UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

HERRAMIENTA DE AUTOR PARA LA DEFINICIÓN DE GUÍAS DE INTERACCIÓN ADAPTATIVAS PARA PERSONAS CON DISCAPACIDAD COGNITIVA

Ingeniería de Telecomunicación

PABLO JESÚS BUENO SOTO

SEPTIEMBRE 2013

Herramienta de autor para la definición de guías de interacción adaptativas para personas con discapacidad cognitiva

AUTOR: Pablo Jesús Bueno Soto

TUTOR: Javier Gómez Escribano

PONENTE: Germán Montoro Manrique

Ambient Intelligence Laboratory (AmILab)

Dpto. de Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Septiembre 2013

RESUMEN

Este trabajo se encuentra enmarcado dentro de las tecnologías asistivas, las cuales tienen como objetivo ayudar a personas con algún tipo de discapacidad o deficiencia utilizando para ello objetos, equipos, sistemas, *software* y/o servicios.

En concreto, se parte de un sistema previo llamado aQRdate que consiste en una aplicación móvil destinada a usuarios con daño cerebral adquirido, buscando complementar la clásica sesión de rehabilitación en la que el paciente realiza actividades cotidianas de diferentes complejidades mientras es observado por su tutor. Dicha aplicación escanea códigos QR que se encuentran en diferentes elementos del entorno y, a partir de ellos, muestra de forma automática un manual adaptado al elemento, el usuario y sus necesidades, y el dispositivo.

Sin embargo, el sistema descrito anteriormente sólo cuenta con la aplicación móvil destinada al paciente (persona con daño cerebral adquirido). Cuando los profesionales que atienden a este tipo de usuarios quieren modificar actividades ya existentes en el sistema o introducir nuevas, no pueden pues no cuentan con los conocimientos necesarios para poder actualizar la información directamente en el servidor. Por ello, este Proyecto Fin de Carrera se propone completar este sistema desarrollando una herramienta de autor que facilite la creación y gestión de las instrucciones necesarias para la realización de actividades de la vida diaria, por parte del personal médico, de una forma sencilla e intuitiva; sin que sea necesario tener conocimientos informáticos avanzados (como hasta ahora).

PALABRAS CLAVE

Herramienta de autor, tecnologías asistivas, daño cerebral adquirido, manual adaptado, códigos QR.

ABSTRACT

This work is related to assistive technologies which aim is to help people with any kind of disabilities or deficiencies by using objects, hardware, software, and/or services.

Specifically, the basis of this project is the system called aQRdate which is a mobile application intended for users who suffer acquired brain injury, in order to support the ordinary rehabilitation sessions in which the patient performs daily life activities of different complexities while is being supervised by his or her caregiver. This application scans QR codes situated in different environmental elements and based on them it shows automatically an adapted manual to the element, the user and his or her needs, and the device.

However, the system described above comprises only the mobile application designed for the patient (who has acquired brain injury). When professionals who take care of this type of users want to modify or add new activities, they cannot do it as they do not have the knowledge for updating the information in the server. For that reason, this Master Thesis proposes to complement this system developing an authoring tool that facilitates the medical professionals to create and manage the necessary instructions to perform daily life activities in a simple and intuitive way; without the need of having advanced computer skills (as heretofore).

KEY WORDS

Authoring tool, assistive technologies, acquired brain injury, adapted manuals, QR codes.

<<A veces sentimos que lo que hacemos es tan solo una gota en el mar,

pero el mar sería menos si le faltara una gota.>>

Teresa de Calcuta (1910 - 1997).

AGRADECIMIENTOS

En primer lugar quiero agradecer a mi tutor, Javier, la confianza en mí desde el inicio de este trabajo. Ya sea desde España o Finlandia, me ha ayudado muchísimo a lo largo de todo este recorrido con sus ideas, comentarios y opiniones. Y como no, no puedo olvidarme también de los momentos de risas tan necesarios. ¡Gracias por todo Javi!

Gracias también a mi ponente Germán. Desde que escogí este Proyecto Fin de Carrera se convirtió en mi segundo tutor. Siempre ha estado disponible para cualquier cosa que he necesitado y, al igual que en el caso de mi tutor Javier, sin su trabajo este proyecto no hubiese sido lo mismo. Por cierto, ¡en todos estos años de carrera no he visto nunca a un profesor contestar tan rápido los emails!.

No puedo olvidarme del laboratorio donde se ha desarrollado este PFC: AmILab. En él he conocido a grandes compañeros y amigos que han contribuido de manera decisiva a que mi trabajo saliese adelante. Muchas Gracias chicos ;).

Desde que empecé la carrera tuve la suerte de rodearme de buena gente que con el paso del tiempo se convirtieron en grandes amigos. Con ellos he pasado muchísimos momentos buenos tanto fuera (fiestas, cumples, viajes…) como dentro de la EPS (clases, prácticas, estudios, exámenes…) imposibles de olvidar. Ahora empieza una nueva etapa y aunque de otra forma, sé que cuento con vosotros del mismo modo.

¡No, no me he olvidado de vosotros! Mis amigos del colegio, pilar fundamental de mi vida: Cris, Xejo (Sergio), Paloma, Esther y Duque (Alejandro). Este trabajo es vuestro también pues me habéis escuchado y animado en infinidad de ocasiones a lo largo de todos estos años.

Por último, pero no menos importante, le agradezco a mi familia su apoyo incondicional. Especialmente a mis padres, los cuales han dado siempre todo por mí, en cada paso que he dado y en éste especialmente, con el único objetivo de verme feliz. Sin vosotros nada sería posible.

¡MUCHAS GRACIAS A TODOS!

GLOSARIO DE ACRÓNIMOS Y DEFINICIONES

3G. *Tercera Generación de telefonía móvil.*

3G es la abreviación de tercera generación de transmisión de voz y datos a través de telefonía móvil mediante UMTS (*Universal Mobile Telecommunications System* o servicio universal de telecomunicaciones móviles).

AMILAB. *Ambient Intelligence Laboratory.*

Laboratorio de inteligencia ambiental de la Escuela Politécnica Superior de la Universidad Autónoma de Madrid, lugar de desarrollo de este proyecto.

API. *Application Programming Interface.*

Interfaz de programación de aplicaciones es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción. Son usadas generalmente en las bibliotecas.

AQRDATE. *Sistema para el recordatorio de actividades de la vida diaria a personas con daño cerebral adquirido.*

Sistema desarrollado en el Trabajo Fin de Máster de Javier Gómez Escribano cuyo objetivo es complementar la rehabilitación clásica de pacientes con daño cerebral adquirido, en la que el paciente realiza actividades cotidianas con el uso del teléfono móvil. En este dispositivo se muestra de forma automática un manual adaptado al usuario y sus necesidades para poder llevar a cabo estas actividades.

AT. *Ayuda Tecnológica.*

Este término hace referencia a toda la tecnología diseñada para compensar una deficiencia o discapacidad.

CEADAC. *Centro de referencia estatal de atención al daño cerebral.*

Está destinado a la rehabilitación integral e intensiva y a la promoción de la autonomía personal en fase subaguda de personas con lesión cerebral adquirida y no progresiva. En él se realizaron con éxito las pruebas del proyecto aQRdate.

CSS. *Cascading Style Sheets.*

Lenguaje de hojas de estilos usado para describir la presentación semántica (el aspecto y formato) de un documento o aplicación.

DCS. *Daño cerebral Sobrevenido o adquirido.*

Hace referencia a personas afectadas por lesiones cerebrales que irrumpen de manera brusca e inesperada en su trayectoria vital, provocando, en la mayoría de los casos, secuelas muy variadas y complejas que afectan profundamente a la autonomía del afectado.

FEAPS. *Confederación española de organizaciones en favor de las personas con discapacidad intelectual o del desarrollo.*

Organismo que trabaja de manera intensa en la inserción laboral de personas con discapacidad intelectual.

