Universidad Zaragoza

Trabajo Fin de Grado.
Grado en Ingeniería Informática

Araword móvil.
Editor de picto-textos para dispositivos móviles.

Araword mobile.
Pict-text editor for mobile devices.

Autor/es

Diego Ceresuela Arrazola

Director/es

Joaquín Ezpeleta Mateo

Universidad de Zaragoza / Escuela de Ingeniería y Arquitectura

2016

Resumen

Araword móvil.
Editor de picto-textos para dispositivos móviles.

AraWord es una aplicación informática de libre distribución consistente en un procesador de textos que permite la escritura simultánea de texto y pictogramas, facilitando la elaboración de materiales de comunicación aumentativa, la elaboración de materiales curriculares accesibles, y la adaptación de documentos para personas que presentan dificultades en el ámbito de la comunicación funcional y de la lectoescritura.

Actualmente Araword sólo cubre el sector de los ordenadores personales. Sin embargo, el enorme crecimiento de la tecnología móvil y las redes sociales ha generado la necesidad de desarrollar Araword móvil (en adelante AWm). Este TFG ha consistido en la creación de la versión móvil de Araword, introduciendo además diversas mejoras y añadiendo funcionalidades demandadas por los usuarios.

AWm es la primera aplicación para *smartphones* que permite utilizar los sistemas de mensajería instantanea para enviar documentos basados en texto y pictogramas. De esta manera, se han eliminado barreras que impedian a los usuarios de sistemas alternativos y aumentativos de la comunicación el uso de estas aplicaciones de manera efectiva. Es una herramienta altamente configurable en cuanto a la visualización de los documentos y la limitación de funcionalidades, ya que existe un amplio rango de usuarios con distintas dificultades que pueden o no estar capacitados para usar o comprender determinados aspectos de la aplicación. Además permite que el usuario (o en su defecto el educador) introduzca nuevos pictogramas para usarlos localmente, añadiendo así un grado más de personalización. Por último, se ha realizado un esfuerzo por mantener la compatibilidad entre Araword y AWm en lo que a los documentos se refiere, permitiendo de esta manera compartir de forma efectiva documentos entre las distintas plataformas.

Durante todas las fases del proyecto se ha contado con un experto logopeda con amplia experiencia en el sector, que ha aportado el punto de vista del usuario y los requisitos del mismo. Aún así, debido a las características del usuario, se ha realizado un amplio periodo de pruebas con expertos que se ha extendido durante 3 meses. La aplicación se ha distribuido a grupos de expertos formados por logopedas, educadores y/o personas con experiencia en el sector TIC para la comunicación aumentativa y alternativa. Estos grupos empezaron siendo reducidos y se ha ido incrementando el numero de expertos conforme avanzaba el desarrollo, generando así un producto de calidad.

Índice general

1. Introducción

1.1. Conocimientos previos

1.1.1. Sistemas aumentativos y alternativos de comunicación

Los Sistemas Aumentativos y Alternativos de Comunicación (SAAC) son formas de expresión distintas al lenguaje hablado, que tienen como objetivo aumentar (aumentativos) las capacidades comunicativas y/o compensar (alternativos) las dificultades de comunicación y lenguaje de personas con discapacidad [1].

La utilización de SAACs no es excluyente del uso del lenguaje común sino que puede ayudar a reforzar algunas carencias y pulir defectos. De hecho los SAACs también pueden utilizarse para ayudar a los niños pequeños a iniciarse en la lecto-escritura, tanto en la lengua materna como en una extranjera.

1.1.2. Pictograma

Un pictograma es una representación simbólica de una palabra o grupo de palabras llamado término, tiene un significado único y sirve para identificar inequívocamente el término que representa. Aparte de la imagen el pictograma puede contener otra información relativa, por ejemplo, al tipo de palabra que representa.

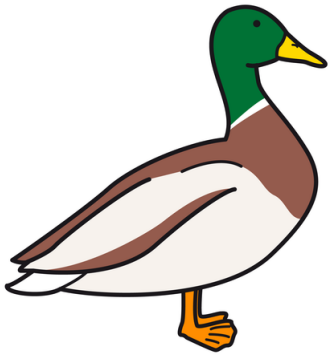

Figura 1.1.: Ejemplo de pictograma para *pato*.

Estos pictogramas se utilizan acompañados de los términos que representan para crear documentos de picto-texto. Además de la propia imagen también se puede representar de manera visual otra información, como el tipo de palabra, utilizando por ejemplo, bordes de colores.

1.1.3. Araword

AraWord es una aplicación informática de libre distribución consistente en un procesador de textos que permite la escritura simultánea de texto y pictogramas, facilitando la elaboración de materiales de comunicación aumentativa, la elaboración de materiales curriculares accesibles, y la adaptación de documentos para personas que presentan dificultades en el ámbito de la comunicación funcional y de la lectoescritura. [2]

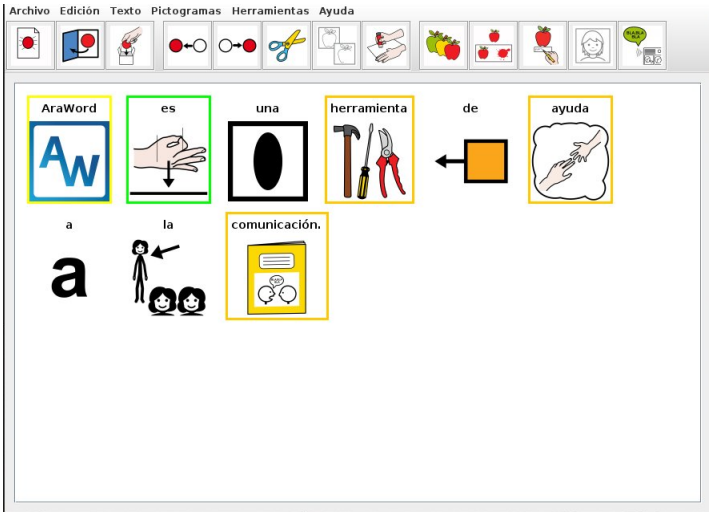

Figura 1.2.: Picto-texto en Araword

En el ámbito de los ordenadores personales Araword es uno de los productos de ayuda a la comunicación más usado debido a la flexibilidad a la hora de crear documentos y las múltiples funcionalidades que ofrece. Algunas de ellas se listan a continuación.

