Universidad Zaragoza
1542

Proyecto Fin de Carrera

AraSuite: Integración de las aplicaciones de TICO y AraWord.

Autor

Adrián Gómez Llorente

Director

Joaquín Ezpeleta Mateo

Escuela de Ingeniería y Arquitectura (EINA)

2014

A mi tutor, Joaquín Ezpeleta, por darme la oportunidad de participar en un gran proyecto y ser paciente ante una dedicación que en algunos momentos fue dificil.

A mis padres, por su dedicación completa, su apoyo incondicional y por empujarme para llegar hasta aquí.

A mi familia, por rodearme con su apoyo y escuchar una y otra vez los entresijos de este proyecto.

A mis amigos y compañeros, porque en esta vida hace falta tener humor para coger fuerzas. En especial a Jorge Pinto, por imprimir su creatividad en el logotipo de AraSuite.

AraSuite: Integración de las aplicaciones de TICO y AraWord

RESUMEN

Este proyecto fin de carrera (PFC) se ha realizado con la colaboración de profesionales del Colegio Público de Educación Especial Alborada (C.P.E.E Alborada) y el Centro Aragonés de Tecnologías para la Educación (CATEDU).

En este PFC se ha realizado el desarrollo de la aplicación llamada AraSuite. Esta aplicación es un conjunto de herramientas que trabajan de forma conjunta para hacer más fácil el trabajo diario con personas que tienen graves trastornos en la expresión oral, de forma que su día a día se vea mejorado.

El presente proyecto surge como solución al problema existente en las aplicaciones TICO y AraWord en las que la información es gestionada de manera independiente por cada una de ellas, generando datos duplicados y un entorno de trabajo que no es efectivo, haciendo que ambas aplicaciones sean difíciles de mantener.

AraSuite parte de la situación actual y crea un entorno que centraliza toda la gestión de la información, ofrece métodos de acceso a los datos y evoluciona la situación actual hacia una suite de aplicaciones que se comportan como una única herramienta facilitando el trabajo diario de profesores y tutores.

Con la creación de AraSuite se agrupa el desarrollo de las aplicaciones existentes TICO y AraWord bajo un mismo código. Además, se define una arquitectura y unos flujos de desarrollo que facilitan la participación de otros desarrolladores para añadir nuevas funcionalidades, de esta manera, se pretende convertir a AraSuite en un referente entre las aplicaciones opensource que facilitan la interacción con personas con dificultades para la expresión oral.

El resultado final obtenido es una aplicación de código libre que actualmente, con casi 2500 descargas mensuales, es usada en el mundo entero por miles de personas. Los usuarios, valoran muy positivamente el uso de la herramienta destacando su facilidad de uso y el gran aporte que hace en el trabajo diario. Además, gracias a la forma de trabajo definida y a la arquitectura aplicada, se ha agilizado la entrega de nuevas versiones de AraSuite que hacen que esta aplicación esté en continua evolución.

Índice

Página

Índice de diagramas

1. Introducción

1.1. Idea general

AraSuite es un conjunto de herramientas desarrolladas con la colaboración de profesionales del Colegio Público de Educación Especial Alborada (C.P.E.E Alborada) y el Centro Aragonés de Tecnologías para la Educación (CATEDU) y tiene como objetivo ofrecer una única suite de aplicaciones que trabajen de forma conjunta para facilitar el trabajo diario con personas con graves trastornos en la expresión oral, de forma que su autonomía y su relación con el entorno se vean mejoradas.

El principal problema de las aplicaciones existentes TICO y AraWord es que la información de pictogramas es gestionada de manera independiente por cada una de ellas, generando duplicidades que no son efectivas y dificultan la mantenibilidad de cada una de las aplicaciones. AraSuite, evoluciona la situación actual y crea un entorno que integra toda la gestión de pictogramas, centraliza los métodos de acceso a la información, y elimina las duplicidades existentes. Además, este entorno, es fácilmente extensible y sirve como base para la inclusión de futuras aplicaciones dentro en la suite.

Además también se pretende organizar el proyecto definiendo flujos de desarrollo que faciliten la participación de otros desarrolladores a la hora de incorporar nuevas funcionalidades, llevando así a AraSuite a ser un referente entre las aplicaciones opensource que facilitan la interacción con personas con dificultades para la expresión oral.

1.1.1. Las herramientas TICO y AraWord

TICO es una aplicación que comenzó su desarrollo en el año 2005 y a través de distintos proyectos fin de carrera, se ha hecho una herramienta que permite a los profesores trasladar el concepto de tablero de comunicación en modo impreso al formato informático para aprovechar las posibilidades que este nuevo entorno ofrece. Para ello, TICO dispone de dos componentes diferenciados que trabajan conjuntamente para obtener el resultado esperado.

Por un lado está el editor que ofrece las herramientas necesarias para diseñar el tablero, permitiendo la inserción de pictogramas y formas, asociación de ac-

ciones, definición de flujos de movimiento y selección, etc. Y por otro lado está el interprete, que ofrece un entorno donde ejecutar y visualizar los tableros de comunicación desarrollados previamente en el editor y cuya principal característica es la herramienta de barrido que permite que el cursor vaya desplazándose de forma automática por todos los elementos con acción asignada para facilitar la interacción con el tablero a las personas con limitaciones motrices.

AraWord nació como un proyecto desarrollado por Joaquín Pérez Marco, su principal objetivo es ofrecer una herramienta que permita la generación de documentos de pictogramas a través de la inserción de texto y posterior conversión automática del texto introducido en pictogramas.

1.2. Objetivos

El objetivo principal de AraSuite es la adaptación de las aplicaciones de TICO y AraWord a un nuevo entorno en el que ambas compartan toda la información relativa a los pictogramas y sus términos asociados, de manera que las operaciones de inserción, modificación o borrado de pictogramas o términos, sean generales para todas las ellas.

Además, la solución desarrollada debe ofrecer una capa de acceso a los datos que garantice una única puerta de entrada y una versatilidad suficiente para que otras aplicaciones puedan ser integradas dentro de la misma suite.

Estas modificaciones deben tener como requisito mantener las exigencias en nivel de velocidad de acceso a los datos de las aplicaciones y satisfacer las necesidades que se hayan detectado en cada uno de los casos.

La solución final debe tener una organización y unos flujos de desarrollo definidos que permitan el trabajo en una comunidad opensource, inicialmente bajo Sourceforge, pero fácilmente adaptable.

Por otro lado también será necesario arreglar diferentes problemas que se pudieran detectar en el transcurso del desarrollo del proyecto en cualquiera de las dos aplicaciones, tanto TICO como AraWord con intención de hacerlas más estables u ofrecer nuevas funcionalidades que se consideren necesarias.

1.3. Estructura del documento

La memoria del PFC se divide en los siguientes capítulos:

I **Introducción:** Explicación breve de la idea general y los objetivos de AraSuite.

II **Análisis:** Estado del arte a la hora de comenzar el PFC y requisitos y objetivos del desarrollo de AraSuite.

III **Diseño:** Descripción técnica de la aplicación realizada para cumplir los objetivos, requisitos y necesidades identificados durante el análisis.

IV **Desarrollo:** Explicación detallada de todo el proceso de desarrollo de la aplicación AraSuite, metodología aplicada, pruebas de calidad realizadas y ejemplos de algunos resultados obtenidos.

V **Gestión del proyecto:** Resumen de la metodología aplicada y desviaciones detectadas.

VI **Conclusiones y trabajo futuro:** Comentario de los resultados obtenidos, opinión personal y posibles maneras de continuar el desarrollo futuro.

VII **Bibliografía:** Referencias de documentación online, artículos y libros utilizados.

VIII **Anexos:** Información adicional detallada que, por brevedad, no han podido formar parte de la memoria principal.

2. Análisis

En esta fase se recoge el estado del arte de las aplicaciones y asienta las bases sobre las que se continuará el desarrollo de AraSuite.

2.1. Terminología

A lo largo del proyecto se usará una serie de términos técnicos que se explican a continuación:

Pictograma: Es un dibujo de fácil entendimiento que se usa para describir un objeto, acción u otra cosa.

Término: Es una palabra que se asocia a un pictograma y que facilita su búsqueda o su uso en las distintas aplicaciones.