GTT. *Gestor de Tiempos y Tareas.*

Es una herramienta para favorecer la autonomía de las personas con discapacidad mental en el ámbito laboral.

GUI. *Graphical User Interface.*

La interfaz gráfica de usuario es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador.

HHAA. *Herramienta de Autor.*

Las herramientas de autor son aplicaciones informáticas que facilitan la creación, publicación y gestión de los materiales educativos en formato digital a utilizar en la educación a distancia mediada por las TIC.

HTML. *HyperText Markup Language.*

Lenguaje de marcación diseñado para estructurar textos y presentarlos en forma de hipertexto, que es el formato estándar de las páginas web.

IDE. *Integrated Development Environment.*

Un entorno de desarrollo integrado es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. Ejemplos de ello son Netbeans o Eclipse.

IEEE. *Institute of Electrical and Electronics Engineers.*

Es una de las mayores asociaciones en el campo de la ciencia y la tecnología. Uno de los trabajos más conocidos que desarrollan es la estandarización.

ISO. *International Organization for Standardization.*

Organismo encargado de promover el desarrollo de normas internacionales de fabricación, comercio y comunicación para todas las ramas industriales a excepción de la eléctrica y la electrónica.

JRE. *Java Runtime Environment.*

El entorno en tiempo de ejecución de Java está conformado por una Máquina Virtual de Java o JVM, un conjunto de bibliotecas Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada. El JRE actúa como un "intermediario" entre el sistema operativo y Java.

JVM. *Java Virtual Machine.*

Una máquina virtual Java es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.

LIFO. *Last In First Out.*

El término LIFO se utiliza en estructuras de datos y teoría de colas. Guarda analogía con una pila de platos, en la que los platos van poniéndose uno sobre el otro, y si se quiere sacar uno, se saca primero el último que se puso. LIFO es el algoritmo utilizado para implementar pilas (como se puede ver en el apartado 5.2.2 de esta memoria).

LITE. *Laboratory of Information Technologies in Education.*

Laboratorio de tecnologías de la información en la educación de la Universidad Rey Juan Carlos cuyo interés principal es el uso de la tecnología informática con fines educativos. Forma parte del proyecto DEDOS (sección 2.1).

LMS. *Learning Management System.*

Un sistema de gestión de aprendizaje es un *software* instalado en un servidor web que se emplea para administrar, distribuir y controlar las actividades de formación no presencial (o aprendizaje electrónico) de una institución u organización.

LTE. *Long Term Evolution.*

Es un nuevo estándar de telefonía móvil basada por completo en protocolos IP. Para unos es vista como una evolución de la norma 3GPP UMTS (3G), para otros como un nuevo concepto de arquitectura evolutiva (4G).

LTSC. *Learning Technology Standards Committee.*

Es el comité de estandarización de las tecnologías aplicadas al aprendizaje perteneciente al IEEE que cubre prácticamente todos los aspectos del aprendizaje basado en ordenador.

MAPS. *Memory Aiding Prompting System.*

Sistema de mensajes diseñado a ayudar a personas con discapacidad cognitiva en sus actividades de la vida cotidiana.

OA. *Objeto de Aprendizaje.*

Según el LTSC el objeto de aprendizaje es: *"cualquier entidad, digital o no digital, que puede ser utilizada, reutilizada o referenciada durante el aprendizaje apoyado en la tecnología*".

PAIR. *Personal Ambient Intelligent Reminder.*

Sistema para ayudar a los cuidadores de personas con trastorno de memoria, permitiendo definir actividades cotidianas que el usuario debe realizar cada día y recordarle las que no se hayan realizado en el momento adecuado avisando al cuidador.

PDA. *Personal Digital Assitant.*

Ordenador de mano originalmente diseñado como agenda electrónica con un sistema de reconocimiento de escritura.

PFC. *Proyecto Fin de Carrera.*

PHP. *PHP Hypertext Pre-processor.*

Es un lenguaje de programación de uso general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico.

PREVIRNEC. *Plataforma tecnológica de rehabilitación cognitiva.*

Sistema creado para realizar rehabilitación a distancia destinado a personas con discapacidad cognitiva.

RF. *Requisito Funcional.*

Define una función del sistema, de *software* o de sus componentes.

RIA. *Rich Internet Application.*

Son aplicaciones web que tienen la mayoría de las características de las aplicaciones de escritorio tradicionales. Estas aplicaciones utilizan un navegador web estandarizado para ejecutarse y por medio de complementos o mediante una máquina virtual se agregan las características adicionales.

RNF. *Requisito No Funcional.*

Especifica criterios que pueden usarse para juzgar la operación de un sistema en lugar de sus comportamientos específicos, ya que éstos corresponden a los RF.

SCORM. *Sharable Content Object Reference Model.*

Conjunto de estándares y especificaciones que permite crear objetos pedagógicos estructurados. Estos objetos han de cumplir una serie de requerimientos.

TAC. *Assistive Technologies for Cognition.*

Bajo este concepto se fijan los requisitos que deben de cumplir las tecnologías para poder adaptarse a la discapacidad.

TFdM. *Trabajo Fin de Máster.*

TFG. *Trabajo Fin de Grado.*

TIC. *Tecnologías de la Información y la Comunicación.*

Agrupan los elementos y las técnicas empleadas para el tratamiento de los datos y la transmisión de la información. Están formadas principalmente por la informática y las telecomunicaciones.

UAM. *Universidad Autónoma de Madrid.*

UI. *User Interface.*

La interfaz de usuario es el medio con que el usuario puede comunicarse con una máquina, un equipo o un ordenador, y comprende todos los puntos de contacto entre el usuario y el equipo. Normalmente suelen ser fáciles de entender y fáciles de accionar.

URJC. *Universidad Rey Juan Carlos.*

WI-FI. *Marca de la Wi-Fi Alliance.*

Mecanismo de conexión de dispositivos electrónicos de forma inalámbrica. Sigue la norma IEEE 802.11.

WYSIWYG. *What You See Is What You Get.*

Se aplica a los procesadores de texto y otros editores de texto con formato que permiten escribir un documento viendo directamente el resultado final, frecuentemente el resultado impreso.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

1 INTRODUCCIÓN

1.1. MOTIVACIÓN

El daño cerebral sobrevenido o adquirido (DCS) hace referencia a diversas lesiones cerebrales, cuyo origen más común son los traumatismos cráneo-encefálicos, accidentes cerebro vasculares, tumores cerebrales, etc. cuyas consecuencias más frecuentes son algún tipo de discapacidad funcional y cognitiva, debido a las secuelas que generan dichos accidentes [1]. Esta enfermedad configura una realidad sanitaria y social de magnitud creciente y gravedad extraordinaria, que exige ofrecer una respuesta cada vez más especializada, en la medida en que los avances tecnológicos y la investigación lo permitan [2].

Las personas que padecen daño cerebral adquirido pueden presentar déficits de memoria y dificultad para ejecutar funciones. En concreto, algunos pacientes debido a la discapacidad cognitiva presentan problemas a la hora de realizar actividades de la vida diaria tales como cocinar, planchar, hacer la compra en el supermercado, etc. Esta situación provoca, en muchos de los casos, que estas personas no sean capaces de vivir de forma independiente [3]. Las dificultades anteriormente descritas motivan a pensar en un posible sistema que pueda ayudarles en la realización de las tareas diarias [4].

El área de investigación en el que se centra este trabajo, en donde se usa la tecnología para ayudar a personas con necesidades especiales, es conocida como “tecnología asistiva” [5].

Para la realización del presente proyecto se parte de un sistema previo llamado “aQRdate” [6] realizado por Javier Gómez Escribano en su Trabajo Fin de Máster

(TFdM), en el cual se propone generar un conjunto de instrucciones personalizadas para realizar una actividad de la vida cotidiana destinada a personas con daño cerebral sobrevenido o adquirido.

