figure 6.

In the table there are three sections distinguished by three different colours. Each section is intended to help answer each research question in order. The first section (RQ1) contains the questions related to the usefulness of the application (U1-U3) and to the probability that the participants would use or recommend the application (P1-P2). The second section (RQ2) asks how easily the testers managed to do various tasks while blindfolded (B1-B5). The third section (RQ3) corresponds with the sighted part (non-blindfolded) of the test and asks (S1-S4) how easy participants found the different tasks and if previous knowledge on musical Braille was required (K1). For each group of related questions, we calculated the arithmetic mean of the average answers and represented it in a box plot (figure 7). Lastly, for the yes/no questions we obtained that 100% of the testers think that the use of computer applications makes obtaining a score in Braille easier than the traditional methods. And 85.7% of the testers don´t know any computer application for translation and/or reading of braille scores.

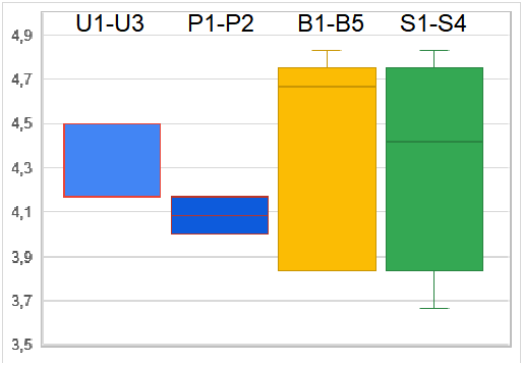



*Figure 7.* Box plot of the different sections studied.

## 6. Discussion

Throughout this section, we will try to answer the research questions set out in section 2 by discussing the evidence reported in the results section. First, we note the need of an application like LiveDots in the fact that although all of the participants (100%) think that the use of computer applications makes the task of obtaining Braille scores easier than the traditional method of printing on paper, hardly any of them (14%) know an application to accomplish it. As to test the utility of the application LiveDots (RQ1) it is necessary to test usability first (RQ2, RQ3), we will answer the research questions in that order.

**RQ2. Can a blind person use the application LiveDots without any assistance?**

Yes. The blindfolded test results show that the application can be easily used through the keyboard using JAWS as a screen reader (4.36/5 in blind usability B1-B5).

**RQ3. Can a sighted person with musical background who doesn´t know musical Braille use the application LiveDots?**

Yes. The sighted test results show that participants without any knowledge on Braille music could use the application without any trouble (4.33/5 in sighted usability S1-S4). Indeed, most of the participants (85%) think that little to no knowledge on Braille music is needed to use the application.

**RQ1. Can the application LiveDots help with the inclusion of a blind student in a music class?**

Yes. After performing the blindfolded and sighted test the participants think that the application and its functionalities can be useful for integrating a blind student in a music class (4.38/5 in usefulness

U1-U3) and they would probably use and recommend it (4.08/5 in P1-P2). In addition, RQ2 and RQ3 show that the developed application is also usable, which is a needed requirement for it to be useful.

These facts lead us to think that a real time interactive application like LiveDots could help integrate blind students in a music class. These findings also suggest that *EDICO*'s effectiveness in integrating blind students into science classes can be transferred to other fields apart from science and music.

## 7. Conclusions and future work

We developed LiveDots as an application to ease the communication between blind music students and sighted music teachers. The present study about provided empirical evidence that real time interactive applications can be used to integrate blind students in sighted music classes. In fact, LiveDots is an example in the music field that can be used to improve accessible education in our society.

The first conclusion is real time interactive applications, such as *EDICO* (application of Mathematics), can be generalized to artistic fields. Also, LiveDots design and the proposed experiment confirmed that the way blind people use computers is entirely different from how sighted users do it. Blind people use a screen reader and the keyboard to navigate with ease and use shortcuts repeatedly. On the other hand, sighted people typically use the mouse to navigate and do not usually know the place of keys.

This study had certain limitations. The instruments used were limited. The blindfolded experiment with sighted people gives an insight into answering our research questions, however the limited time available for the experiment and the global pandemic situation did not allow us to test the application with blind people. Non-sighted people could test the application deeper and check if habitual accessible procedures are followed in the application and if it is comfortable to use.

As a result of the success of the study, we are preparing a new experiment in cooperation with the *ONCE* to take this technology to the classroom. This will be a study involving blind people and music teachers to check the potential of the application LiveDots.

## Acknowledgements

This project has been partially founded by the Ministry of Science, Innovation and Universities of Spain (Didascalias, RTI2018-096401-A-I00) and it was supported by the *ONCE*-Tiflotechnology Chair.