- Alternar entre los distintos pictogramas que representan un término.
- Dividir una palabra compuesta en sus términos simples y viceversa.
- Cambiar el contenido de un texto sin variar el pictograma.
- Procesar textos copiados y pegados convirtiendolos en picto-texto.

Sin embargo, presenta varias limitaciones:

- La expansión de la tecnología móvil ha supuesto la apertura de un sector al que Araword no estaba destinado, desaprovechando de esa manera una gran oportunidad.
- No existe una integración de la aplicación con las redes sociales, por lo que compartir los picto-textos supone un esfuerzo extra.

- El proceso de actualización de los pictogramas en Araword es tedioso ya que incluye a terceras personas no dedicadas al proyecto, que tienen que crear la nueva base de datos de manera manual. Además no existe la posibilidad de descargar sólo los nuevos pictogramas, lo cuál supone un consumo de ancho de banda muy alto en el momento de actualizar.

1.2. Objetivos del proyecto

El primer objetivo consiste en desarrollar AWm manteniendo la flexibilidad de Araword en cuanto a configuración y libre escritura de picto-texto, integrando la aplicación con las redes sociales, añadiendo todas aquellas funcionalidades, útiles, ofrecidas por la plataforma y eliminando todas aquellas que ahora no tengan utilidad.

Por otro lado, el segundo objetivo es desarrollar un servicio de gestión de pictogramas, que permita la descarga parcial de los pictogramas por parte de la aplicación, reduciendo así el tiempo de actualización (aspecto clave para las plataformas móviles) y que introduzca procedimientos automáticos para la adición, modificación y eliminación de pictogramas de la base de datos. De esta manera, se consigue suprimir el trabajo manual y la intervención de terceros ajenos al proyecto.

Las partes I y II de esta memoria describen el proceso seguido para el desarrollo de las herramientas que cubren los objetivos citados anteriormente.

Parte I.

Araword

2. Análisis del problema

Se ha acudido al Colegio público de educación especial (CPEE) Alborada para mediante la observación directa y la entrevista con varios logopedas conocer en detalle cuales son las limitaciones del usuario y qué características tiene que tener la nueva aplicación. Posteriormente se han fijado los requisitos del producto haciendo especial hincapié en los criterios de usabilidad.

2.1. Requisitos

El análisis de requisitos se realizó tras la primera visita al CPEE Alborada en colaboración con uno de los profesores y el director del proyecto. Algunos de los más importantes se muestran a continuación (para ver todos los requisitos y su clasificación ir al apéndice A.2).

Código	Requisito
RF1	Se debe poder generar documentos basados en picto-texto.
RF2	Los documentos podrán guardarse para ser recuperados posteriormente.
RF3	Se podrá enviar un documento entre dispositivos de tal manera que el documento recibido se pueda modificar como uno guardado.
RF4	Se permitirá compartir un documento a través de las redes sociales y aplicaciones de mensajería instantánea.
RF6	Se permitirá crear nuevos términos con su correspondiente pictograma.
RNF1	Los documentos mantendrán el formato de Araword PC.
RNF2	Los idiomas soportados por la aplicación serán, al menos, los mismos que para los documentos.
RNF4	Se podrán limitar las acciones realizables desde un modo administrador.
RNF5	Se permitirá personalizar el tamaño de los pictogramas, tamaño de la letra, escala de grises del pictograma, velocidad del lector y el color del borde asociado a cada tipo de palabra.

Cuadro 2.1.: Resumen de los requisitos

2.2. Araword

Se ha hecho un análisis en profundidad de Araword para comprender cómo funciona, determinar cuales son los principales problemas que se van a tener que afrontar y concer el formato de documentos de Araword.

2.3. ¿Cómo funciona?

La herramienta tiene, en rasgos generales, que detectar cada pulsación del teclado y modificar, si fuera necesario, los pictogramas que se visualizan junto al texto. Para ello los pasos a seguir son los siguientes.

1. Detectar en qué término se ha producido el cambio.
2. Formar un contexto con los términos colindantes ya que un cambio en una palabra puede resultar en una expresión compuesta. Por ejemplo el texto "de olor" está compuesto por dos términos simples (de y olor) con sus respectivos pictogramas mientras que "de color" es un único término con su pictograma.
3. Crear todos los posibles compuestos, que incluyan el término que ha cambiado, en el contexto y ordenarlos por longitud (número de palabras) de mayor a menor.
4. Buscar en la base de datos de verbos conjugados.
   a) Si el término es una forma conjugada sustituir en la búsqueda el término compuesto por su forma en infinitivo.
5. Buscar en la base de datos de términos.
   a) Si el término existe se le asignan los pictogramas que le corresponden, eliminar todos los términos simples del documento y cambiarlos por el término compuesto.
   b) Si no existe pasar al siguiente término (de menor longitud) hasta llegar a uno que exista, o bien, llegar al término de longitud uno inexistente, en este caso se le asigna un pictograma en blanco.

Analizando el funcionamiento se pueden deducir diversos problemas que tendrán que afrontarse. Algunos de los más importantes son los siguientes.

- La pulsación de una tecla dispara una serie de acciones (búsquedas en las bases de datos) costosas en tiempo, lo que puede generar problemas de concurrencia al pulsar teclas más rápido de lo que se tarde en buscar los términos en la base de datos. Bloquear la escritura mientras se busca no es una opción porque entorpecería la escritura del usuario.

- En el momento de introducir los nuevos términos compuestos habrá que eliminar los simples, aunque pueden haberse disparado otros cambios. Es decir, el documento puede haber variado mientras que se busca en las BDs, por lo que habrá que encontrar la manera de saber dónde introducir los resultados.

Otro aspecto fundamental es la existencia de dos bases de datos, una con los términos y la información relativa y otra/s con los verbos conjugados del idioma elegido. Trás realizar un análisis se ha concluido que, dado que la tecnología elegida en su momento para desarrollar las bases de datos de Araword es completamente portable a la plataforma móvil, se van a reutilizar. En el apéndice A.3.2 se detalla el análisis de las bases de datos.

2.3.1. Documentos AWZ

Dado que uno de los requisitos es mantener la compatibilidad con la versión de PC, se ha realizado un examen exhaustivo del formato de documentos usado por Araword.