Para llevar a cabo cada tarea, el usuario dispone de un dispositivo móvil (generalmente teléfono móvil). Con este dispositivo el usuario acerca el teléfono a un código QR[1] [7], el cual se detecta y se decodifica de forma automática. La información que hay guardada en los códigos QR corresponde con el nombre de la actividad a realizar. Usando el nombre decodificado de la tarea, y basándose en las necesidades de cada usuario, el sistema muestra una serie de instrucciones para ayudar al individuo a realizar la actividad. Estas instrucciones son mostradas en el dispositivo móvil en diferentes formatos: texto, contenido gráfico o incluso mediante locuciones verbales [6].

El sistema anteriormente descrito se encuentra actualmente desarrollado y está siendo utilizado en el Centro de Referencia Estatal de Atención al Daño Cerebral (CEADAC)[2] en el proceso de rehabilitación en personas con daño cerebral adquirido, con resultados positivos. Cabe destacar que el proyecto "aQRdate" ha sido supervisado en todo momento por personal médico cualificado tales como neuropsicólogos, fisioterapeutas, terapeutas ocupacionales, etc.

Este Proyecto Fin de Carrera se centra en el desarrollo de un programa que facilite la creación y gestión de las instrucciones necesarias para la realización de actividades utilizando el sistema "aQRdate". La herramienta de diseño a crear, también llamada herramienta de autor, está destinada al personal médico encargado de la rehabilitación de personas con problemas cognitivos, los cuales desarrollarán las guías de interacción adaptadas a cada paciente.

[1] Código QR es una marca registrada de DENSO WAVE INCORPORATED.
[2] CEADAC: http://www.ceadac.es

1.2. OBJETIVOS

Como se ha comentado en el apartado anterior, este PFC cubre la necesidad de crear una herramienta de diseño (herramienta de autor) en la cual el personal médico que se encarga de la rehabilitación de pacientes con daño cerebral adquirido pueda diseñar los conjuntos de instrucciones necesarios para la realización de las tareas diarias.

Para ello, el sistema a desarrollar deberá cumplir las siguientes características:

1. El objetivo principal de este PFC es que, mediante el uso de la herramienta de autor, se puedan diseñar las tareas de la vida cotidiana paso a paso para la asistencia a personas con daño cerebral adquirido. Hasta ahora no era posible añadir tareas pues no había ninguna herramienta que posibilitase esta opción.
2. Uno de los objetivos fundamentales del presente proyecto es el desarrollo de una aplicación sencilla e intuitiva, para ser usado por personas con conocimientos informáticos a nivel usuario. Este objetivo es de vital importancia ya que, según los expertos, hasta más del 70% de los sistemas creados para las tecnologías asistivas terminan siendo abandonados [8]. Como problemas principales que provocan este abandono se encuentran la dificultad de configuración del sistema así como de adaptación del *software* al usuario.
3. Aunque, como se ha comentado en el punto anterior, la aplicación nace con el objetivo de ser fácilmente utilizable por personas que carecen de conocimientos informáticos avanzados, esta aplicación suministra un menú de ayuda, en donde el usuario puede revisar el funcionamiento de este programa.
4. Dado que diferentes pacientes con discapacidad cognitiva pueden requerir diferentes conjuntos de instrucciones a la hora de realizar una actividad, el sistema debe permitir personalizar las actividades para cada paciente.
5. El *software* a desarrollar no sólo posibilitará la opción de añadir nuevas tareas sino también modificar las ya existentes de una forma rápida. Además se tendrá muy en cuenta el concepto de reusabilidad para crear nuevas tareas partiendo de otras ya presentes en el sistema.
6. En todo momento, el usuario de esta herramienta tendrá a su disposición una vista previa de la tarea creada paso a paso, similar a la que la persona con daño cerebral adquirido vería en el teléfono móvil.

7. Se debe obtener una aplicación informática robusta. Puesto que este sistema será usado por personal no experto en aspectos relacionados con la informática, será de especial importancia que el programa no presente fallos.

1.3. SEGUIMIENTO TEMPORAL DEL PROYECTO

El diagrama de Gantt que recoge el seguimiento temporal del PFC se corresponde con la Figura 1.1. Las tareas principales que ha tenido este trabajo son:

1. **Estudio de las herramientas de autor**: El alumno se ha documentado sobre las herramientas de autor en varios ámbitos, haciendo especial hincapié en las que se engloban dentro de las tecnologías para la asistencia.
2. **Estudio del daño cerebral adquirido**: Se realiza un breve estudio sobre este tipo de lesiones. Se analizan las causas, el tratamiento y la evolución de este tipo de pacientes.
3. **Estudio del sistema aQRdate**: Se estudia todo el sistema previamente implementado, el cual motiva la realización de este PFC.
4. **Estudio del lenguaje de programación JavaFX**: Con los manuales que Oracle[3] suministra a los usuarios de este tipo de lenguajes y junto con el uso de los libros dedicados a JavaFX [9] y [10], se aprende a programar en este lenguaje.
5. **Diseño de la interfaz**: En esta etapa se fijan los requisitos funcionales y no funcionales que debe cumplir esta herramienta de autor. A partir de ellos se crean los prototipos para que sean valorados antes de ser finalmente implementados. En este proceso se requirió el diseño de tres prototipos distintos como recoge el cronograma.
6. **Implementación de la herramienta de autor**: Es el periodo más dilatado en el tiempo dentro de este PFC ya que engloba todas las etapas desde el inicio en la creación de esta aplicación hasta su última versión.
7. **Redacción de la memoria**: En este diagrama de Gantt se observa el periodo de tiempo destinado a la redacción de la presente memoria.
8. **Creación de la presentación**: Una vez terminada la memoria, se crea la presentación de este PFC.