GalleryManager: Es una aplicación que permite realizar distintas operaciones tipo CRUD (En inglés: Crear, Leer, Actualizar y Eliminar) con los pictogramas y sus términos asociados y sirve como nexo de unión de toda la información para las distintas aplicaciones que lo usan, además también incluye una serie de interfaces que permiten al usuario trabajar con los pictogramas.

2.2. Requisitos

Durante el análisis se han recogido muchos requisitos, como puede verse en el anexo A, tanto de las aplicaciones existentes, TICO y AraWord, como requisitos que se tendrán que cumplir a la finalización de la nueva aplicación AraSuite.

Los requisitos de este proyecto podrían englobarse en tres grupos. El primer grupo estaría formado por requisitos extraídos del uso de la aplicación por parte de padres, profesores o tutores, englobados como usuarios, tales como gestionar los términos, buscar con filtros, importar, exportar y otros. Los usuarios, especialmente en la aplicación de TICO, trabajan directamente administrando los pictogramas y términos asociados, añadiendo nuevas imágenes, modificando las existentes, etc.

El segundo grupo de requisitos estaría formado por aquellos que obligan a

este proyecto a ofrecer una interfaz de métodos que permita a las aplicaciones gestionar los pictogramas y términos de una manera interna. Esto será especialmente útil para AraWord, ya que es una aplicación que necesita realizar búsquedas en los pictogramas para convertir un término, o varios, en un pictograma mientras que el usuario escribe.

El tercer grupo de requisitos lo formarían aquellos que añaden nuevas funcionalidades o son más técnicos, como la actualización automática o mantener la velocidad de acceso a la información.

2.3. Casos de uso

Tras un análisis de los requisitos del proyecto, de recoger impresiones del director del proyecto, y analizar las necesidades que tenían los usuarios que habían estado usando TICO y AraWord, se han identificado varios casos de uso derivados de las operaciones con los pictogramas que AraSuite debe ofrecer al usuario y a otras aplicaciones. En el anexo B se puede ver un detalle de todos ellos.

En la figura 2.1 se puede ver el diagrama de casos de uso general. En él se han identificado cuatro actores, Usuario, que representa a las personas que van a usar la aplicación, por ejemplo un profesor, el usuario Aplicación, que representa a las aplicaciones que necesitan acceder a la gestión de pictogramas, y TICO y AraWord que representan a las aplicaciones ya existentes y que hacen uso de los pictogramas.

Usuario puede realizar todas las operaciones relacionadas con la gestión de pictogramas, pero para realizar cualquiera de ellas, es obligatorio que la aplicación lance la Interfaz de administración del GalleryManager".

Por otro lado, podemos ver que los actores TICO y AraWord tienen como casos de uso, "Lanzar la interfaz de administración del GalleryManager", y además, tiene un caso de uso que podríamos definir como una operación interna que permite realizar búsquedas sobre la base de datos.

2.4. Interfaz de acceso a los datos

Para analizar cuál es la mejor opción a la hora de diseñar el interfaz de acceso a datos del GalleryManager que posteriormente utilizarían TICO, AraWord u otras futuras herramientas, se realizó un estudio de los distintos métodos que se ejecutaban sobre la base de datos de pictogramas por parte de ambas aplicaciones para identificar aquellas necesidades que pudieran tener.

Como podemos ver en el anexo E, TICO y AraWord tienen variados métodos de acceso a la base de datos (BD). Es por esta razón por la que se ha decidido que el GalleryManager se comporte, *grosso modo*, como un DAO (Data Access Object).

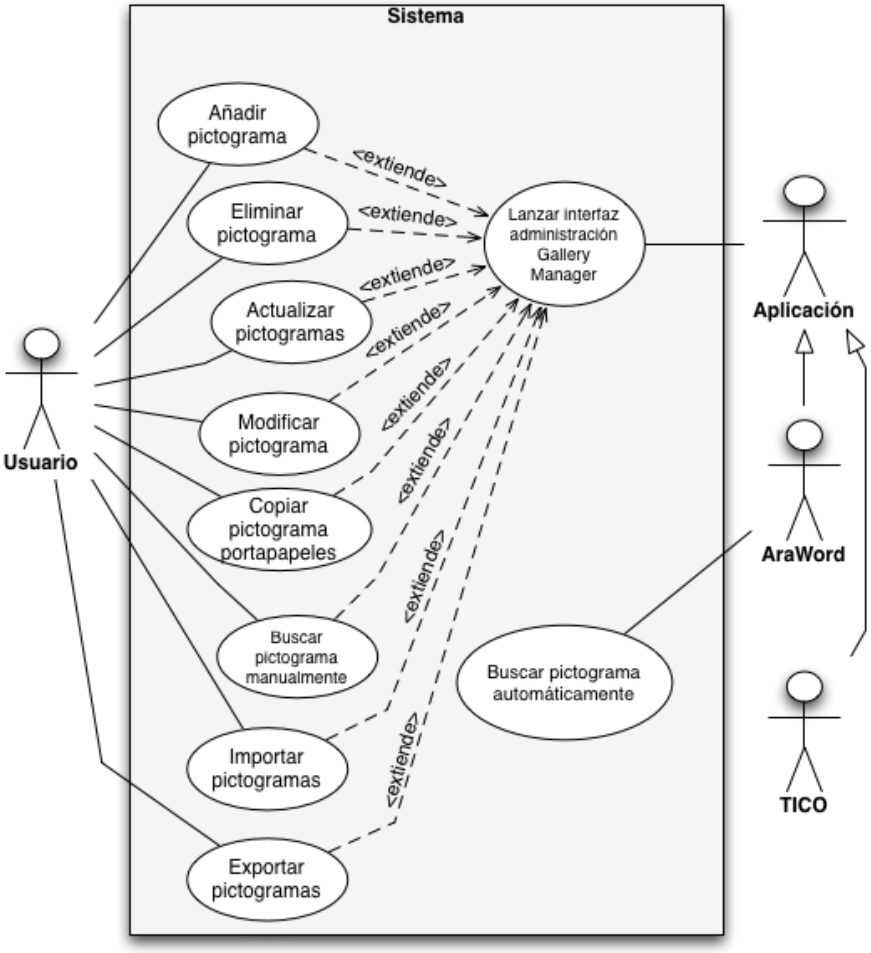

Figura 2.1: Casos de uso general

De esta manera, el GalleryManager publicará métodos generales de acceso a la BD, abstrayendo a las aplicaciones que usen la librería de otras cosas como la conexión con la BD, drivers de acceso, etc.

2.5. Especificación del plan de pruebas

La especificación de un plan de pruebas permite garantizar al desarrollador un cierto nivel de calidad de la aplicación y asegura que en el futuro el desarrollo contendrá menos errores debido a las baterías de test.

Por las características de este proyecto se ha decidido que el plan de pruebas a realizar será el siguiente:

- **Pruebas unitarias:** Son las pruebas formales que garantizan que un método, dados unos parámetros de entrada, tiene el comportamiento esperado.
- **Pruebas de sistema:** Son aquellas pruebas que garantizan el correcto funcionamiento de los puntos críticos identificados en el anexo F.2.
- **Pruebas de aceptación del usuario:** Son las pruebas que se realizan en un entorno de funcionamiento controlado que garantizan que el software desarrollado es funcional en condiciones normales.

En el anexo F se puede ver una explicación ampliada de las pruebas a realizar y el momento de hacerlas.

3. Diseño

En esta sección de la memoria se va a exponer la propuesta planteada para resolver todas las necesidades y requisitos que se identificaron en el apartado anterior.

3.1. Especificación de casos de uso

Antes de realizar el diseño de las interfaces se han tomado los casos de uso identificados en el anexo B, a partir de los cuales se ha realizado un estudio de las acciones que tiene que desempeñar cada usuario para hacer cada una de las operaciones.

Para ello, se ha realizado una especificación de los casos de uso mediante diagramas de actividad que servirán para el posterior diseño de las interfaces finales que se realiza en la sección 3.2.

A modo de ejemplos, a continuación se detallan dos casos de uso. Consúltese el Anexo D para un desarrollo completo.

Caso de uso: Copiar pictograma a portapapeles

- **Definición:** El usuario realizará una búsqueda para encontrar el pictograma que desea copiar al portapapeles, posteriormente tendrá que pulsar el botón de copiar y la imagen se enviará al portapapeles desde donde podrá ser usada por otra aplicación. Véase la figura 3.1.
- **Precondición:** La versión de Java instalada en el sistema operativo debe soportar el copiado al portapapeles.
- **Postcondición:** La imagen estará disponible en el portapapeles.
- **Interfaz que lo implementa:** Interfaz de edición de un pictograma ya existente.