El formato de documentos de Araword tiene la extensión AWZ y consiste en un fichero comprimido (*ZIP*) que contiene por un lado un archivo "base.awd" con la información del documento en *XML* (el orden de los términos, el idioma del documento, el color de los tipos de palabras, la fuente usada, etc...) y por otro una carpeta "exportbbdd". Esta carpeta almacena la información de los pictogramas: por un lado todas las imágenes y por otro, un archivo XML llamado "images.xml" con la información asociada a los pictogramas. En el apéndice A.3.1 se encuentran los DTDs así como ejemplos de los ficheros XML.

2.4. Paradigmas

Se ha de revisar el estado del arte para conocer las estrategias y tecnologías disponibles para elegir la que mejor se adecúe al problema expuesto. En lo referente a las estrategias se han identificado cuatro paradigmas distintos: nativo, multi-plataforma, web e híbrido. Para cada uno de ellos se han analizado las ventajas y desventajas de su uso.

2.4.1. Desarrollo nativo

Un desarrollo nativo supone la utilización del más bajo nivel ofrecido por la plataforma para construir la aplicación usando para ello el framework ofrecido por los desarrolladores de la plataforma.

A favor	En contra
Disponibilidad de una API propia de la plataforma sobre la que desarrollar.	Cada plataforma requiere de una implementación independiente.
Existencia de un *SDK* específico, librerías, *frameworks*, compiladores y depuradores desarrollados para dicha plataforma.	Tecnología específica y curva de aprendizaje pronunciada en caso de que no se conozca el lenguaje de programación.
Control casi completo sobre el funcionamiento de la aplicación.	Requiere de instalación y al menos un dispositvo físico donde depurar.
Mayor rendimiento, ya que se controla el hardware del dispositivo al más bajo nivel posible.	Dificultad para implementar una arquitectura u orgranización distinta de la predefinida por la plataforma.
Mejor interacción y adaptación por parte del usuario, ya que este no sale del entorno de ejecución.	Coste de mantenimiento y actualización mayor, especialmente si se soportan múltiples plataformas.

Cuadro 2.2.: Pros y contras del desarrollo nativo

2.4.2. Desarrollo multi-plataforma

El desarrollo multi-plataforma es básicamente igual al nativo pero se utiliza un compilador, provisto por un tercero, que permite realizar una compilación cruzada entre varias plataformas.

A favor	En contra
Un único desarrollo permite a la aplicación funcionar sobre otras plataformas especificadas por los propietarios del *framework* utilizado.	Dependencia del proveedor del compilador, no se puede sacar una versión para un dispositivo no soportado o una nueva plataforma.

Cuadro 2.3.: Pros y contras del desarrollo multi-plataforma

2.4.3. Desarrollo web

El desarrollo web consiste en la creación de una página web que permita simular un entorno de aplicación genérico de las distintas plataformas.

A favor	En contra
Simplicidad y variedad en la tecnología de desarrollo, se puede elegir cualquier tecnología web. Curva de aprendizaje suave.	Acceso limitado o inexistente al *hardware*, imposibilidad de almacenar un estado persistente en el cliente o utilizar funciones propias del dispositivo.
Facilidad de desarrollo, no requiere de instalación de *SDK*, se puede trabajar sin librerías de terceros, incluso sin las propias de la plataforma.	Requerimiento de una conexión a internet continua. No existe la posibilidad de un modo offline ni siquiera con prestaciones reducidas.
Mayor capacidad para encontrar documentación que permita seguir un estandar abierto.	Imposible de instalar en un dispositivo o publicar en una tienda virtual.
Multi-plataforma, un único desarrollo permite a la aplicación funcionar sobre diferentes plataformas.	Incapacidad de alcanzar el rendimiento del código nativo, sobre todo en la visualización de animaciones.
Capacidad para obtener métricas sobre tus usuarios, se conoce perfectamente el número de visitas así como el origen de las mismas.	Dependencia íntegra del navegador y necesidad de implementar y testear la aplicación para distintos navegadores desde el principio.
Desarrollo rápido, mayor tiempo para perfeccionar la aplicación o integrar nuevas funcionalidades.	Necesidad de un servidor web que soporte la apliación, realice los procesos y contenga la base de datos.

Cuadro 2.4.: Pros y contras del desarrollo web

2.4.4. Desarrollo híbrido

El desarrollo híbrido utiliza librerías de terceros que realizan la traducción entre el lenguaje dinámico web y el lenguaje nativo, permitiendo generar una aplicación nativa con una interfaz web.

A favor	En contra
Curva de aprendizaje ligera para desarrolladores web.	Rendimiento limitado a una visualización web.
Puede instalarse y desplegarse en la tienda virtual de la plataforma.	-
Un único código para múltiples plataformas.	-
Acceso casi completo al hardware del dispositvo, asi como a las APIs.	-

Cuadro 2.5.: Pros y contras del desarrollo híbrido

2.4.5. Conclusión

Se ha optado por descartar un desarrollo nativo debido al alto coste en tiempo de realizar y documentar ambas aplicaciones ya que el tiempo es limitado y se desea añadir funcionalidades a la aplicación. El desarrollo web queda descartado dado que uno de los requisistos es que la aplicación funcione sin conexión a internet. Finalmente se realizará un desarrollo híbrido.

2.5. Tecnología

En cuanto a las tecnologías disponibles para el desarrollo híbrido, debido al hecho de que es una tecnología en crecimiento y de fácil prototipado, se ha elegido Ionic, una plataforma para desarrollo híbrido basada en AngularJS y Cordova.

AngularJS sostiene la parte web ofreciendo un paradigma de progamación modelo-vista-controlador. El modelo–vista–controlador (MVC) es un patrón de arquitectura de software, que separa los datos y la lógica de negocio de una aplicación de la interfaz de usuario y del módulo encargado de gestionar los eventos y las comunicaciones. Para ello MVC propone la construcción de tres componentes distintos que son el modelo, la vista y el controlador: por un lado define componentes para la representación de la información, y por otro lado para la interacción del usuario. Este patrón arquitectural se basa en las ideas de reutilización de código y la separación de conceptos, características que buscan facilitar la tarea de desarrollo de aplicaciones y su mantenimiento [4].

Las vistas se implementan con HTML y CSS y la lógica con JavaScript. Cordova es un *software* que ofrece la posibilidad de acceder a las funciones nativas del dispositivo a través de JavaScript. Las dos en conjunto ofrecen toda la tecnología necesaria para llevar a cabo un desarrollo híbrido.