[3] Oracle: http://www.oracle.com/

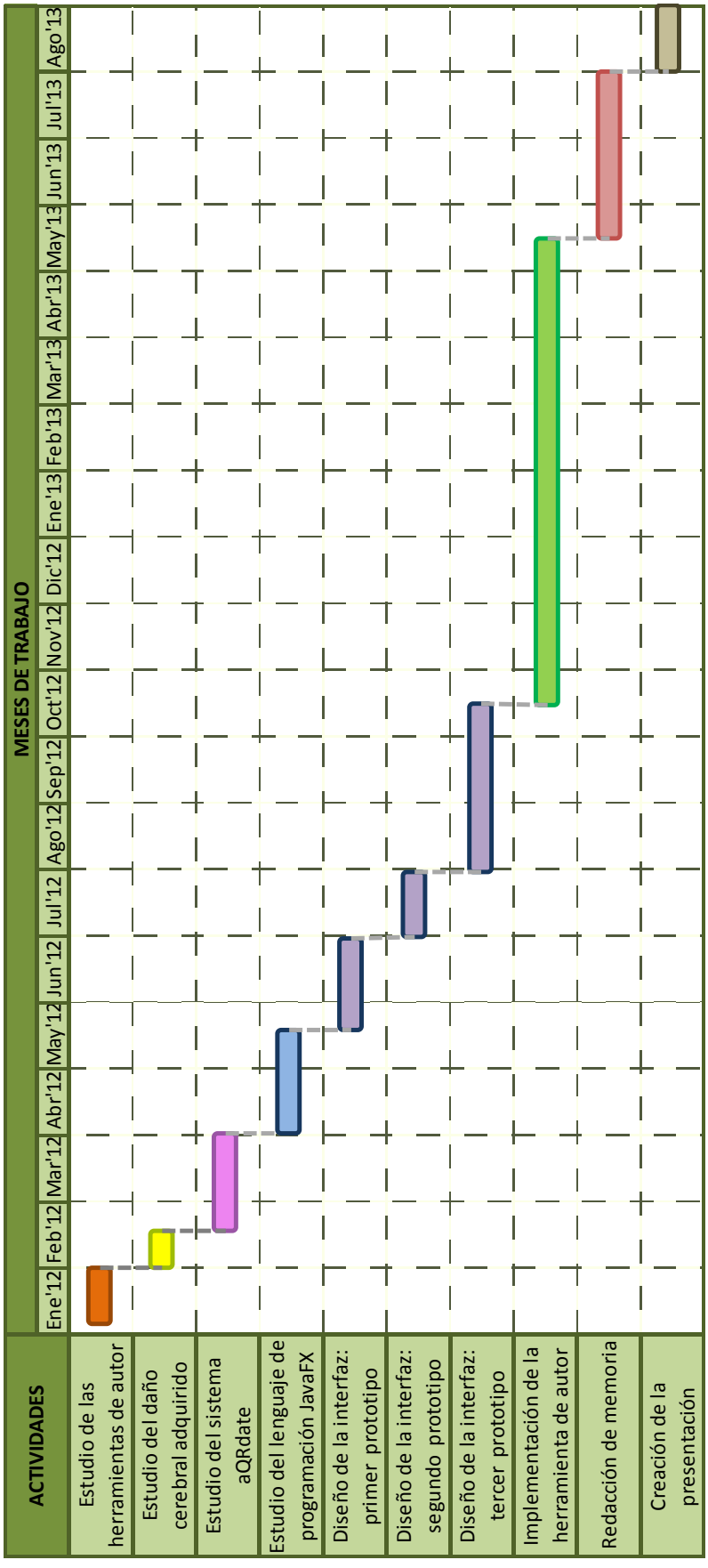


Figura 1.1: Cronograma del Proyecto Final de Carrera.

1.4. ORGANIZACIÓN DE LA MEMORIA

La memoria de este proyecto final de carrera está presentada de la siguiente forma:

CAPÍTULO 1. INTRODUCCIÓN

Este capítulo muestra la motivación para la realización de este proyecto, los objetivos que se pretenden conseguir durante su desarrollo, el cronograma temporal de este trabajo y finalmente la organización por capítulos de la memoria.

CAPÍTULO 2. ESTADO DEL ARTE

Se explica con detalle la definición de herramienta de autor según varios autores. Se muestran estas aplicaciones en diferentes ámbitos como pueden ser la educación o el diseño de páginas web. A continuación se introduce al lector el concepto de tecnologías para la asistencia (tecnologías asistivas) y por último se muestran ejemplos de este tipo de tecnologías, en especial dentro de ellas las que hacen referencia al objeto de este proyecto, las herramientas de autor.

CAPÍTULO 3. aQRdate: APLICACIÓN MÓVIL

En este tercer capítulo se narra el sistema llamado “aQRdate” cuyo objetivo es generar instrucciones personalizadas para realizar actividades de la vida cotidiana, destinado a personas con daño cerebral, mediante el uso del teléfono móvil.

CAPÍTULO 4. aQRdate: HERRAMIENTA DE AUTOR. DISEÑO

En el capítulo de diseño se recogen los requisitos que tiene que cumplir la interfaz a desarrollar. Una vez establecidos, se sigue un proceso de diseño iterativo que consiste en crear un prototipo y seguidamente evaluarlo. A partir de los resultados obtenidos, o bien se crea un nuevo prototipo que refleje todas las valoraciones y cambios oportunos, o bien se da por concluida la etapa de diseño y es entonces cuando se pasa a la etapa de implementación. Para la realización de este *software* ha sido necesario crear tres prototipos hasta cumplir con los requisitos fijados.

CAPÍTULO 5. aQRdate: HERRAMIENTA DE AUTOR. IMPLEMENTACIÓN

En este punto se relatará la herramienta de autor implementada. Se detallará cómo se ha implementado y se explicará, de manera individual, cada una de las opciones que

esta aplicación puede ofrecer al usuario. Además se incluye el resultado que usuario obtendría en diferentes ejecuciones del programa.

CAPÍTULO 6. CONCLUSIONES Y TRABAJO FUTURO

Por último, se mostrarán las conclusiones obtenidas por la realización de este proyecto fin de carrera así como aquellos aspectos que en un futuro se podrían realizar para mejorar el sistema presentado (trabajo futuro).

2
ESTADO DEL ARTE

En el proceso de producción de contenido educativo basado en el uso de las Tecnologías de la Información y las Comunicaciones (TIC), convergen conocimientos y habilidades que tienen su origen en el campo pedagógico, informático, del diseño gráfico, el vídeo y el trabajo con el sonido. Todo esto hace que se trate de un proceso complejo [11].

Esta situación obliga a los usuarios que crean este tipo de contenidos (autores) a desarrollar nuevos aprendizajes y cambios en su modo de actuación. Esta dificultad se ha visto reducida por la creación de aplicaciones informáticas, llamadas herramientas de autor (HHAA), lenguajes de autor o *software* de autor, que facilitan la creación de este tipo de información sin necesidad de conocimientos informáticos avanzados.

En este capítulo se describirán varias herramientas de autor de distintos ámbitos tales como la educación o el diseño web, pero se hará un especial hincapié en las que están englobadas dentro de las tecnologías asistivas.

2.1. HERRAMIENTA DE AUTOR

En la literatura existen varias definiciones de herramientas de autor, y posiciones diversas en cuanto a su denominación (véase referencias desde [12] hasta [20]). En gran parte de estas definiciones se identifica el contenido con los conocimientos, lo cual da una visión muy estrecha del primero.

En síntesis, podemos decir, que las herramientas de autor son aplicaciones que disminuyen el esfuerzo a realizar por los profesores, tutores, educadores, etc.,

ofreciéndoles indicios, guías, elementos predefinidos, ayudas y una interfaz amigable para crear materiales educativos en formato digital.

En este marco asumimos la definición de Tom Murray [21] : "*Las herramientas de autor son aplicaciones que tienen la intención de reducir el esfuerzo necesario para producir software, cargando con la responsabilidad en los aspectos mecánicos de la tarea, guiando al autor, y ofreciéndole elementos predefinidos que puede relacionar conjuntamente para satisfacer una necesidad particular*".

Cada sistema de autor provee herramientas o indicios que ayudan al autor en este proceso de estudiar sus partes y elaborar contenido al nivel necesario de detalle, para un modelo particular de elementos instructivos y sus relaciones.

En estos años ha ido creciendo este tipo de *software* cuyo objetivo es facilitar la creación, publicación y gestión de material didáctico, con distinto grado de interactividad, que permite elaborar información de tipo gráfico, audio, vídeo, etc. Se trata por tanto de aplicaciones informáticas que ofrecen la posibilidad de realizar un proceso de enseñanza-aprendizaje multimedia [11].

Sin embargo, la aparición de este tipo de herramientas mostró inicialmente un escenario desalentador para muchos docentes y terapeutas al encontrarse que estaban pensadas para usuarios con amplios conocimientos en informática.

Años después, con el aumento de la demanda de formación con el uso de las TIC se propició una mayor evolución en la producción de este tipo de contenidos multimedia; lo que potenció la investigación y el desarrollo, por parte de instituciones, universidades y empresas, de herramientas cada vez más fáciles de utilizar por el usuario y los profesores o tutores.

Los intentos para utilizar herramientas digitales en la creación de materiales educativos han sido muchos y han respondido a varios ejes de clasificación. Uno principal ha sido considerar la creación de materiales como una actividad estandarizable o, por el contrario, como una actividad creativa y completamente personal. Entre ambos extremos se han situado muchas opciones [11].