Caso de uso: Actualizar pictogramas

- **Definición:** El usuario pulsará el botón de comprobar actualizaciones. En caso de no haber en el servidor de pictogramas ninguna versión posterior a

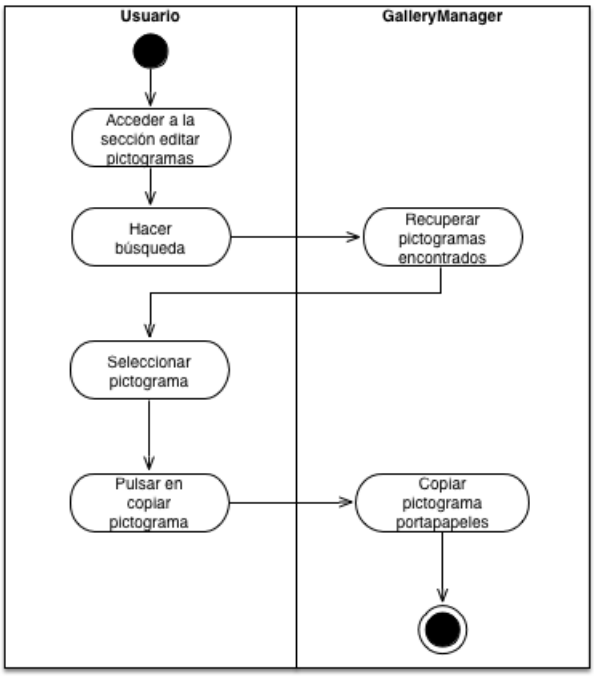

Figura 3.1: Diagrama de actividad Copiar pictograma portapapeles

la instalada en el ordenador del usuario, se mostrará un diálogo informando de que se tiene la última versión. En caso de que haya un nuevo paquete de pictogramas se mostrará información del mismo. Si el usuario decide actualizar se descargará el paquete, se descomprimirá y se importarán los pictogramas mostrando la interfaz de la figura C.6. Los detalles se muestran en el diagrama de la figura 3.2.

- **Precondición:** El ordenador debe tener acceso a Internet.
- **Postcondición:** Si hay una nueva versión de pictogramas se descargarán, importarán y sustituirán a los que tengan el mismo nombre. En caso contrario no se producirán cambios.
- **Interfaz que lo implementa:** Interfaz Actualizar pictogramas.

3.2. Diseño de interfaces

Una de las funcionalidades que tiene que ofrecer AraSuite es una serie de interfaces de gestión que posteriormente se integrarán en las distintas aplicaciones, como TICO y AraWord, y permitirán al usuario acceder a las funcionalidades de los pictogramas.

TICO y AraWord son proyectos que ya contienen, por duplicado, distintas

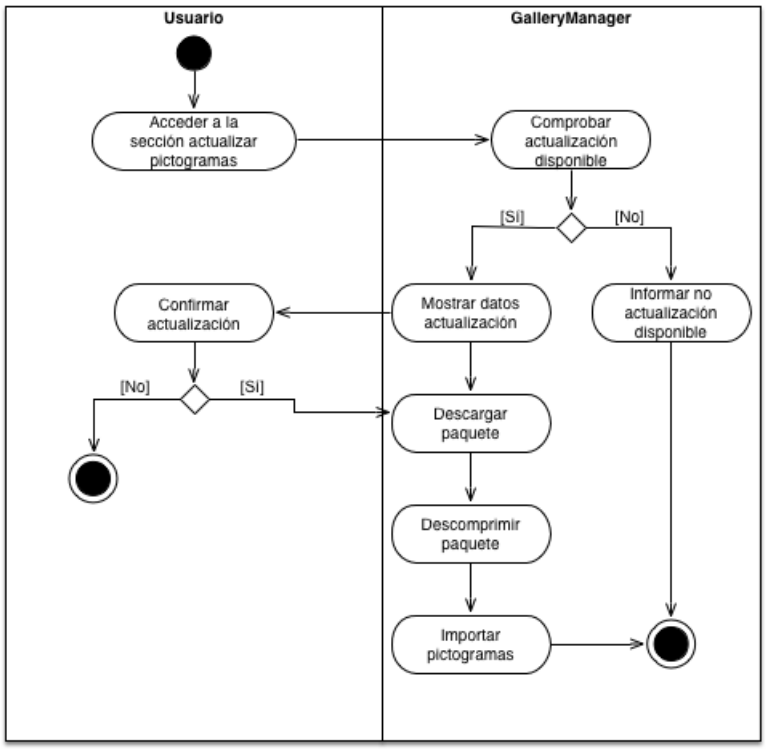

Figura 3.2: Diagrama de actividad Importar pictogramas

interfaces que permiten realizar operaciones de gestión sobre los pictogramas. Además estas herramientas ya están en producción y son muchos los usuarios que las utilizan, por lo tanto, se ha decidido que las nuevas interfaces del GalleryManager serán similares a las ya existentes en TICO y AraWord, de esta manera se mantendrá la usabilidad actual.

TICO, AraWord y otras posibles aplicaciones futuras tendrán que lanzar las interfaces de gestión que ofrece el GalleryManager. Tras evaluar como sería la mejor forma de integrar estas interfaces en el resto de aplicaciones, se ha decidido que la mejor solución es que el GalleryManager tenga una interfaz principal como la que se puede ver en la imagen de la figura 3.3. Esta interfaz será el punto de acceso a partir de la cual el usuario podrá navegar por el resto de interfaces de gestión de pictogramas.

De esta manera se ha conseguido que en caso de añadir una nueva funcionalidad de gestión de pictogramas al GalleryManager, el cambio será transparente para el resto de aplicaciones ya que únicamente habrá que añadir la nueva opción a la interfaz principal. Y además, aislamos el GalleryManager del resto de aplicaciones y podemos reaprovechar esta organización para dotar al GalleryManager de un sistema de interfaces que le permitan trabajar como una aplicación independiente.

Figura 3.3: Prototipo de la interfaz principal

Se han realizado prototipos de todas las interfaces de la aplicación para cumplir las funcionalidades detectadas en el diagrama de casos de uso de la figura 2.1. Estos prototipos y una explicación detallada de cada uno de ellos puede encontrarse en el anexo H.

3.3. Diseño de la base de datos

Uno de los requisitos es que las velocidades de importación y acceso a datos fueran mejores que las existentes. Por lo tanto es necesaria una remodelación de la base de datos ya que hay operaciones de tipo join[1] que lastran las consultas a la base de datos.

Por esta razón, se ha decidido plantear un diseño de la base de datos que evitara operaciones de "join" de tablas, para reducir de esta manera el coste de operación en la base de datos y agilizar así las búsquedas.

Antes de pasar a explicar el diseño de la base de datos que se ha desarrollado conviene explicar algunas consideraciones previas que se han tomado:

- **Uso del inglés para la nomenclatura de las tablas y los atributos:** Así se evitan posibles problemas de codificación de caracteres.
- **Uso de enteros auto-incrementados como *primary keys*:** En aquellos casos en los que sea necesario establecer relaciones entre tablas, se usarán enteros auto-incrementados como *foreign keys* aunque el contenido de las propias tablas pueda contener una *primary key*. A pesar de que esto introduce una clave que pudiera resultar innecesaria por existir una *primary key* natural, se ha decidido que los beneficios prevalecen frente a las contraindicaciones. Las razones que han ayudado a tomar esta decisión son: que se permite que las primary key naturales cambien de valor,

[1] Operación que devuelve como resultado el producto cartesiano de dos o más tablas. Estas operaciones tienen un alto coste operacional.

el rendimiento es mayor dados los índices más pequeños y las relaciones se entienden mejor.

En el diagrama de la figura 3.4 se ve el diseño de la base de datos propuesto. Para más información sobre la BD se puede ver el anexo I.