3. Diseño de la solución

3.1. Prototipado

3.1.1. En papel

Se han creado, junto al experto logopeda, prototipos en papel que han permitido diseñar las pantallas más complejas y la navegación entre las mismas. El hecho de contar con un experto en las fases tempranas del diseño ha supuesto no tener que modificar el diseño a lo largo de todo el proyecto y ha sido un aspecto clave.

Aunque la fase de análisis ha concluido en este momento se sigue sin tener un conocimiento profundo del usuario final, por lo que contar con un experto es primordial. En el apéndice A.5 pueden encontrarse los prototipos.

3.1.2. Software

Debido al desconocimiento previo de la tecnología y con intención de comprobar si el problema era resoluble de manera eficiente, se decidió empezar con un prototipo vertical. La funcionalidad elegida fue la expuesta en el RF-10. La aplicación debería ser capaz de generar documentos basados en picto-texto.

Una vez completado el prototipo vertical se ha procedido a realizar otro horizontal que incluya todas las pantallas y opciones, pero sin tener funcionalidad ninguna. Este prototipo ha sido modificado múltiples veces, para corregir detalles

Al realizar prototipos en *software* nos hemos arriesgado a dar la impresión de tener un producto muy pulido cuando aún es muy pronto. Sin embargo la estrecha comunicación con el director del TFG y el experto logopeda lo han evitado. Poder controlar las expectativas nos ha permitido aprovecharnos de la facilidad de prototipado ofrecida por la tecnología elegida.

3.2. Esquemas

3.2.1. Estructural

El paradigma de programación MVC fuerza al menos tres capas: modelo, vista y controlador. Sin embargo, por claridad, los servicios/modelos se han tratado de estructurar en dos capas: por un lado los elementos para acceder a las bases de datos y por otro el resto de modelos. El desconocimiento de la tecnología ha hecho que el *software* pase por varias etapas, estructuralmente hablando. La estructura final es la siguiente.

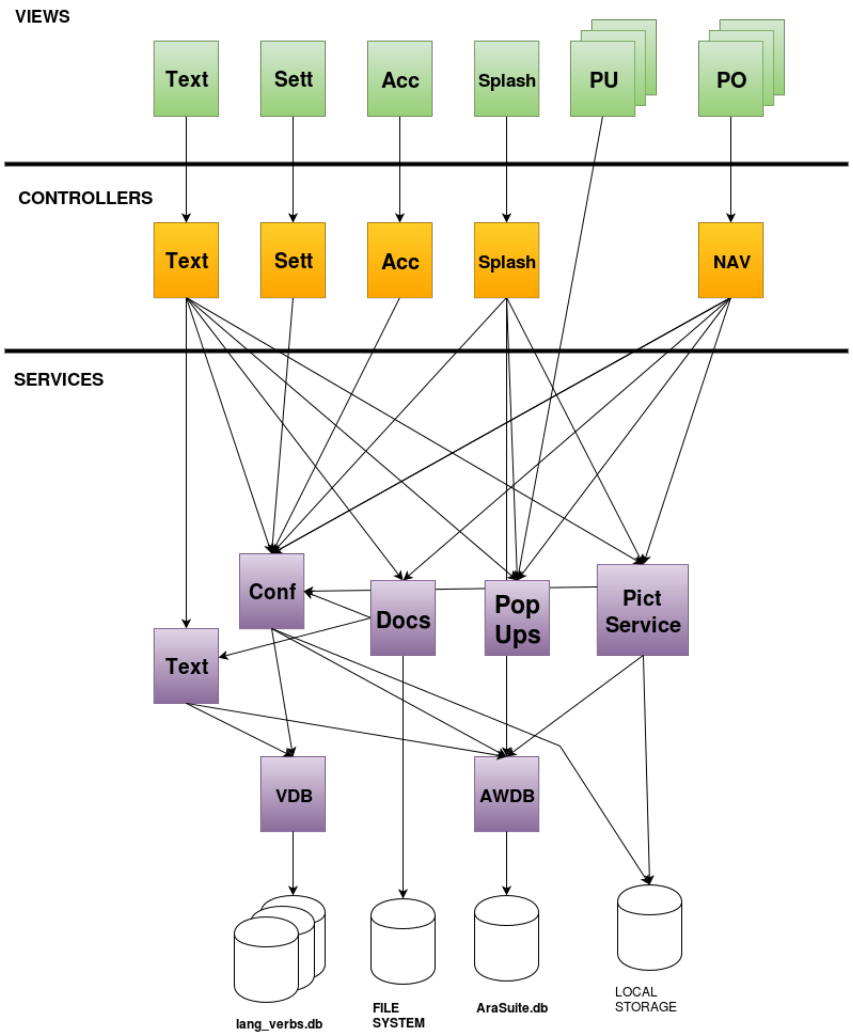

Figura 3.1.: Estructura final

Las vistas corresponden a cada una de las pantallas de la aplicación y las ventanas emergentes. Los controladores implementan parte de la lógica sencilla y hay uno por cada vista a excepción de los popups que se han tratado de unificar en un único servicio proveedor de popups. Por último los servicios contienen la lógica pesada y permiten el acceso al almacenamiento del dispositivo, ya sean las bases de datos o el almacenamiento local (necesario para guardar documentos y parámetros de configuración). En el apéndice A.6 se pueden ver los diferentes esquemas y el progreso seguido.

3.3. Representación de la información

Ha habido que definir los tipos de datos que se iban a usar así como su representación posterior en las vistas. Esta representación no ha variado desde la primera etapa de diseño aunque se ha añadido alguna información posteriormente.

3.3.1. Modelo del término

Cada uno de los términos tiene una serie de información asociada:

- Un identificador único que no sólo se usa para localizar el término en el modelo sino que identifica tambien el contenedor HTML que representa dicho término.
- Una lista de pictogramas y un índice que indica qué pictograma está actualmente en uso. Para cada uno de los pictogramas:
  - El nombre de la imagen.
  - El tipo de palabra, por ejemplo "casa" es un nombre común y una forma conjugada del verbo "casar". Tendrá los pictogramas de ambos y serán distintos tipos.
  - La imagen leída en codificación base64 o nada si el pictograma no ha sido visualizado aún.
- El número de palabras que componen el término.