En la actualidad una concepción importante para crear este tipo de herramientas informáticas es la referida a los objetos de aprendizaje (OA), concepto que apareció en los años noventa del siglo pasado [22]. La definición de los OA según el Comité de Estándares de Tecnologías de aprendizaje (LTSC – *Learning Technology Standars Commite*), perteneciente al IEEE (*Institute of Electrical and Electronics Engineers*) es: "*cualquier entidad, digital o no digital, que puede ser utilizada, reutilizada o referenciada durante el aprendizaje apoyado en la tecnología*" [23]. Las características principales que se pueden destacar de este tipo de objetos son:

- Reutilizables. Pueden servir para la creación de otros OA y para diversas aplicaciones.
- Accesibles. Capacidad para acceder a estos objetos desde un sitio distante.
- Interoperables. Es decir, que se puede operar con estos OA en distintas plataformas de *software* y *hardware*.
- Portables. Se mueven y se albergan en diferentes plataformas sin cambio en su contenido.
- Durables. Permanecen intactos en las actualizaciones de *software* y *hardware*.

Uno de los modelos de referencia para la creación de objetos de aprendizaje más utilizado es el modelo de referencia SCORM (*Sharable Content Object Reference Model*) [24].

En la literatura podemos encontrar varios autores que hablan sobre los objetivos y características que han de tener las herramientas de autor. Uno de ellos, Tom Murray [21], fija como objetivos los siguientes:

1. Reducir el esfuerzo a la hora de generar el contenido dentro de la herramienta. Esta reducción de esfuerzo puede ser de tiempo, coste, y/u otros recursos.
2. Crear un *software* que pueda ser usado por personas con conocimientos informáticos a nivel usuario.
3. Ayudar al diseñador/autor a organizar los contenidos.
4. Dar soporte al autor fomentando los buenos principios de diseño. Para ello el sistema debe ser capaz de proporcionar soporte con el uso de la estructura del programa, puede hacer recomendaciones y también imposiciones.

5. Crear el material educativo por parte del autor de una forma rápida para poder ser evaluado con premura y por tanto, realizar los cambios oportunos en el diseño.

Este mismo autor destaca las características que a su juicio deben tener cualquier herramienta de autor:

- Fácil uso, requieren poca formación previa. La edición de la información debe ser WYSIWYG (*What You See Is What You Get*).
- Además, este tipo de sistemas debe generar una vista previa con un alto grado de fidelidad con el resultado que será mostrado al usuario final. De esta forma el creador de la información puede ver rápidamente la representación del trabajo que ha realizado y cambiar en consecuencia el contenido necesario.
- Características tan básicas como deshacer, cortar y/o pegar pueden llegar a ser extremadamente útiles durante el diseño por parte del autor.
- El contenido generado deber ser modular y reusable.
- Elevada automatización para la creación de la información.
- Es necesario disponer de un sistema de ayuda al tutor.
- El material con el que se trabaja es independiente de la plataforma, almacenado en sitios locales o remotos.
- Para la creación de este tipo de *software* no es necesaria una conexión permanente a internet.

Las herramientas de autor más básicas son aquellas que solamente permiten un conjunto limitado de acciones para que el usuario interactúe con el sistema, como por ejemplo, navegar entre distintas páginas hipervinculadas o ir de una diapositiva a la siguiente. Ejemplos de estas herramientas pueden ser PowerPoint de Microsoft o Impress de Sun Microsystems. Sin embargo, las herramientas de autor más conocidas son las destinadas a la educación. Ejemplos de ello son:

- **Hot Potatoes**. Un programa con el que es posible crear seis tipos de ejercicios (opción múltiple, ordenación, asociación, rellenado de huecos…) acompañados de retroalimentación e integrando audio y vídeo [25].

- **JClic**. Se utiliza para realizar diferentes actividades educativas digitales: rompecabezas y ejercicios de asociación, entre otros [26].

- **Cuadernia**. Herramienta de creación de contenidos digitales educativos de la Consejería de Educación y Ciencia de Castilla La Mancha. Con ella se pueden crear recursos reutilizables, cuenta con un editor de cuadernos digitales, y una biblioteca de cuadernos para Infantil, Primaria y Secundaria [27].

- **Ardora**. Con esta sencilla aplicación multilingüe se pueden crear tanto actividades en formato HTML como páginas multimedia: crucigramas, sopas de letras, galerías de imágenes, etc [28].

- **Exe learning.** Herramienta para el diseño, edición y desarrollo de contenidos didácticos que posibilita su publicación en Internet así como su importación a plataformas de gestión de aprendizaje **(**LMS - *Learning Content Management System*), como, por ejemplo, Moodle o E-ducativa. Es una herramienta de fácil uso que permite crear contenidos con un diseño web atractivo [29].

- **DEDOS.** Sistema en el que se realizan actividades educativas destinadas a personas con síndrome de Down utilizando mesas multicontacto. En estos entornos se lleva a cabo un aprendizaje colaborativo, lo que fomenta enormemente el desarrollo cognitivo y social de los usuarios. Este proyecto se lleva a cabo en el Laboratorio de Inteligencia Ambiental (AmILab-UAM) y en el Laboratorio de Tecnologías de la Información en la Educación (LITE-URJC) [30].

Hoy en día podemos encontrar herramientas de autor más complejas, las cuales incluyen lenguajes de programación propios, como es el caso de Adobe Flash o de lenguajes de programación externos como el programa Dreamweaver. Además poseen ayudas para facilitar su manejo a usuarios sin conocimientos de programación.

- **Adobe Flash**. Esta aplicación permite la creación y manipulación de gráficos vectoriales (imagen digital formada por objetos geométricos independientes) con posibilidades de manejo de código mediante un lenguaje propio llamado ActionScript.

- **Dreamweaver**. Este programa está destinado a la construcción, diseño y edición de sitios, vídeos y aplicaciones Web basados en estándares. Utiliza lenguajes de programación como PHP, HTML así como hojas de estilo en cascada (CSS - *Cascading Style Sheets*).

2.2. TECNOLOGÍAS ASISTIVAS

En el entorno de la discapacidad, la idea inicialmente contenida en el concepto de Ayuda Tecnológica (AT), también conocida como "tecnología de adaptación" o "tecnología de ayuda", ha sido muy comúnmente identificada con el simple concepto de herramienta. Bajo esta perspectiva, la AT puede ser identificada con el utensilio o artefacto (objeto físico) diseñado para compensar una deficiencia o discapacidad. A este efecto, este objeto físico actuará, bien sustituyendo una función o capacidad sensorial carecida por el sujeto; bien potenciando los restos de la misma en orden a la realización de una tarea determinada. Por extensión, esta definición continuó siendo válida, aún después de la introducción de los recursos informáticos; sin más que centrar la consideración del *software* como un "artefacto virtual" de uso combinado con otro real (el *hardware*) [31].

Una definición más precisa de las tecnologías para la asistencia viene definida en el estándar ISO9999 [5]: "*tecnología asistiva es cualquier dispositivo, equipo, instrumento o software producido para prevenir, compensar, monitorizar, calmar o neutralizar*

discapacidades de las estructuras del cuerpo o su funcionalidad, restricciones en actividades o problemas en la participación social".

Esta norma hace un exhaustivo repaso sobre todos los productos de apoyo para personas con discapacidad así como su clasificación. Ésta incluye desde las prótesis para distintos miembros de cuerpo hasta el mobiliario y adaptaciones para viviendas y otros inmuebles. Este PFC se centra en el punto 22 de la clasificación por niveles que recoge este estándar, el cual versa sobre los productos de apoyo para la comunicación y la información. Más concretamente en el punto 22.33 ya que hace referencia al uso de ordenadores y terminales.