We would also like to thank to the *ONCE* for their collaboration in this project, particularly to Pablo Carreño Gea, who was part of the *EDICO* project, for his cooperation in the development of LiveDots.

## References

Abramo, J. M., & Pierce, A. E. (2013). An ethnographic case study of music learning at a school for the blind. *Bulletin of the Council for Research in Music Education*, *195*, 9–24. https://doi.org/10.5406/bulcouresmusedu.195.0009

Buhagiar, M. A., & Tanti, M. B. (2011). *Working toward the inclusion of blind students in malta: the case of mathematics classrooms (malta'daki görme engelli öğrencilerin katilimini sağlamaya yönelik çalişma: matematik dersleri örneği)* (Vol. 7, Issue 1). http://eku.comu.edu.tr/index/7/1/mabuhagiar_mbtanti.pdf

Candé, R. de. (2002). *Nuevo diccionario de la musica / New dictionary of music*. Grasindo. http://books.google.com/books?id=4Dh0t9P5tqIC&pgis=1

Capozzi, A., De Prisco, R., Nasti, M., & Zaccagnino, R. (2012). Musica parlata : AAA methodology to teach music to blind people. *ASSETS'12 - Proceedings of the 14th International ACM SIGACCESS Conference on Computers and Accessibility*, 245–246. https://doi.org/10.1145/2384916.2384975

Carenas, J. M., Cabra, A. B., Mata-García, M. G., Gea, P. C., & Hernández, D. H. (2018). *Prácticas EDICO ¿ Qué es EDICO ?* 100–108.

Dos, J. A., & Borges, S. (2014). Teaching Music to Blind Children: New Strategies for Teaching through Interactive Use of Musibraille Software MicroFenix View project Mapavox View project. *Procedia - Procedia Computer Science*, *27*, 19–27. https://doi.org/10.1016/j.procs.2014.02.004

Encelle, B., Jessel, N., Mothe, J., Ralalason, B., & Asensio, J. (2009). BMML: Braille Music Markup Language. In *The Open Information Systems Journal* (Vol. 3).

Frederick, by W. (2009). *Quality of Experience in Mainstreaming and Full Inclusion of Blind and Visually Impaired High School Instrumental Music Students*.

Goldstein, D. (2000). Music pedagogy for the blind. *International Journal of Music Education*, *os-35*(1), 35–39. https://doi.org/10.1177/025576140003500112

Goto, D., Minamikawa-Tachino, T., & Gotoh, N. (2007). A transcription system from MusicXML format to Braille music notation. Eurasip Journal on Advances in Signal Processing, 2007(1), 1–9.

Homenda, W. (2008). Breaking accessibility barriers: Computational intelligence in music processing for blind people. *Studies in Computational Intelligence*, *107*, 207–232. https://doi.org/10.1007/978-3-540-77662-8_9

JAWS – Freedom Scientific. (n.d.). Retrieved May 30, 2020, from https://www.freedomscientific.com/products/software/jaws/

Johnson, S. (2015). *Understanding Is Seeing*. https://doi.org/10.1093/OXFORDHB/9780199331444.013.7

Kent, D. (2012). *What is Braille?*. Enslow Publishing, LLC.

Koelsch, S. (2013). From Social Contact to Social Cohesion—The 7 Cs. *Music and Medicine*, *5*(4), 204–209.

Naruedomkul, K. (2013). *Aim-Math: An audio-based interactive media for learning mathematics*.

Nicotra, G., & Quatraro, A. (2008). CONTRAPUNCTUS Project: A new computer solution for braille music fruition. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *5105 LNCS*, 303–309. https://doi.org/10.1007/978-3-540-70540-6_45

*ONCE*. (n.d.). *BRAITICO - Web de Educación de la ONCE*. Retrieved May 30, 2020, from https://educacion.*ONCE*.es/braitico

Quaglia, B. W., (2015). Planning for Student Variability: Universal Design for Learning in the Music Theory Classroom and Curriculum. *Music Theory Online*, *21*(1).

Repain, A., Marzin, A., Sacc, C., Royer, J., Lang, M., Kainz, Ms. S., Froment, N., & Loubatier, X. (n.d.). *GitHub - mlang/freedots: MusicXML to Braille Music transcription*. Retrieved June 1, 2020, from https://github.com/mlang/freedots

Smaligo, M. A. (1998). Resources for Helping Blind Music Students. *Music Educators Journal*, *85*(2), 23–45. https://doi.org/10.2307/3399168

Veia Progetti. (n.d.). *Bme2*. Retrieved May 5, 2020, from https://www.veia.it/en/bme2_product

Werner Schweer. (n.d.). *Software gratuito de composición y notación musical | MuseScore*. Retrieved May 30, 2020, from https://musescore.org/es