En JavaScript, la representación de datos se hace por defecto utilizando JSON (un formato de texto ligero para el intercambio de datos). Una descripción formal del tipo de dato término es la siguiente:

```
{
    'id': String,           /* Identificador único */
    'value': String,        /* El término en si ej: "de color" */
    'pictos': [{
        'picto':String,     /* ej: "62243.png" */
        'type':Number,      /* El tipo de palabra 0-5 */
        'base64':String     /* El picto leido en base64 */
    }],                     /* Todos los pictogramas asociados al término */
    'pictInd': Number,      /* El pictograma en uso de la lista */
    'words': Number        /* Número de palabras del término ej: 2 */
}
```

3.3.2. Modelo/vista del documento

Así pues, el documento es una lista de términos. Para cada uno de los términos (existentes en el controlador) existe una representación en HTML en la vista, de tal manera que:

1. La imagen en base64 del pictograma seleccionado es el contenido de una etiqueta *img*. Redimensionada según la configuración.
2. El propio término es el contenido de una etiqueta *input*.
3. El tipo de palabra define el color del borde del pictograma. Es decir el borde de la etiqueta *div* que contiene la imagen y el *input*.

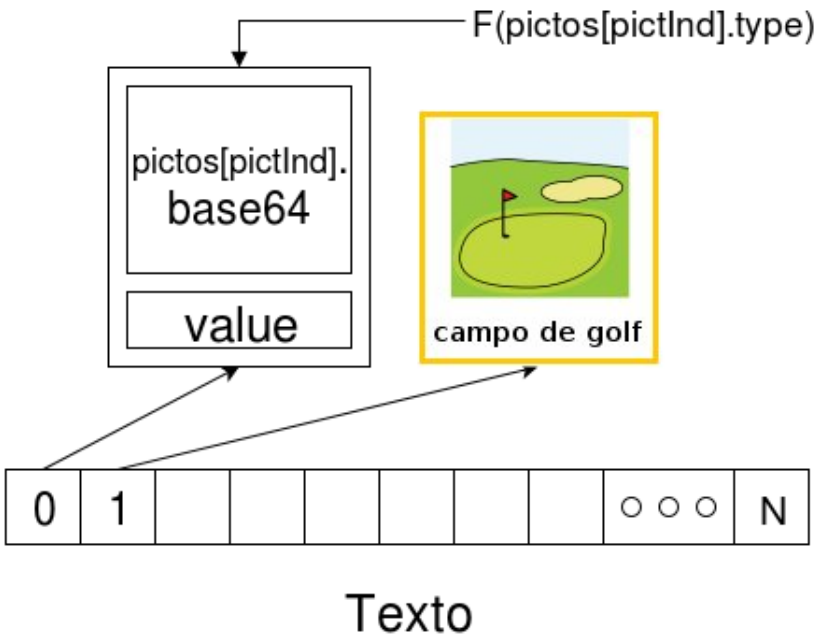

Figura 3.2.: Relación entre el modelo y la vista

Las representaciones de cada término se mostrarán de izquierda a derecha y de arriba abajo. El número de términos por linea variará en función del tamaño (configurable) de los pictogramas.

4. Implementación

La implementación se ha llevado a cabo en procesos cíclicos, siempre con prototipos o versiones funcionales, desde el primer prototipo vertical, el cuál ya era usable, hasta la última versión.

4.1. Vista principal (MVC)

Para comprender de manera más completa el modelo vista controlador que implementa AngularJS se va a explicar cómo se ha implementado la representación del picto-texto. Los módulos que intervienen son los siguientes.

Función	**Nombre**	**Fichero**
Vista	text	text.html
Controlador	textController	text.controller.js
Servicio	textAnalyzer	text.service.js

Cuadro 4.1.: MVC para la vista principal

El servicio es quien tiene el array de términos que conforma el picto-texto y las funciones para tratar con el mismo. Exporta el siguiente objeto JSON.

```
var service = {
    processEvent: processEvent,     /* Función para procesar los cambios */
    deleteWord: deleteWord,         /* Función para borrar un término*/
    setCaret: setCaret,             /* Función para modificar el cursor */
    addEmptyWord: addEmptyWord,     /* Función para añadir un término vacío */
    text: text,                     /* El array de términos */
    errors: errors,
    radius: radius,                 /* Radio de análisis para el algoritmo */
    docName: '',                    /* Nombre del documento */
    tryCompound: tryCompound,       /* Función para crear compuesto */
    splitWord: splitWord            /* Función para separar compuesto */
};
```

El controlador tiene el texto por defecto con el término Araword y lo pone en el servicio; a partir de ese momento el servicio y el controlador tienen el mismo objeto como texto y trabajan sobre lo mismo.

```
vm.myText = [{
    'id': getId(),
    'value': 'AraWord',
    'pictos': [{'picto':'25748.png', 'type':4}],
    'pictInd': 0,
    'words': 1,
    'autofocus': true
}];

textAnalyzer.text = vm.myText;
```

La vista tiene la representación HTML del texto que hay en el controlador. Si analizamos la vista de fuera hacia adentro, es decir, del contenedor más grande al más pequeño, observamos lo siguiente:

```
<div class="visible" id="text" style="display: inline-block;">
```

Contiene todo el documento. El atributo "style" especifica cómo se van a situar los elementos en el interior de este contenedor.

```
<div class="inline padding"
       ng-repeat="word in text.myText track by word.id"
       ng-attr-id="{{word.id}}">
```

Esta seccion de código corresponde al contenedor de cada término. Tiene el atributo ng-repeat por lo que se repetirá para cada término de la lista. De tal manera que cualquier referencia posterior a "word" es una referencia a un término.

```
<img class="picto"
           data-ng-src="{{  word.pictos[word.pictInd]['base64']
           || text.readPicto(word.pictos[word.pictInd]) }}"
           hm-tap="text.singleClickAction(word)"
           hm-press="text.showOptions(word)"
           hm-swiperight="text.swipe(\$event, word)"
           hm-swipedown="text.otherSwipe(word)"/>
```

El código anterior representa una imagen con el pictograma asociado al término. El atributo "data-ng-src" contiene la imagen en base64. En el caso de que aún no se haya leído la imagen ésta se lee y se asigna. Después se asocian una serie de funciones a ciertos eventos: "hm-tap" para la pulsación simple, "hm-press" para mantener pulsado, "hm-swipe" para los desplazamientos a izquierda y derecha.

```
<input type="text"
           ng-change="text.onChange(word)"
           ng-keyup="text.onKeyUp(\$event,word)"
```

```
ng-model="word.value"
ng-model-options="
    { 'updateOn': 'default blur', 'debounce':
        { 'blur':50,
        'default':5000 }
    }"
focus="{{ word.autofocus }}"
pu-elastic-input
class="word"
single
autocapitalize="off"/>
```

Por último, cada término está escrito en un "input". Al igual que con la imagen, ciertos eventos disparan algunas funciones: "ng-change" para los cambios y "ng-keyup" para cada pulsación. Por otro lado, el contenido se enlaza con la etiqueta "ng-model".