En la literatura autores como LoPresti et al. [32], definieron el término Tecnologías Asistivas para la Cognición (TAC - *Assistive Technologies for Cognition*) cuyo aspecto más relevante en este contexto es la relación entre la discapacidad cognitiva y la Interacción Persona-Ordenador. En este aspecto el autor habla sobre la importancia de adaptar la tecnología asistiva a usar por el usuario concreto con discapacidad ya que éste puede presentar diversas habilidades y limitaciones. Por todo ello, fija como requisitos que deben de cumplir las TAC para adaptarse a la discapacidad, la presentación de información en diversos modos: gráficos, vídeo, audio, etc. y de la forma más simple posible, como puede ser de forma secuencial. También hay que atender a otras limitaciones que pueden acompañar a la discapacidad ya que con frecuencia, este tipo de personas presentan limitaciones sensoriales y/o físicas (visión, capacidad auditiva, tacto, control motor fino como puede ser la capacidad de escritura, capacidad para hablar, coordinación). Por todo lo anteriormente relatado, estos autores señalan la importancia de la inclusión del usuario final en la etapa de diseño (conocido como modelo centrado en el usuario o diseño participativo).

En la revisión que se realiza en este PFC sobre las tecnologías para la asistencias nos encontramos con Braddock *et al*. [33]. Estos autores afirman que se producirá un aumento considerable de personas que sufran discapacidad cognitiva en los próximos años, y como resultado hay un creciente interés en el desarrollo y comercialización de nuevas tecnologías para las personas con este tipo de deficiencias. Este tipo de

tecnologías no sólo son de utilidad para las personas con discapacidad cognitiva sino también a aquellas personas que a causa de la edad se les produce un deterioro en sus facultades cognitivas. Por todo ello, este tipo de herramientas asistivas ayudan a lograr una mayor independencia, productividad y calidad de vida.

Para concluir, Braddock *et al*. realizan una clasificación de las herramientas en dos grupos, atendiendo al área tecnológica en el que se encuadran:

- Tecnologías para el apoyo personal, como son las PDAs, comunicación y aprendizaje asistidos por ordenador y Diseño Universal. El uso de las PDAs es especialmente importante ya que otorga al usuario ubicuidad cuando realiza las actividades así como movilidad.
- Sistemas de cuidado asistido. Entre ellos encontramos casas y sistemas de transporte inteligentes, robots personales y tecnologías de realidad virtual.

2.3. HERRAMIENTAS DE AUTOR EN LAS TECNOLOGÍAS ASISTIVAS

Uno de los autores más relevantes en el campo de las tecnologías asistivas es Stefan Carmien. Este investigador señala que los individuos que poseen discapacidad cognitiva están a menudo incapacitados para vivir de forma autónoma debido a las deficiencias de memoria, de atención y dificultad para ejecutar funciones. En concreto, algunos pacientes presentan problemas a la hora de realizar actividades de la vida diaria tales como cocinar, planchar, hacer la compra en el supermercado, etc [3]. Por ello, Carmien propone un sistema de mensajes diseñado para ayudar a estas personas en sus actividades de la vida cotidiana llamado MAPS (*Memory Aiding Prompting System*) [34] y [35]. Esta herramienta dispone de dos partes: la interfaz del paciente que se mostrará en un dispositivo móvil y la herramienta de autor destinada a los terapeutas (Figura 2.1).

La primera parte, la interfaz para el paciente, se ha diseñado pensando en sus necesidades, adaptándose tanto a ellas como a las limitaciones que presenta el dispositivo móvil.

Figura 2.1: Dispositivo móvil del paciente (en la parte izquierda de la imagen) y la herramienta de autor (en la parte derecha de la imagen).

Por otro lado, la herramienta de autor permite a los cuidadores o terapeutas describir los pasos a seguir para completar las tareas de forma sencilla y muy visual. Para la creación de este *software* el autor recalca la importancia de hacer un programa para el cual los terapeutas necesiten unos conocimientos informáticos similares a los que se necesitan para mandar un correo electrónico o escribir un sencillo texto en Microsoft Word. Esta idea se tendrá muy en cuenta a la hora de crear la herramienta de autor que motiva este PFC.

El *software* de autor del proyecto MAPS proporciona al cuidador todas las herramientas necesarias para crear, anotar, modificar el contenido de las tareas y ver la vista previa del material creado. Para ello, una vez seleccionada la tarea que el terapeuta quiera crear, hay que dividirla en segmentos sencillos (pasos), pensando en las personas con discapacidad que serán a los que se les muestre estos segmentos. Una vez realizada la división, se seleccionará mediante la herramienta de autor, las imágenes y locuciones verbales que se quiera mostrar en cada uno de los pasos de la tarea. Finalmente, una vez creada toda la tarea, ésta se almacena en el servidor para que el paciente ya pueda verla desde el dispositivo móvil. Otro de los aspectos en los

que enfatiza Stefan Carmien es el concepto de reutilización de las tareas ya existentes en el sistema, por parte del terapeuta, para crear otras nuevas, lo que provoca un ahorro de tiempo en el diseño.

Otra aplicación dentro de las tecnologías asistivas es el Gestor de Tiempos y Tareas (Time and Task Manager-GTT), desarrollado por Alberto Ferreras *et al*. en el Instituto de Biomecánica de Valencia en el año 2010 [36], para favorecer la autonomía en el ámbito laboral. Según un estudio de la asociación FEAPS (Confederación Española de Organizaciones En Favor de Personas Con Discapacidad Intelectual) sobre el empleo de las personas con discapacidad intelectual, se calcula que más de un 60% de las personas que tienen alguna discapacidad intelectual y que están en condiciones de trabajar se encuentra en situación de desempleo.

Muchos de los problemas que este sector de población presenta hacen referencia a aspectos como el control de tiempos, la independencia en la realización de tareas, los hábitos de trabajo o la comunicación interpersonal. Por todo lo relatado, GTT persigue mejorar la calidad de vida de las personas con discapacidad intelectual, facilitándoles la integración sociolaboral.

Figura 2.2: Interfaz de la aplicación GTT.

El Gestor de Tiempos y Tareas (GTT), al igual que la aplicación de Carmien (MAPS), se divide en dos plataformas:

- Una aplicación de escritorio (herramienta de autor), que es un programa de ordenador que permite crear y administrar las diferentes prestaciones y contenidos que posteriormente se le facilitarán al trabajador.

- Una aplicación portátil para PDA (Pocket PC) y teléfonos móviles táctiles que usará los ficheros generados en la aplicación de escritorio. Ésta es la aplicación que usará el trabajador.

En ambos casos, esta aplicación incluye tres apartados:

- "Cómo se hace" ofrece un tutorial de las distintas tareas que tiene que realizar el trabajador. La PDA muestra paso a paso cómo se tiene que hacer una tarea determinada. Para ello utiliza una combinación de texto, imágenes y audio. El terapeuta puede diseñar la estructura de tareas y subtareas más apropiada para un trabajador concreto mediante el uso de la herramienta de autor. Este módulo también puede usarse como comunicador personal, componiendo mensajes que combinen imágenes y audio.
- "Para hoy" consiste básicamente en una agenda con los eventos más importantes que el usuario debe recordar. A la hora del evento salta una alarma que avisa al usuario de dicho evento mediante texto, imagen o audio. Para ello, la persona encargada del trabajador deberá introducir en el *software* de autor todos los ítems.
- "Informa-T" es un módulo en el que pueden realizarse actividades adicionales (presentaciones y cuestionarios) que refuercen aspectos laborales como el conocimiento del entorno, el uso de herramientas o la prevención de riesgos laborales, entre otros. Este módulo, como en los dos casos anteriores, es gestionado por el tutor del trabajador, usando la herramienta de autor proporcionada.