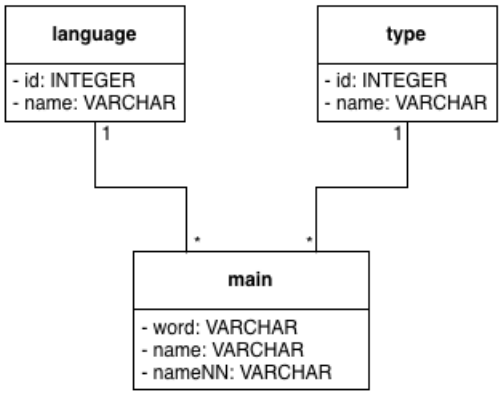

Figura 3.4: Diseño de la BD

3.4. Diseño de la API de acceso a datos

Durante el análisis, en el apartado 2.4 y en el anexo E se han mostrarlo cuales son las necesidades de acceso a datos de TICO y AraWord.

En función de las necesidades tan diferentes de ambas aplicaciones me he decantado por ofrecer una solución que fuera lo más generalista posible. Esta solución se ha basado en ofrecer los siguientes métodos:

query(): Ejecuta una consulta en la BD y devuelve el resultado en un cursor que se podrá recorrer para hacer uso de los datos obtenidos.

update(): Ejecuta una operación para modificar un valor existente en la BD.

prepareStatement(): Cachea una consulta en la BD para agilizar su posterior ejecución.

De esta manera el GalleryManager se comporta como un *Data Access Object* (DAO, Objeto de Acceso a Datos) suministrando una interfaz común de acceso a la BD a cualquier aplicación. En el diagrama de la figura 3.5 se muestra solución expuesta.

Con esta solución podemos garantizar que se mantiene el control sobre todas las operaciones realizadas sobre la base de datos. Además, se controla el número de conexiones abiertas contra la base de datos ya que el GalleryManager gestiona la creación y ciclo de vida de las conexiones con la DB. De esta manera evitamos

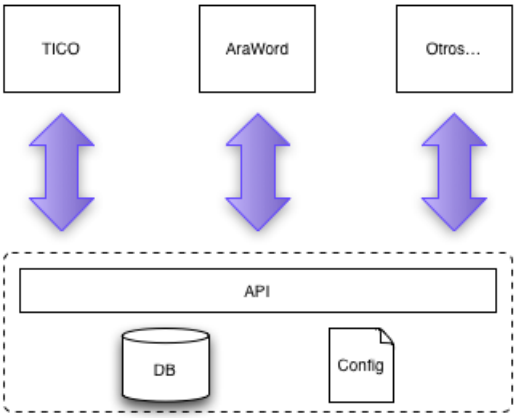

Figura 3.5: API de acceso a datos

que se abran múltiples conexiones y optimizamos los recursos, ya que abrir una conexión con la DB es una de las operaciones que mayor coste operacional tiene.

Además, como podemos ver en el diagrama de la figura 3.5, el API del GalleryManager también interactúa con un fichero de configuración. Este fichero contendrá todos parámetros de configuración de la BD, como el directorio donde se encuentra, la localización de los pictogramas, la última fecha de actualización de pictogramas y la URL de actualización automática. El fichero de configuración tendrá una estructura que contendrá en cada línea un par clave valor con la información necesaria.

3.5. Arquitectura de directorios

Otro de los requisitos del proyecto es que las aplicaciones de TICO y AraWord se comporten como una única suite compartiendo imágenes y base de datos.

Finalmente las aplicaciones se han organizado como se muestra en la imagen de la figura 3.6. De esta manera las aplicaciones están juntas bajo un mismo directorio y el usuario intuye la aplicación como si fuera una única suite y no varias aplicaciones sueltas.

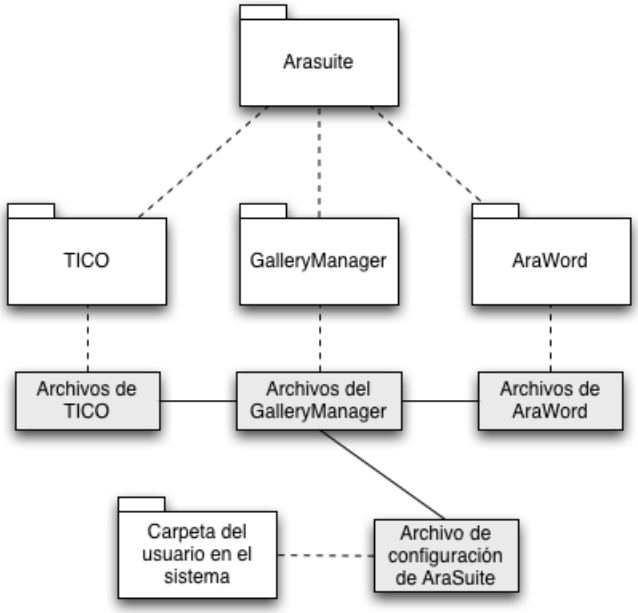

Figura 3.6: Arquitectura de directorios

4. Desarrollo

En esta sección se detallan algunas de las decisiones principales que se han tomado a lo largo del proceso de desarrollo. En este proyecto ya se partía de varios desarrollos anteriores y el objetivo era la adaptación de varias aplicaciones que ya estaban desarrolladas, por lo que algunas partes del proceso están muy condicionadas por este entorno.

4.1. Tecnologías empleadas

Java es un lenguaje de programación compilado a "Bytecode", que se ejecuta sobre una máquina virtual llamada JVM. En este caso, la decisión de usar Java formaba parte de los requisitos del proyecto, pero aún con todo, la decisión estaba totalmente justificada ya que Java es multiplataforma.

SQLite es un sistema de gestión de bases de datos SQL. En este caso, el uso de SQLite también viene impuesto por los requisitos del proyecto, pero aún con todo también se adapta a las necesidades del nuevo desarrollo. Entre las virtudes de SQLite se encuentran su tamaño reducido, con unos 2,5MB, la integración con Java y su alto rendimiento[1].

ANT es una librería Java que facilita la compilación de proyectos basados en Java. En este proyecto se ha usado la versión ANT 1.7.0. El uso de ANT también ha sido heredado de los proyectos anteriores, pero también se adaptaba perfectamente a los objetivos del nuevo desarrollo.

Subversion y Sourceforge son un sistema de control de versiones para proyectos de desarrollo y una plataforma que permite la distribución de proyectos de software libre y te da la opción de crear un repositorio SVN para ellos.

En el caso de este proyecto, el uso de la dupla Subversion y Sourceforge formaba parte de los requisitos del proyecto. Aún con todo se valoró la idea de usar otro sistema de control de versiones como GIT junto con la plataforma GitHub, ya que GIT tiene algunas virtudes como que es más eficiente a la hora de crear ramas, no introduce ficheros adicionales junto con los fuentes del proyecto y hay un mayor control de cambios gracias a las *pull request* que permiten a los

[1] Datos obtenidos de: `http://www.sqlite.org/speed.html`

desarrolladores evaluar los cambios antes de introducirlos en el código final.

A pesar de las virtudes de GIT y Github, finalmente nos decantamos por el uso de SVN y Sourceforge porque AraWord y TICO ya estaban desplegados usando este sistema y tenían sus páginas creadas en Sourceforge, de manera que al usuario le podía resultar confuso que AraSuite se distribuyera otra plataforma como Github con GIT.

4.2. Herramientas utilizadas

Como hemos dicho, algunas de las herramientas utilizadas a lo largo del desarrollo venían impuestas por el entorno actual donde se iba a realizar el proyecto.

- **Eclipse IDE:** IDE de desarrollo de Java. Este entorno venía definido como requisito del entorno. Además se ha usado junto con un "plugin" llamado WindowBuilder que permite desarrollar interfaces mediante un editor WYSIWYG.
- **Omnigraffle:** Aplicación que permite desarrollar cualquier tipo de diagrama o boceto de interfaz. Se ha utilizado para diseñas los diagramas de la aplicación y los bocetos de las interfaces.
- **DokuWiki:** Con esta herramienta se generó una Wiki en la que se fueron plasmando la documentación que se fue generando a lo largo del proyecto. Se decidió usar este método porque permitía un fácil acceso al tutor para la revisión de los documentos y además le permitía añadir comentarios o modificaciones, generando así un entorno de documentación "distribuído". La URL de la Wiki es: http://gidhe.es/agomez.
- **Dropbox:** Aplicación que permite almacenar archivos en internet con un sistema muy básico de control de versiones. Esta aplicación se ha usado una vez que se comenzó la documentación en LaTeX.
- **LaTeX:** Es una herramienta para componer documentos. Se ha usado para generar la versión final de la documentación.
- **Sublime Text 3:** Es un editor de textos para Mac OS X. Se ha usado junto con un paquete *LatexTools* que facilita el desarrollo de textos en LaTeX.