4.2. Algoritmo principal

Se ha denominado algoritmo principal al encargado de evaluar los cambios en el texto, generar los posibles términos compuestos, buscarlos en la base de datos e insertarlos en el texto en caso de que sea necesario.

Se han realizado 3 revisiones del algoritmo, desde la rev. A utilizada en el primer prototipo vertical que sólo utilizaba la base de datos de términos hasta la rev. D que usa las bases de datos de términos y conjugaciones. El uso y estructura de las bases de datos está especificado en el apéndice A.3.2. Los pasos que sigue dicho algoritmo en su última versión son los siguientes:

1. Crear un contexto que incluya los términos no compuestos, ni vacíos y que estén en un radio preestablecido (parámetro del programa) con respecto al término donde el cambio se ha producido.

2. Generar todos los posibles términos compuestos en el contexto y ordenarlos de mayor a menor por número de palabras.

3. Para cada uno de los términos generados.

   a) Buscar en la base de datos de verbos el término; si está, añadirlo a los resultados, en caso de que no haya ningún otro término de mayor longitud que incluya las palabras del mismo.

   b) Buscar en la base de datos de términos; si está, añadir los pictogramas al verbo, en caso de que el paso anterior haya obtenido resultados o añadirlo a los resultados.

4. Para cada resultado.

a) Si el resultado tiene una posición superior a la longitud del texto (el texto ha variado y ha menguado en tamaño, generalmente por la inclusión de un compuesto) se añade al final el nuevo término.

b) Si el resultado es igual al elemento del texto con la misma posición no se modifica.

c) Si el resultado es compuesto se eliminan los términos simples y se introduce el compuesto en su lugar.

d) Si el resultado es simple y la posición está dentro del contexto se modifica el término que ya existe en el texto.

e) Si el resultado es simple y la posición está fuera del contexto se añade el resultado al final del contexto.

En las revisiones A, B y C el algoritmo principal se ejecutaba cada vez que sucedía un cambio en un término (con un *debounce* de 600ms, es decir, cuando sucede un cambio y pasan 600ms sin producirse otro) y cuando el cursor abandonaba dicho término. Esto suponía serios problemas debido a la asincronía de los resultados, ya que los términos de un cambio anterior pueden llegar posteriormente. Además, también existe otro problema con el tiempo de *debounce*, ya que no todas las personas tardan en escribir lo mismo. Así pues se decidió que el algoritmo principal solo se ejecutase cuando el cursor abandonase el término. De esta manera se evitan los problemas de asincronía y de *debounce*.

4.3. Integración con redes sociales

Para conseguir enviar picto-texto a través de las redes sociales se ha usado la librería *html2canvas*, que permite obtener una imagen de un elemento HTML. De esta manera se puede capturar todo el documento y compartirlo como una única imagen a través de cualquier red social.

El principal problema es que los navegadores optimizan el tiempo de carga de las páginas *renderizando* sólo los elementos que aparecen en la pantalla. Eso supone que si el texto ocupa más que la pantalla en la imagen no aparecerá todo el texto, únicamente la parte *renderizada*.

Se ha solucionado utilizando el atributo *overflow* de CSS para forzar el *renderizado* completo así como un servicio de Ionic que permite hacer *scroll* para solventar el hecho de que el atributo CSS solo hace visible el contenido oculto por debajo de la pantalla, no el que se esconde por encima.

4.4. Envío de documentos

Para mantener la compatibilidad con Araword PC se ha implementado un servicio denominado *parser* que se encarga de traducir los documentos de su forma nativa en JSON al formato común en *XML*, y viceversa.

Para enviar un documento se utiliza el servicio de *parsing* con el objetivo de generar un fichero AWZ y luego se envía a través de cualquiera de los servicios de mensajería presentes en el dispositivo. Además se ha asociado el formato AWZ con la aplicación AWm para permitir abrir los documentos directamente desde el navegador de archivos.

4.5. Documentación

La documentación de usuario se ha realizado en colaboración con los profesores del CPEE Alborada ya que son ellos los que conocen a los usuarios y mejor pueden generar dicha documentación.

La documentación técnica se ha creado utilizando *JSDoc*. En concreto, un plugin para AngularJS llamado angular-jsdoc, lo que ha permitido generar una documentación navegable en formato HTML que se ofrece junto a la propia aplicación.

5. Pruebas

5.1. Con experto

Contar con un experto desde las fases tempranas del diseño ha permitido no introducir casi ningún cambio en la interfaz desde el primer prototipo en papel hasta el producto final.

Se han realizado reuniones mensuales a lo largo de los ocho meses que ha durado el proyecto, en las que se ha evaluado, por un lado, la usabilidad de la aplicación, y por otro, el funcionamiento de la misma.

En estas pruebas se utilizan los conocimientos del experto para poder determinar qué funcionalidades no triviales requiere el usuario. De esta manera se han determinado requisitos no funcionales como la implementación de un control de accesos o la limitación de las funcionalidades por parte del administrador.

5.2. Usuarios

Una vez desarrollada la primera versión se entregó a un grupo de una decena de personas que incluían logopedas y personas con concimientos técnicos en el ámbito de los productos de ayuda a la comunicación. Una vez distribuida se recabó información enviada por cada uno de los *testers* y se aplicaron las correcciones pertinentes.

Cuando la aplicación estuvo avanzada, se distribuyó a un grupo mucho más amplio, de aproximadamente un centenar de personas, todas ellas con experiencia en el cuidado de gente con discapacidad, logopedas o educadores. Al igual que en el proceso anterior, se obtuvieron múltiples informes. Los informes y las decisiones tomadas a partir de ellos se encuentran en el apéndice A.7.