Otro sistema que opera dentro de las tecnologías asistivas es el proyecto llamado PREVIRNEC para realizar tele-rehabilitación cognitiva basada en entornos virtuales [37]. En España se calcula que cada año se producen 80.000 nuevos casos de traumatismos craneoencefálicos cuya principal causa son los accidentes de tráfico. La necesidad de rehabilitación surge al considerar las secuelas cognitivas a largo plazo, y su impacto en la adaptación funcional, la independencia y su calidad de vida de la persona afectada por el daño cerebral y de su familia. Sin embargo este tipo de rehabilitación es un proceso costoso y los recursos que se pueden dedicar son limitados (normalmente son terapeutas los que se ocupan de hacer la recuperación de

estas personas en pequeños espacios de tiempo). Por ello, surge la telerehabilitación entendida como "*servicios de rehabilitación por medio de las tecnologías de la información para la intervención y soporte a distancia de personas con discapacidad*". Este tipo de rehabilitación tiene como ventajas la accesibilidad, aplicación de programas flexibles al usuario y reducción del coste económico de los tratamientos.

La aplicación está basada en un esquema modelo vista-controlador en Web. En este esquema volvemos a tener una clara división:

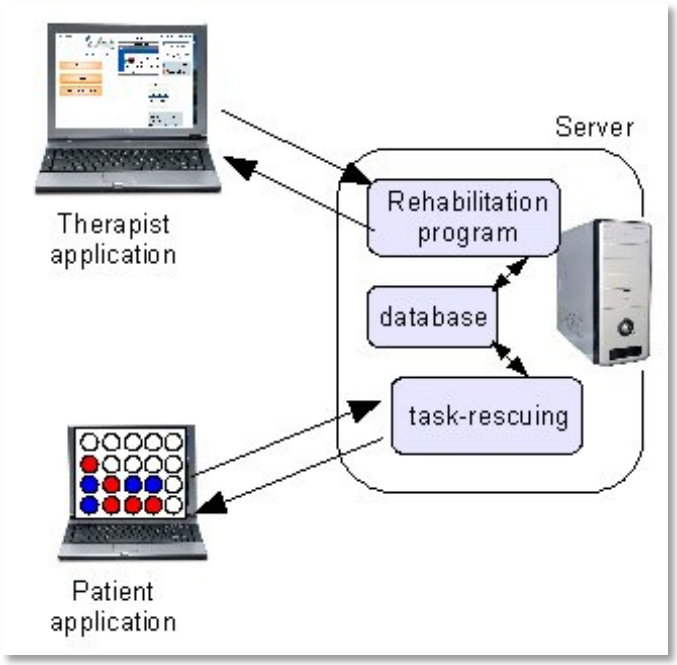


Figura 2.3: Esquema de la aplicación PREVIRNEC.

- Por un lado, tenemos a los neuropisicólogos que diseñan los programas de rehabilitación por medio de la herramienta de autor que el sistema les proporciona. Éstos pueden consultar la base de datos de cada paciente. En ella se encuentra información acerca del historial, de la rehabilitación hecha, así como los resultados alcanzados. En función de ello, los profesionales diseñan los ejercicios atendiendo al objetivo específico de la actividad (pueden ser para la memoria, para la atención o para la ejecución de funciones). Por tanto, por cada paciente, los neuropisicólogos definen una lista de ejercicios para realizar por el paciente (personalización que se considerará a la hora de realizar la herramienta de autor del sistema aQRdate), clasificados por el nivel de dificultad.

- Por otro lado tenemos a los pacientes que se conectan al servidor en el que se encuentran los planes de rehabilitación y realizan sus ejercicios correspondientes.

PREVIRNEC se concibe, por tanto, como una herramienta para intensificar la rehabilitación cognitiva, individualizar los planes terapéuticos y monitorizar la evolución del paciente a lo largo del tratamiento; y todo ello de un modo sostenible.

Por último, otra herramienta para la asistencia de personas con discapacidad cognitiva podría ser el programa realizado en la Universidad Autónoma de Madrid y la Universidad Carlos III de Madrid llamado PAIR (*Personal Ambient Intelligent Reminder*) del año 2012 [38]. Este sistema sirve para ayudar a los cuidadores de personas con trastorno de memoria, en un entorno inteligente, permitiendo definir actividades cotidianas que el usuario debe realizar cada día, así como recordarle las que no se hayan realizado en el momento adecuado avisando al cuidador.

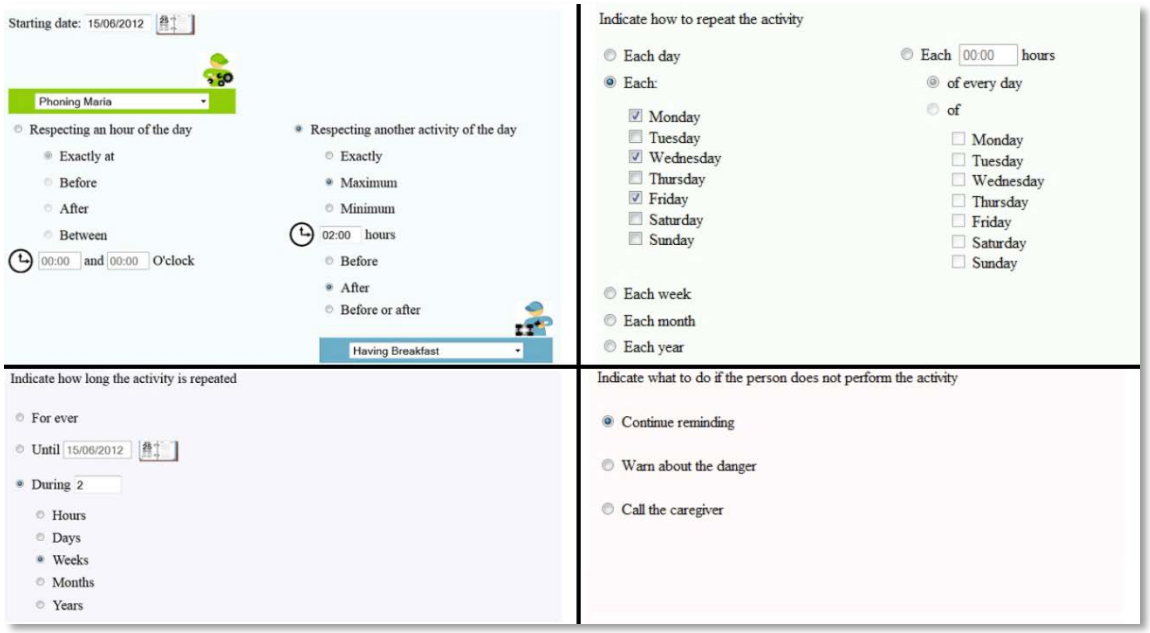


Figura 2.4: Herramienta de autor PAIR destinado a los terapeutas o tutores.

Para poder usar este sistema de recordatorios, previamente el terapeuta o tutor, debería haber introducido por medio del *software* de autor proporcionado en Web, la lista de actividades que el paciente tiene que realizar así como la hora de comienzo y su duración.

2.4. CONCLUSIONES

En este capítulo se han analizado las herramientas de autor, definidas para que los tutores, cuidadores o terapeutas puedan diseñar contenido multimedia (imagen, vídeo, audio, etc.) con conocimientos informáticos básicos.

Autores como Murray [21] destacan como objetivos a tener en cuenta para este tipo de *software*: reducir el esfuerzo a la hora de generar el material multimedia ya sea en tiempo, coste y/u otros recursos, crear un programa que pueda ser usado por personas con conocimientos informáticos básicos, ayudar al usuario (autor) a organizar los contenidos, promover los principios para un buen diseño mediante recomendaciones y/o imposiciones y finalmente desarrollar una herramienta de autor que permita un diseño de la información de una forma rápida con el objetivo de poder evaluarla y por tanto, realizar los cambios oportunos en el diseño.