4.3. Metodología de desarrollo

Para seleccionar la metodología de desarrollo que se iba a usar en el proyecto se analizaron las dos metodologías ágiles más extendidas.

4.3.1. Metodología Scrum

La metodología Scrum presenta los roles de *ScrumMaster*, que dirige y organiza las tareas a realizar en cada periodo, el *ProductOwner*, que define los hitos a alcanzar y el *Team* que representa al equipo de desarrollo.

El Scrum define periodos de trabajo de entre 1 y 4 semanas llamados Sprints. Al principio de cada Sprint se realiza un *Sprint planning* en el que se eligen qué tareas, de las que ha definido el *ProductOwner* en el *backlog* (lista de tareas pendientes), se van a realizar.

Al finalizar el sprint, se realiza una *retrospective*, en la que se analizan los posibles problemas que hayan surgido durante ese periodo y se finaliza con una pequeña demo de las tareas realizadas.

4.3.2. Metodología Extreme Programming

La metodología Extreme Programming, comúnmente llamada metodología XP, define periodos de trabajo de entre 1 y 2 semanas. Al principio de este periodo se debe realizar una reunión llamada *Planning game* en la que se concretan las tareas a realizar, esta reunión se divide en *Release Planning* donde se seleccionan los requisitos del cliente que se van a comprometer en el periodo y se plasman en *User Stories* y la *Iteration Planning* donde los desarrolladores dividen los requisitos identificados con anterioridad en pequeñas tareas.

Una de las principales características de esta metodología es la programación en parejas, llamada *Pair Programming*. XP define que las tareas pueden realizarse entre dos desarrolladores delante de un mismo ordenador, uno de los desarrolladores se encargará del código concreto que se está desarrollando mientras que el otro pensará en las influencias a gran escala que puedan suponer los cambios.

4.3.3. Elección de la metodología

Tras haber analizado dos de las metodologías ágiles disponibles, se determinó que la que mejor se adaptaba a la situación actual era una modificación de la Metodología Scrum ya que, aunque ambas se ajustaban globalmente a los requisitos especificados, XP está más indicada para desarrollos en equipos grandes en los que se pueda hacer *pair programming*.

Además, se definieron que los roles de los miembros del proyecto serían:

- **Project Owner:** Este rol estaría representado por el tutor del proyecto ya que conoce el backlog de tareas y los objetivos finales.
- **Scrum Master:** Este rol lo representaría el proyectante, ya que conoce mejor las exigencias de su calendario personal.

- **Development Team:** Este rol lo representaría el proyectante únicamente.

A la hora de elegir la herramienta que serviría como soporte de la metodología, se decidió usar la Wiki en vez de otras herramientas, como por ejemplo Jira, ya que resultaba fácil de instalar y mantener y el equipo ya había tenido experiencias anteriores satisfactorias. En ella se llevó un registro tanto de la documentación de la aplicación como del proceso de desarrollo. Para tener un seguimiento del proceso de desarrollo, en la Wiki se anotaban, el backlog, los sprints de desarrollo, el estado de las tareas, las fechas de entrega y las desviaciones.

4.4. Interfaces del GalleryManager

Durante la etapa de análisis se identificaron los casos de uso que se pueden ver en el anexo B. Posteriormente, en la etapa de diseño se detallaron los casos de uso mediante diagramas de actividad, anexo D. Además, se realizaron los prototipos de las interfaces del GalleryManager que podemos encontrar en el anexo H.

A continuación se muestran las interfaces que fueron creadas para ofrecer las nuevas funcionalidades o rediseñadas para mejorar su funcionalidad respecto de las ya existentes. El resto de interfaces se pueden ver en el anexo C.

Exportar búsqueda

A la hora de realizar una exportación, el usuario previamente tiene que realizar una búsqueda. La interfaz existente no permitía al usuario ver de forma rápida todos los resultados encontrados porque se mostraban en grupos de cuatro elementos con botones para avanzar o retroceder. Por esta razón, se decidió rediseñar la interfaz de exportación de una búsqueda para añadir un scroll vertical que facilitara al usuario recorrer todos los resultados encontrados de una forma más cómoda y rápida. El resultado del rediseño de la interfaz es el que se puede ver en la imagen de la figura 4.1.

Actualizar automáticamente pictogramas

La actualización automática de pictogramas es una de las nuevas funcionalidades que incluye AraSuite, por esta razón fue necesario crear la interfaz que se muestra en la figura 4.2. En esta interfaz se muestra al usuario la información que se ha obtenido del servidor sobre el nuevo paquete de pictogramas que está disponible para actualizar.

4.5. Puntos destacados de la implementación

En esta sección se muestran los puntos destacados surgidos durante el desarrollo de la aplicación.

Figura 4.1: Interfaz de exportación de búsqueda

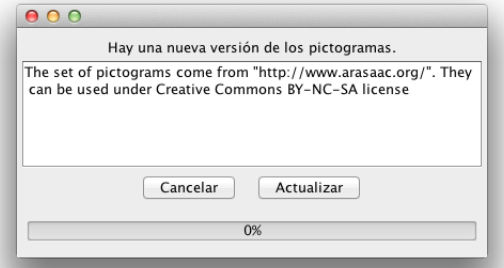

Figura 4.2: Interfaz de Actualizar automáticamente pictogramas

4.5.1. Rediseño de la interfaz de resultados de una búsqueda

Como ya he comentado en la sección 4.4, se realizó un rediseño en la forma en la que se mostraban los resultados de una búsqueda bien para los flujos de edición de un pictograma como para la exportación de una búsqueda. El diseño, como se puede ver en la interfaz de la figura 4.1 sufrió un cambio para añadir un scroll vertical que mostrara los resultados de la búsqueda. Este scroll vertical permite al usuario desplazarse por los pictogramas encontrados de una forma más rápida y fluida.

Durante el desarrollo de esta interfaz nos encontramos el problema de que si se realizaba una consulta que devolviera muchos resultados, la navegación por las imágenes devueltas era muy lenta o se colgaba por alcanzar el límite de

memoria RAM.

Esto era porque todas las imágenes, aunque no se mostraran al usuario estaban pintadas y ocupaban memoria RAM. Por lo tanto decidí sobrescribir el método de "renderización" de imágenes de la tabla por defecto, para pintar las imágenes en función de cuando se iban necesitando. De esta manera solo estarían ocupando memoria aquellas imágenes que se estuvieran mostrando al usuario.

El resultado fue, que el usuario podía desplazarse fluidamente por los resultados encontrados aunque estos fueran un gran número.

4.5.2. Localización de los archivos del GalleryManager

Uno de los problemas iniciales que nos encontramos fue que como el GalleryManager se ejecuta como un JAR desde cualquier aplicación, la base de datos, los pictogramas y los ficheros de configuración no podían ser relativos al JAR porque entonces el "path" era relativo a la aplicación que hacía uso del GalleryManager. De esta manera, si TICO ejecutaba el GalleryManager, se creaban unos ficheros de configuración, una BD y una carpeta de pictogramas, mientras que si lo hacía AraWord, se duplicaban estos archivos en su directorio.

Tras analizar las distintas posibilidades se decidió que la mejor opción era situar esa carpeta en el sistema de directorios del usuario, de esta manera conseguíamos que los archivos siempre estuvieran en el mismo sitio, se evitaban problemas de escritura por ser un directorio del usuario y se restringía de alguna manera el acceso directo a la BD y las imágenes evitando que el usuario podría corromper el sistema.

De esta manera los directorios se organizaron tal y como se ve en el apartado 3.5.

4.5.3. Generación y distribución de versiones intermedias

TICO y AraWord disponían de archivos build.xml que se ejecutaban con ANT y que permitían compilar y generar una versión en ambos casos.

AraSuite es un proyecto que combina estas dos aplicaciones y hace que las dos integren el GalleryManager para acceder a los pictogramas y los términos asociados. Por esta razón, los archivos build.xml que tenían las dos aplicaciones ya no servían porque no usaban la nueva dependencia del GalleryManager.

Por lo tanto, cada vez que se quería generar una versión sobre la que hacer "testing" se iniciaba un proceso muy tedioso que quedaba restringido únicamente a los desarrolladores e impedía que cualquier persona que no estuviera familiarizada con el entorno fuera capaz de generar una versión. Por lo tanto se decidió desarrollar un archivo build.xml para cada uno de los proyectos que al ejecutarlo con ANT, compilaría las distintas aplicaciones resolviendo las dependencias que tenían cada una de ellas con el GalleryManager.