6. Cierre del proyecto

6.1. Memoria

La memoria se ha creado utilizando LaTeXa través de la plataforma ShareLatex, lo que ha facilitado el aprendizaje del lenguaje TeX. Se ha generado por separado la memoria de los apéndices.

6.2. Trabajo pendiente

Se han encontrado diversos obstáculos que no han podido ser superados ya sea por la dificultad de los mismos o por la falta de tiempo. Se listan a continuación las tareas pendientes.

- Se sugirió la posibilidad de que en caso de escribir un término sin ningún pictograma asociado se generase automáticamente una imagen con el propio término. Se ha intentado usar la misma librería que para enviar los documentos a través de las redes sociales pero no se ha conseguido que funcione.
- Realizar la compilación para otras plataformas como iOS o Windows Phone. No se ha hecho porque de momento no existe la posibilidad de distribuir la aplicación para dichas plataformas en su correspondiente *market*.
- Añadir un método de recuperación de contraseñas que permita recordar la contraseña de administrador.

6.3. Conclusiones

Trabajar con una nueva tecnología supone un hándicap que no se abandona cuando aprendes la tecnología, ya que es en ese momento cuando te das cuenta de las cosas que no has hecho correctamente. Reparar los errores conceptuales que se han acarreado durante el desarrollo debido al desconocimento es muy costoso en tiempo y además no se plasma en avances en la aplicación.

El prototipado cuando se usan tecnologías híbridas como Ionic es extremadamente rápido, pudiendo de esta manera no utilizar herramientas de prototipado. El principal problema, sobre todo cuando tienes un cliente, es no dar la sensación de tener un producto terminado en una fase temprana. Si consigues mantener las expectativas del cliente

dentro de unos límites, tal y como se ha hecho en este proyecto se puede ahorrar mucho tiempo.

La asincronía en JavaScript es un tema realmente complicado y en aplicaciones que requieren un análisis pesado en tiempo real generan diversos problemas con soluciones no triviales. Si las tareas son pesadas en cómputo se tiene que mantener la asincronía para no generar una degradación de la experiencia de usuario, en cualquier otro caso es mejor utilizar funciones síncronas.

Parte II.

PictoServer

7. Análisis del problema y diseño de la solución

7.1. Contexto

Hasta ahora la manera de actualizar el conjunto de pictogramas que utiliza Araword era mediante un fichero ZIP estático, alojado en un servidor web. Dicho fichero era generado de manera manual y sustituido completamente con cada actualización. Esto implica que en Araword tras una actualización toda la información local de los pictogramas desaparece.

Con la creación de la aplicación móvil se llegó a la conclusión de que descargar todo el conjunto de pictogramas para cada actualización, desde un dispositivo móvil, sería demasiado costoso. Así pues se decide diseñar un servicio web que actualice tanto la aplicación móvil como la de sobremesa.

El servicio de actualización ofrecerá la posibilidad de descargar sólo los pictogramas que falten en el dispositivo. Además, se podrán añadir pictogramas e idiomas al servidor a través de un formulario (previa autenticación).

7.2. Análisis tecnológico

Se ha decidido trabajar con el *stack* MEAN que incluye MongoDB, Express, AngularJS y Node.js. Se ha elegido esta tecnología porque se tiene experiencia con ella y se sabe que el desarrollo es poco costoso en tiempo. Una breve descripción del *stack* es la siguiente:

- MongoDB: Es un sistema gestor de bases de datos NoSql orientado a documentos.
- Express: Es una infraestructura de aplicaciones web Node.js mínima y flexible que proporciona un conjunto sólido de características para las aplicaciones web y móviles. [3]
- AngularJS: Es una plataforma para el desarrollo de aplicaciones web basada en JavaScript que implementa el modelo vista controlador.
- Node.js: Es un entorno de ejecución para JavaScript orientado al desarrollo de aplicaciones del lado del servidor.

7.3. Diseño del servicio

El servicio tiene que cubrir dos aspectos básicos. Por un lado, actualizar el conjunto de pictogramas y actualizar los idiomas soportados por la aplicación, con las nuevas bases de datos de verbos si fuese necesario. Por otro lado, permitir actualizar el conjunto de pictogramas que provee el propio servicio.

Se han definido dos recursos *pictos* y *languages*. Sobre el recurso *pictos* se ha definido una operacion download que permite descargar el conjunto de pictogramas añadidos desde una fecha. La lista completa de endpoints se pueden encontrar en el apéndice B.1.

7.3.1. Pictograma

El pictograma es el recurso básico, contiene la información de todas las palabras asociadas a un único pictograma así como su última fecha de actualización. Esto permite al servidor filtrar los resultados en función de la fecha que acompañe la petición.

```
{
    "_id" : ObjectId,           /* Id interno */
    "path" : String,            /* Nombre del picto ej: 2001.png */
    "date" : ISODate,           /* Fecha de última modificación */
    "word" : [{
        "type" : String,        /* Tipo de palabra ej: "verbo" */
        "language" : String,    /* Idioma ej: "Castellano" */
        "value" : String        /* El término en si */
    }]                          /* Las palabras asociadas al picto */
}
```

7.3.2. Language

Al recurso *language* no se accede de manera individual sino que se solicita la lista de todos los idiomas. Cada idioma tiene la información necesaria para que la aplicación funcione correctamente. El nombre largo es necesario para la versión de PC de Araword mientras que el *locale* es necesario para usar el *text-to-speech* de la aplicación móvil.

```
{
    "_id" : ObjectId,       /* Id interno */
    "code" : String,        /* Código de idioma ej: "es" */
    "locale" : String,      /* Locale ej: "es-ES" */
    "long" : String,        /* Nombre del idioma ej: "Castellano" */
    "haveVerbs": Boolean    /* Cierto si hay verbos conjugados */
}
```

Para acceder a un recurso *language* antes se consulta la carpeta donde están todas las bases de datos de verbos conjugados, en caso de que exista la base de datos el atributo *haveVerbs* se pone a verdadero. Una base de datos de verbos conjugados contiene todas las conjugaciones de los distintos verbos en un idioma determinado

8. Implementación

La utilización del *stack* MEAN para desarrollar servicios web está bastante estandarizada y no da lugar a la toma de muchas decisiones durante la implementación. Algunos de los principales problemas que han surgido y sus soluciones se explican a continuación.

8.1. Desbordamientos de pila en traducciones

Uno de los aspectos más importantes de pictoServer es que trabaja con XML y objetos JSON de gran tamaño, ya que al iniciar el proyecto se contaba con 9306 pictogramas y unos 139.000 términos, esto ha dado lugar a problemas de desbordamiento de la pila.

Lo que se ha intentado es realizar traducciones parciales e ir escribiéndolas poco a poco en disco, esto hace que el contenido nunca esté por completo en la RAM y evita los desbordamientos.

8.2. Ficheros ZIP en memoria

Generar dinámicamente un fichero ZIP con 9306 imágenes es un proceso costoso y aquellas librerías que crean el fichero completo primero en RAM y luego lo vuelcan a disco también dan lugar a desbordamientos de la pila.

Para solucionar este problema lo que se ha utilizado el comando *zip* del sistema GNU/Linux que funciona por debajo ya que el tiempo de respuesta es muchísimo mas pequeño y no genera ningún desbordamiento.

Además con el objetivo de disminuir el tiempo de creación de los ficheros ZIP se ha decidido mantener siempre un fichero *latest.zip* que contenga la última versión completa de los pictogramas, de tal manera que para todo cliente que visite el *endpoint* sin ninguna fecha en el parámetro *date* será como si pidiesen un archivo estático.

Utilizar el comando zip en vez de un módulo en JavaScript nos permite utilizar opciones para añadir contenido al fichero en vez de generarlo de nuevo, lo cuál también reduce el tiempo de compresión cada vez que se añaden nuevos pictogramas al servidor.

8.3. Actualización del conjunto de pictogramas

Para actualizar los pictogramas del servidor se ha creado un *frontend* que se ha alojado en el mismo servidor. Se dispone de un formulario con dos entradas, una para el fichero ZIP que contenga únicamente los pictogramas y otro para un fichero XSLX con la

información de los pictogramas. La descripción detallada de cómo debe ser el XSLX está en el apéndice B.2.

9. Pruebas

9.1. Pruebas de desarrollo

Durante el desarrollo de los *endpoints* se han ido realizando múltiples pruebas. Para ello se ha utilizado la herramienta *Postman*, un complemento de Google Chrome, que nos permite realizar peticiones GET, POST, PUT y DELETE.

- Se han realizado **pruebas de carga** para determinar qué cantidad de trabajo puede suponer una caída del sistema o una degradación de las prestaciones.
- Pruebas de **caja negra** para comprobar que ningún valor de los parámetros en los *endpoints* hace caer el servidor.

Durante la fase de pruebas se observó que el proceso crítico es el de creación del nuevo fichero *latest.zip* que contiene la última versión de los pictogramas. Tras la optimización descrita en la Sección 8.2 se ha reducido el uso de memoria del servidor, evitando así la caída del mismo durante el proceso.

9.2. Pruebas con usuarios

Aunque en la primera fase de pruebas los pictogramas de Araword se incluían en el propio APK, a partir de la versión 0.0.4 se integró el servidor de pictogramas, por lo que todas las pruebas con usuarios de Araword se han sustentado en el conjunto de pictogramas de pictoServer.

Además, se solicitó a los usuarios que instalasen la aplicación al mismo tiempo, para conseguir el máximo número de descargas simultaneas. Ninguno de los informes ha reportado algún problema con el servidor, por lo que podemos considerar que responde correctamente.

Por otro lado, desde la versión 0.1.13, se ha añadido la funcionalidad de actualizar pictogramas que no solo utiliza el endpoint */pictos/download* sino que además consulta los idiomas soportados para determinar si se ha de descargar una nueva base de datos de verbos conjugados o no, y si se ha de añadir un idioma a la tabla *Languages*. A pesar del incremento de tráfico entre el servidor y las aplicaciones no se ha recibido ningún informe negativo.

10. Cierre del proyecto

10.1. Memoria

La memoria se ha creado utilizando LaTeXa través de la plataforma ShareLatex lo que ha facilitado el aprendizaje del lenguaje TeX. Se ha generado por separado la memoria de los apéndices.

10.2. Trabajo pendiente

En cuanto al trabajo a realizar, se han definido ciertas tareas que se han de terminar en un futuro, algunas de ellas por temas de seguridad y otras por temas de usabilidad:

- Modificar las cabeceras de CORS que envía el servidor para acomodar los permisos al uso habitual del servidor.
- Crear un conjunto de pruebas automáticas de los *endpoints* para evitar la comprobación manual tras cada modificación o parche del servidor.
- Modificar el *frontend* para permitir la creación de usuarios por parte de usuarios previamente autenticados.

10.3. Conclusiones

El desarrollo de servicios web utilizando el stack MEAN es rápido y cuando se tiene experiencia, poco costoso. A pesar de ello, en el momento en el que se tienen que realizar tareas no habituales, como puede ser la compresión de ficheros muy grandes o el tratamiento de objetos JSON muy grandes, buscar soluciones supone un coste en tiempo altísimo.

11. Gestión de proyecto

Se ha utilizado GitHub como repositorio de código, se ha mantenido de manera privada hasta tener una versión estable de la aplicación. En cuanto al repositorio documental se ha utilizado DropBox, lo cual ha garantizado la persistencia de las actas de las reuniones así como los diagramas, bocetos y el cuaderno de bitácora, donde quedan reflejadas todas las horas invertidas.

11.1. Distribución temporal

El tiempo del proyecto se ha repartido en las siguientes tareas:

- Estudio de Araword (8h).
- Aprendizaje de la tecnología (28h).
- Desarrollo del prototipo software (16h).
- Análisis, diseño e implementación de AWm (110h).
- Análisis, diseño e implementación de pictoServer (80h).
- Reuniones con el director y el logopeda (15h).
- Creación de la documentación técnica (19h).
- Escritura de la memoria (30h).

En total el tiempo dedicado al proyecto ha sido de 306 horas, concentradas en su mayoría en el desarrollo de AWm y pictoServer.

Bibliografía

[1] Arasaac. sistemas aumentativos y alternativos de comunicación. http://arasaac.org/aac.php. Accessed: 2016-08-2.

[2] Catedu. http://aratools.catedu.es/araword/#quees. Accessed: 2016-08-30.

[3] Expressjs. http://expressjs.com/es/. Accessed: 2016-08-6.

[4] Wikipedia. https://es.wikipedia.org/wiki/Modelo-vista-controlador. Accessed: 2016-08-30.