Posteriormente, se evolucionó este sistema como se puede ver en el apartado 4.8 y se creó una aplicación web llamada AraSuite Generator que ejecutaba estos build.xml para automatizar todo este proceso.

El resultado de esta solución era una web que permitía al proyectante o al tutor generar una versión con un único click, algo que fue vital de cara a probar nuevas funcionalidades, identificar errores y realizar un seguimiento del proyecto por parte del tutor, que se hacía a menudo por la metodología de desarrollo utilizada.

4.5.4. Internacionalización de la aplicación

Tanto TICO como AraWord son aplicaciones que están internacionalizadas. Sin embargo, al ser aplicaciones que se desarrollaron independientemente, cada una de ellas tiene sus idiomas y su sistema de internacionalización.

El problema es que ambas aplicaciones hacen uso del GalleryManager y, por lo tanto, el GalleryManager debe estar preparado para adaptarse a la forma de internacionalizar de TICO o AraWord, ya que las interfaces que muestra tienen que estar en el idioma de la aplicación *padre*. Dada esta situación teníamos varias dificultades que había que resolver.

Cómo hacer llegar al GalleryManager el idioma de la aplicación que lo ejecuta

El GalleryManager tiene una única manera de invocar sus interfaces que es a través de la invocación del "frame" principal. A la hora de invocar este "frame" se pasa como parámetro el idioma de la interfaz de la aplicación padre y el GalleryManager guarda el idioma en un fichero de configuración para que se use durante toda la ejecución.

A continuación se muestra un extracto del código fuente para ver cómo el GalleryManager recibe el idioma como parámetro y lo guarda en el archivo de configuración.

```
public mainFrame(String language) {

  // Save languge into configuration file.
  TConfiguration.setLanguage(language);

  // Load language for interface from configuration file
  TLanguage.initLanguage(TConfiguration.getLanguage());

  // [...]
}
```

Y, a continuación, se muestra cómo se invoca el GalleryManager desde una aplicación como TICO o AraWord.

```
public void actionPerformed(java.awt.event.ActionEvent evt) {
  mainFrame f = new mainFrame(G.applicationLanguage);
  f.setVisible(true);
```

```
    f.pack();
}
```

Cómo relacionar el idioma de la aplicación padre y los idiomas del GalleryManager

TICO y AraWord tenían formas de identificar los idiomas que no eran estándares, por ejemplo el Castellano se identificaba con la clave "Castellano", el Inglés con la clave "Ingles", etc. Por esta razón, y con intención de estandarizarlo, se decidió que el GalleryManager identificaría los idiomas usando el estándar ISO 639-1 [2].

Para relacionar las claves de TICO con el GalleryManager se hizo un fichero *lang.properties* como el que se muestra a continuación:

```
CASTELLANO=es
INGLES=en
FRANCES=fr
PORTUGUES=pt
PORTUGUES_BRASIL=br
CATALAN=ca
ITALIANO=it
```

Así, el GalleryManager al recibir "Castellano" buscaría la clave en el fichero *language.properties*, que en este caso sería "es" y cargaría el fichero de idiomas *es.properties*.

Este sistema permite añadir un idioma al GalleryManager de una forma sencilla ya que solo habría que introducir la nueva clave en el fichero *language.properties*, por ejemplo, "Euskera=eu" y añadir el fichero *eu.properties* al directorio de idiomas del GalleryManager.

4.5.5. Actualización automática de pictogramas

Uno de los requisitos de AraSuite, era que tuviera implementada la actualización automática de pictogramas. Esta actualización ha sido uno de los puntos importantes que se han afrontado durante el desarrollo del proyecto.

Finalmente se ha optado por la solución que se muestra en el diagrama de actividad de la figura 3.2. En resumen, esta solución ha consistido en establecer en el archivo de configuración de AraSuite el servidor donde encontrará la información sobre las nuevas actualizaciones.

Cuando el usuario quiera realizar una actualización, AraSuite consultará un archivo remoto de información con una estructura como la que se muestra a continuación:

```
DOWNLOAD_URL=https://s3.amazonaws.com/pictos.zip
MD5= 01be832107feefebba5bf76275e8185a
RELEASE_DATE=04-07-2013
```

[2] Más información en: `http://es.wikipedia.org/wiki/ISO_639-1`

De este fichero obtendrá principalmente, la fecha del paquete, y la URL de descarga. En caso de que se haya publicado un nuevo paquete y que el usuario quiera actualizar, lo descargará y comenzar con el proceso de importación.

De esta manera el sistema queda más configurable y nos permite mover los paquetes de pictogramas a otros servidores en caso de que se espere un gran número de actualizaciones, y tan solo habría que cambiar la URL de descarga en el fichero remoto de información.

4.5.6. Creación y distribución de versiones finales con instalador

Uno de los requisitos de AraSuite era generar instaladores de la aplicación para los tres sistemas operativos predominantes, Windows, MacOS y Linux.

Se buscaba un generador de instaladores que a partir del paquete creado por el generador de versiones, que se puede ver en el anexo K, creara 3 ejecutables para cada uno de los SO que hemos comentado anteriormente.

En este momento se realizó un análisis de varias aplicaciones que permitían generar los instaladores, tales como: IZPack, PackJacket o Install4J.

Finalmente nos decantamos por usar Install4J ya que además de generar el instalador, permite crear lanzadores de aplicaciones, incluir o excluir archivos para cada SO, genera desinstalador y permite poner splash screens e iconos configurables.

Install4J además también permite ser integrado con ANT de manera que en un futuro se podría automatizar toda la generación de instaladores e integrarlo con un sistema de integración continua que nos generaría los instaladores a partir del código fuente, ante eventos como la creación de un “TAG” en el repositorio, o en cada “commmit” en “trunk”.

4.5.7. Mejora de la velocidad de importación

Una de las necesidades de este proyecto es que la velocidad de acceso a los pictogramas fuera la misma o mejor que la existente hasta el momento, por esta razón, en este apartado se va a realizar un estudio de una de las partes de la aplicación que mayor coste operacional tiene, la importación de pictogramas a la base de datos. Esta importación consiste recorrer e importar los datos de pictogramas que aparecen en un XML y copiar los pictogramas al directorio interno del GalleryManager.

La operación de trasladar las imágenes al directorio del GalleryManager es una operación de copia de ficheros. Por lo tanto las posibles reducciones que se puedan llevar a cabo en este apartado son bastante escasas. Es por eso por

lo que vamos a centrar nuestros esfuerzos en reducir el proceso de "parsing" e importación de los datos de los pictogramas desde el XML a la base de datos.

Muestra usada para la importación

En el estudio de la velocidad de importación se han analizado los tiempos de ejecución para el antiguo entorno usado en TICO y AraWord y para el nuevo entorno de AraSuite. Para ello se han realizado 10 ejecuciones usado una muestra de pictogramas de las siguientes características:

- Número de pictogramas a importar: 1400
- Espacio en disco de los pictogramas: 67MB
- Términos a insertar en la BD: 22.217

Resultados del análisis

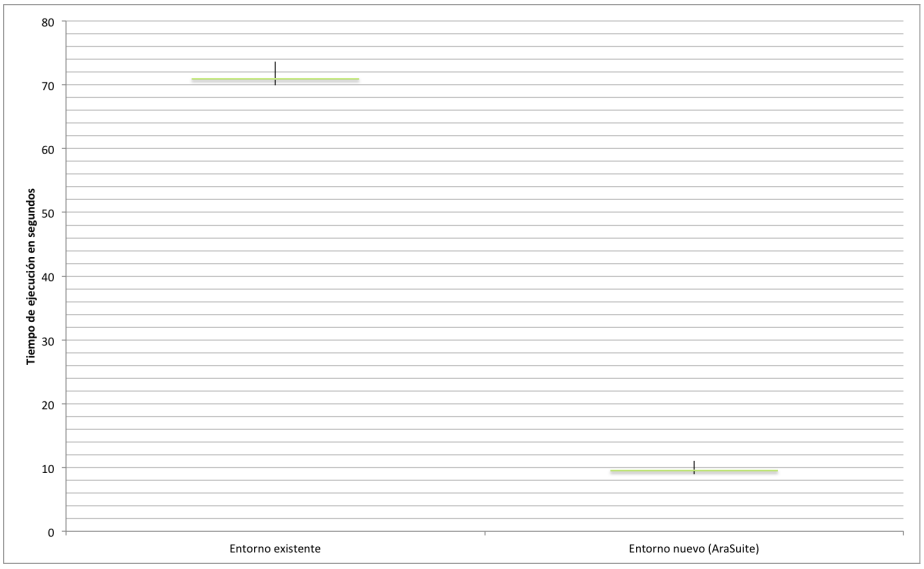

Figura 4.3: Gráfica comparativa de la velocidad de importación

Los resultados obtenidos son un tiempo medio de ejecución de 71 segundos para el caso del entorno ya existente y de 9,6 segundos para el caso del nuevo entorno de AraSuite. En la gráfica de la figura 4.3 se pueden ver los tiempos mínimo, máximo y medio para cada uno de los entornos. Analizando los resultados obtenidos el tiempo medio de importación de la nueva versión de AraSuite (9,6 segundos) presenta una mejora del 86 % frente a la versión de TICO (71 segundos).

4.6. Ejecución del plan de pruebas

En la sección 2.5 se ha definido el plan de pruebas que se iba a ejecutar de cara a garantizar la calidad del software desarrollado.

En esta sección se explica cuando se han realizado las ejecuciones del plan de pruebas, los resultados obtenidos e información adicional sobre cada uno de ellos.

4.6.1. Ejecución de las pruebas unitarias

Durante el desarrollo de este proyecto se creó un entorno de pruebas unitarias que permitieron probar funciones complejas para garantizar que cualquier desarrollo posterior no modificara el resultado esperado.

Esto, además de garantizar el desarrollo realizado, también debía servir para establecer una base que permitiera tener pruebas unitarias en futuros desarrollos. A continuación se explica cómo se estructuraron y se ejecutaron las pruebas unitarias.

Descripción de las pruebas unitarias

Se han especificado ciertos patrones para ayudarnos a identificar en qué consiste cada una de las pruebas unitarias que se desarrollan.

Para hacerlo lo más mantenible posible, se ha evitado añadir JavaDoc explicativo a cada uno de los tests ya que si el test cambia también hay que cambiar el JavaDoc y podemos llegar a situaciones incongruentes.

Así que para saber qué realiza cada prueba unitaria se usará el título de la misma de la siguiente manera: Nombre del método probado, precondición de la prueba y resultado esperado, usando un formato *camelcase*.

Un ejemplo de título de una prueba unitaria sería: "escapeQueryWithSeveralRegexShouldReturnEscaped" esto identificaría una prueba unitaria que haría lo siguiente:

- **Nombre del método probado:** escapeQuery.
- **Precondición:** Se van a usar varias expresiones regulares
- **Resultado esperado:** Las queries correctamente escapadas. (Opcional)

Entorno de ejecución de pruebas

Las pruebas unitarias se han desarrollado usando JUnit 4.0 ya que está completamente integrado tanto con el IDE (Eclipse) como con ANT y resulta muy

cómodo para realizar test unitarios sobre código Java. La ejecución se ha automatizado mediante un "target" de ANT para poder ejecutarlo durante la compilación de la aplicación y que un fallo en las pruebas unitarias haga que no se compile una versión. De esta manera garantizamos que todas las versiones distribuidas habrán pasado satisfactoriamente las pruebas unitarias.

Cada una de las ejecuciones de las pruebas unitarias dan como resultado un informe HTML que contiene información sobre los tests que se han pasado, el tiempo de ejecución, y el resultado de cada uno de ellos.

4.6.2. Ejecución de las pruebas de sistema

En el anexo F se identificaron los puntos críticos de la aplicación. Al finalizar el desarrollo se realizaron pruebas para confirmar que la aplicación era estable en cualquiera de esos puntos críticos.

El proceso y los resultados de las pruebas de sistema realizadas se encuentra en el anexo G. La ejecución de las pruebas del sistema fue satisfactoria para todos los puntos críticos, sin embargo, la ejecución de alguna de ellas sirvió para detectar puntos de mejora en el rendimiento de la aplicación.

Por ejemplo, las pruebas de exportación de una búsqueda detectaron que esta operación se prolongaba durante mucho tiempo. Por esta razón se decidió modificar el algoritmo de exportación para minimizar el tiempo de ejecución.

4.6.3. Ejecución de las pruebas de aceptación

Las pruebas de aceptación tienen que garantizar que la aplicación se comporta de manera estable en situaciones normales de ejecución.

Una vez que se tuvo una versión que se consideraba estable de la aplicación se liberó la versión AraSuite 1.0.0 para los 3 sistemas operativos y se puso al alcance de todos los usuarios en el portal de Sourceforge.

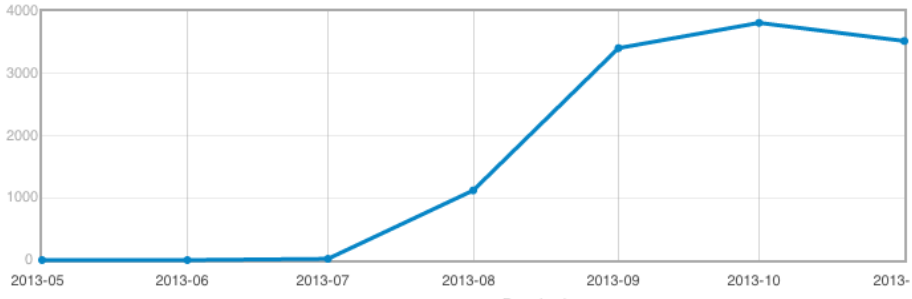

Figura 4.4: Gráfica de descargas de AraSuite en Sourceforge

Tal y como se puede ver en la gráfica extraída de Sourceforge de la figura 4.4, AraSuite comenzó teniendo 1120 descargas en el primer mes, creciendo

exponencialmente hasta las casi 4000 descargas actuales. Gracias a la activa participación de los usuarios, se han ido recogiendo errores y sugerencias de mejora, que han sido incluidos por el equipo de desarrollo en nuevas versiones de AraSuite.

Cabe destacar que ninguno de los errores encontrados desde la versión 1.0.0 se ha considerado un error crítico o bloqueante. En la mayoría de los casos han sido errores menores que no influían de forma negativa en la estabilidad de la aplicación.

Por lo tanto, se considera que estas pruebas de aceptación basadas en el feedback recibido por parte de los usuarios son suficientes para garantizar que la aplicación es estable.

4.7. Definición del flujo de desarrollo

En esta sección se definen todos los detalles relacionados con el desarrollo de nuevas funcionalidades o corrección de errores en AraSuite, veasé el anexo J para información más detallada.

4.7.1. Estructura del repositorio

AraWord y TICO son aplicaciones en continuo desarrollo donde varias personas están trabajando en nuevas funcionalidades o correcciones de errores. Uno de los problemas detectados en los repositorios de TICO y AraWord es que no hay un flujo de trabajo definido por lo que a lo largo del tiempo se ha llegado a una situación de falta de organización en ambos repositorios que ni si quiera sigue las convenciones de uso de SVN.

Con la creación de AraSuite se ha creado un repositorio nuevo que sigue las normas de SVN y además define flujos de trabajo claros que facilitan el desarrollo conjunto. A continuación se muestra como queda organizado el repositorio de AraSuite:

- **Trunk:** Contiene la última versión estable de la aplicación con las nuevas funcionalidades y correcciones realizadas.
  - **AraWord:** Contiene los ficheros de AraWord.
  - **GalleryManager:** Contiene los ficheros del GalleryManager.
  - **TICO:** Contiene los ficheros de TICO.
  - **Utils:** Contiene aplicaciones externas relacionadas con AraSuite.
    - **Arasaac2xml:** Aplicación que genera un XML con la información de Arasaac.
    - **ArasuiteVersioner:** Aplicación web que genera versiones de AraSuite.

- **Branches:** Contiene los desarrollos de nuevas funcionalidades. Deben tener un nombre descriptivo de la funcionalidad realizada.
- **Tags:** Contiene cada una de las versiones liberadas de AraSuite.

4.7.2. Flujos de trabajo

A la hora de trabajar en AraSuite deben seguirse los flujos de trabajo definidos a continuación.

- **Creación de nuevas funcionalidades:** Deben realizarse siempre bajo un branch con un nombre descriptivo (add-sound-to-cells, create-new-awesome-functionality, etc.) Una vez terminadas y sincronizadas con trunk deben ser sometidas a varias pruebas de ejecución que garanticen que la nueva funcionalidad no ha introducido errores en el código existente. Una vez terminado este proceso pueden ser mergeadas con trunk.
- **Corrección de bugs:** Se llevará un seguimiento de los bugs mediante los tickets de Sourceforge, añadiendo una descripción detallada del problema, la criticidad del bug y la persona que lo corregirá. Los bugs se corregirán directamente en trunk puesto que debe tratarse de pequeñas modificaciones que únicamente resuelven el problema detectado.
- **Release de versiones de AraSuite:** Cuando se considere que hay suficientes cambios se realizará un pequeño periodo de pruebas sobre trunk, una vez finalizado y comprobado que todo funciona corréctamente se generará un nuevo tag. El tag se nombrará numéricamente y de forma ascendente en tres niveles x.x.x, por ejemplo: 1.1.0, 2.2.0, etc. El primer valor indicará "major releases" de la aplicación que incluirán grandes cambios y funcionalidades. El segundo valor agrupará a pequeñas funcionalidades o correcciones de bugs, mientras que el tercer valor servirá para identificar pequeñas releases liberadas con 1 o 2 bugs resueltos. Estos tags nunca serán modificados.

En la imagen de la figura 4.5 puede verse de forma más descriptiva la organización del repositorio.

4.8. Generador de versiones

Como se ha visto en el apartado 4.5.3 para probar una versión de AraSuite hacía falta hacer un checkout de los tres repositorios TICO, AraWord y GalleryManager. Esto dificultaba mucho que cualquier persona que no estuviera directamente relacionada con el desarrollo pudiera realizar pruebas de nuevas funcionalidades o encontrar bugs en la versión de AraSuite.

Entonces se decidió que facilitar la generación de versiones de AraSuite podía ser crítico de cara a ir descubriendo nuevas funcionalidades que pudieran ser

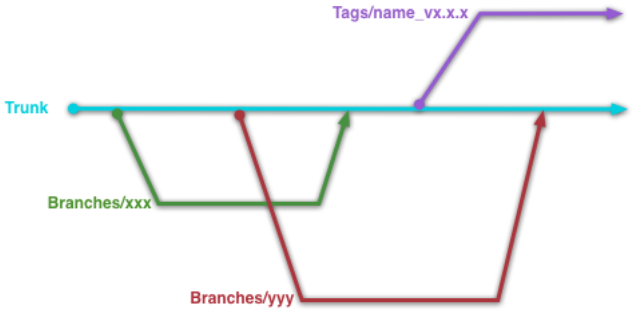

Figura 4.5: Estructura del repositorio

útiles, e identificar bugs de forma temprana gracias a distribuir versiones entre usuarios que testearían la aplicación. Por esta razón se decidió crear un generador de versiones con una usabilidad muy sencilla.

Este generador de versiones está liberado en la url `http://arasuitegenerator.adgomez.com`. Para ver información detallada sobre el generador de versiones se puede ver el anexo K.

5. Gestión del proyecto

La gestión del proyecto se vio muy influenciada por la situación actual del proyectante, ya que en el momento de empezar el proyecto ya había comenzado a trabajar en una empresa y por lo tanto el tiempo dedicado al proyecto estaba fuertemente influenciado por los problemas de agenda y las variaciones en las cargas de trabajo de la misma.

5.1. Sprints de desarrollo

La organización del proyecto se ha visto influenciada principalmente por la disponibilidad del proyectante. Los periodos de trabajo, o sprints, se definieron en ciclos de 3 semanas, en la imagen de la figura 5.1 podemos ver el tiempo invertido en cada periodo y su desviación.

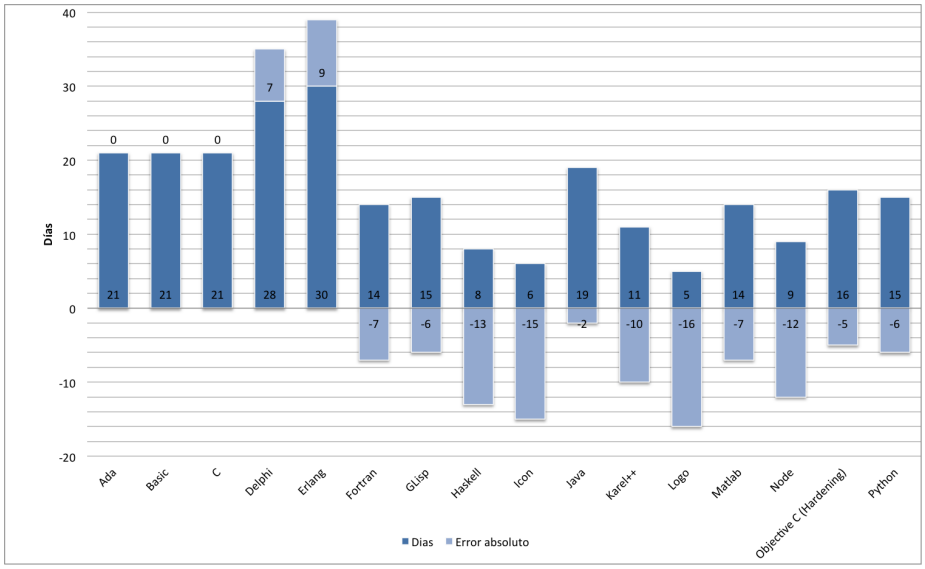

Figura 5.1: Análisis de desviación de los sprints

6. Conclusiones y trabajo futuro

Al comienzo de este proyecto existía TICO y AraWord, ambas aplicaciones eran funcionales pero seguían caminos de desarrollo que no convergían. Esto suponía un problema de cara al futuro de las aplicaciones y la evolución de las mismas, ya que, aunque ambas explotan los pictogramas y sus términos asociados, lo hacen de distinta manera, introduciendo errores por duplicado y abriendo la posibilidad de que diverjan tanto en su uso que el usuario se sienta confuso.

Por eso, en este proyecto fin de carrera se ha completado el desarrollo de la primera versión de AraSuite y se ha establecido el camino sobre el que continuará su evolución. Esta aplicación ha conseguido reunir las dos aplicaciones existentes de TICO y AraWord, introduciendo nuevas funcionalidades y mejoras al ecosistema ya existente, haciéndolas más robustas y estables.

Además, como resultado final del proyecto se ha conseguido que los usuarios de TICO y AraWord migren a la nueva aplicación de AraSuite sin que esto suponga un problema. Es más, a día de hoy, AraSuite tiene más de 2500 descargas al mes y se postula como una aplicación referente en su sector, siendo la satisfacción de los usuarios una de sus mejores puntos.

Como valoración personal tengo que añadir que participar en el desarrollo de AraSuite y poder ofrecer a los usuarios una aplicación tan aceptada, ha sido una satisfacción personal y un orgullo, el colofón de la aplicación sería que la comunidad de desarrolladores aproveche su licencia GNU/GPL para poderla evolucionar y adaptar a las nuevas necesidades que surjan.

6.1. Trabajo futuro

AraSuite es una aplicación que requerirá de nuevas funcionalidades según vayan identificándose las necesidades que vayan demandando sus usuarios, pero a priori, y centrándonos en el desarrollo, sería necesario establecer un sistema de integración continua como Jenkins que automatice aún más la generación de versiones.

Además, ya que AraSuite está enfocado al uso de pictogramas veo como punto importante evolucionar la actualización automática para que sea una ac-

tualización progresiva que permita a los usuarios descargar de forma automática únicamente aquellos pictogramas que desee o quizás pictogramas separados por paquetes, por ejemplo, pictogramas de animales, de objetos concretos, lugares, etc.

Por otro lado, y como un desarrollo paralelo a AraSuite, sería una gran opción convertir la aplicación de escritorio existente en una aplicación online que ofrezca al usuario las mismas funcionalidades pero con las capacidades de las aplicaciones en la nube. De esta manera, podríamos mantener la gestión de pictogramas de lado del servidor y el usuario siempre usaría la última versión de la aplicación y de los pictogramas.

Bibliografía

[1] Web de Java `http://www.java.com/`

[2] Web del Proyecto TICO `http://www.proyectotico.com`

[3] Web de ARASAAC `http://www.arasaac.org`

[4] Web de SQLite `https://sqlite.org`

[5] Web de Jenkins `http://jenkins-ci.org`