Además, estos programas han de tener una edición sencilla, del tipo WYSIWYG (*What You See Is What You Get*). Deben generar una vista previa con un alto grado de fidelidad con el resultado que será mostrado al usuario final de esta información. Características tan básicas como deshacer, cortar y/o pegar pueden llegar a ser extremadamente útiles durante el diseño por parte del autor. El contenido generado deber ser modular y reusable con una elevada automatización para la creación de la información. Es necesario proveer a los autores de un sistema de ayuda. El material con el que se trabaja es independiente de la plataforma, almacenado en sitios locales o remotos. Por último se destaca que para la creación de este tipo de *software* no es necesaria una conexión permanente a internet.

En este estado del arte se han presentado desde las herramientas de autor más sencillas como puede ser PowerPoint hasta las más complejas que incluyen la posibilidad de usar lenguajes de programación, como es el caso de Adobe Flash. Sin embargo, las herramientas de autor más conocidas son las que se utilizan en el ámbito de la educación. Ejemplos de ello son: Hot Potatoes, JClick, Cuadernia, Ardora y Exe learning.

Otro aspecto destacado dentro de esta revisión bibliográfica es la tecnología asistiva, marco en el que se encuentra encuadrado este PFC [31]. Son tecnologías cuyo fin es prevenir, compensar, monitorizar, aliviar, calmar o neutralizar discapacidades de las estructuras del cuerpo o su funcionalidad, restricciones en actividades y problemas en la participación social mediante el uso de dispositivos, equipos, instrumentos, tecnología o *software*. LoPresti *et al*. [32] fijan como requisitos que deben de cumplir las TAC para adaptarse a la discapacidad, la presentación de información en diversos modos: gráficos, vídeo, audio, etc. y de la forma más simple posible como puede ser de forma secuencial. También hay que atender a otras limitaciones que pueden acompañar a la discapacidad, como las limitaciones sensoriales y/o físicas. Estos autores señalan la importancia de la inclusión del usuario final en la etapa de diseño.

Por último, en este capítulo se muestran cuatro sistemas de ayuda a personas con discapacidad cognitiva:

- MAPS destinado a la realización de tareas diarias [34] y [35].
- GTT que intenta mejorar la calidad de vida de las personas con discapacidad intelectual, facilitándoles la integración sociolaboral [36].
- PREVIRNEC pensado para realizar telerehabilitación cognitiva [37].
- PAIR que permite definir actividades cotidianas que el usuario debe realizar cada día y recordarle las que no se hayan realizado en el momento adecuado avisando al cuidador [38].

De todos estos sistemas las ideas principales que se pueden destacar para tener en cuenta a la hora de realizar la herramienta de autor de aQRdate son:

- Diseñar un programa sencillo, de tal forma que los conocimientos previos que se necesiten para uso sean mínimos.
- Además, se debe proporcionar al cuidador todas las herramientas necesarias para crear y modificar el contenido de las tareas.
- El sistema debe poder adaptar las tareas a las necesidades del usuario final.
- Otro de los aspectos a tener en cuenta es el concepto de reutilización de las tareas ya existentes en el sistema con el fin de ahorrar tiempo en el diseño.

3

aQRdate: APLICACIÓN MÓVIL

El sistema aQRdate, fue desarrollado como un Trabajo Fin de Máster [6] dentro del Laboratorio de Inteligencia Ambiental (AmILab) de la Universidad Autónoma de Madrid en estrecha colaboración con el Centro de Referencia Estatal en Atención al Daño Cerebral, CEADAC.

El objetivo de este TFdM fue la creación de una aplicación móvil destinada a usuarios que padecen daño cerebral adquirido, buscando sustituir la clásica sesión de rehabilitación en la que el paciente realiza actividades cotidianas de diferentes complejidades mientras es observado por su tutor. Por tanto, el uso de este programa pretende estar integrado en su actividad diaria, si bien es cierto que el fin último consiste en que el paciente obtenga unas destrezas mediante su uso que le permitan realizar las tareas diarias sin el soporte y guiado que proporciona la aplicación.

Esta aplicación escanea códigos QR que se encuentran en diferentes elementos del entorno y, a partir de ellos, se muestra de forma automática un manual adaptado al elemento, el usuario y sus necesidades, y el dispositivo. El hecho de que el usuario sólo tenga que enfocar con la cámara del teléfono al elemento y que, automáticamente, se muestre el manual es una solución muy útil para aquellos usuarios que encuentran dificultades para seleccionar un elemento de una lista, por no poder leer, por ejemplo.

El manual a mostrar cada vez que se escanea un código QR, puede constar de instrucciones específicas (por ejemplo, las instrucciones para hacer la colada) o información genérica (por ejemplo, la información asociada a la puerta de casa puede recordar las cosas que hay coger antes de salir).

Además, estos manuales (el conjunto de instrucciones que lo componen) se pueden adaptar a las necesidades del usuario, añadiendo o eliminando pasos de forma que se varíe la cantidad de ayuda conforme avanza en su rehabilitación.

3.1. REQUISITOS DEL SISTEMA

Los usuarios con daño cerebral adquirido usarán el sistema durante su proceso de rehabilitación, tanto en su domicilio como en el CEADAC. Esta aplicación se centra en la asistencia de actividades de la vida diaria, dada su importancia dentro del proceso de rehabilitación.

Los requisitos establecidos son:

- **Accesibilidad móvil**: Se trata de un sistema que da soporte a actividades de la vida diaria, luego éstas se pueden llevar a cabo en cualquier momento y lugar, siempre y cuando disponga de conexión a la red. Además, en el dispositivo móvil se deberá presentar toda la información necesaria para que el usuario sepa completar la tarea, sin tener que desplazarse a algún sitio para obtener la información: le bastará con las instrucciones que le proporciona el dispositivo que lleva consigo.
- **Presentación de la información**: Tanto la interfaz de usuario como la interacción deben adaptarse a las necesidades del usuario y las restricciones del dispositivo. Para esta adaptación, entre otras medidas, se debe dar la posibilidad de incluir información multimodal (imágenes, texto, vídeos, audios, voz, etc.).
- **Simplicidad**: Se debe de perseguir en todo momento la simplificación de la interfaz de usuario manteniendo aquellos elementos que sean estrictamente necesarios para la realización de las tareas.
- **Modelo centrado en el usuario o diseño participativo**: Como se ha comentado en el capítulo 1.2 se debe incluir al usuario en el proceso de diseño. En este caso, se incluye al equipo médico del CEADAC, en calidad de expertos en las necesidades de los pacientes.

3.2. ARQUITECTURA PROPUESTA

Como se ha descrito en el apartado anterior, es necesario que se utilice un dispositivo capaz de acceder a la información en cualquier momento y en cualquier lugar (acceso ubicuo). Por estas características se podría pensar en un ordenador portátil pero se escogió como medio el teléfono móvil pues es un dispositivo de un tamaño menor, muy extendido en la sociedad actual, con suficiente potencia tanto de cálculo como de comunicación (3G, LTE y Wi-Fi) y con una duración de la batería mayor.

Para realizar la arquitectura del sistema, en primer lugar, se pensó en almacenar toda la información asociada a las tareas en el teléfono móvil y de esta forma no habría consumo de datos pues no se necesitaría la conexión a Internet. Sin embargo, esta implementación presenta una clara desventaja relacionado con el versionado de las actividades, ya que cada vez que el personal médico realizase cualquier cambio (añadir, modificar o suprimir información de cualquier actividad), deberían actualizar la versión del programa en cada uno de los dispositivos móviles de los usuarios. Por esta razón se decidió centralizar toda la información en un componente al que accedan todos los terminales móviles mediante conexión a internet: arquitectura cliente-servidor.

Con esta arquitectura, los terapeutas sólo tendrían que acceder a la información almacenada en el servidor cuando quisieran realizar cualquier modificación y los usuarios (clientes) recibirían la nueva versión la siguiente vez que ejecutasen la actividad.

3.3. EL SERVIDOR

El desarrollo de este TFdM se llevó a cabo dentro el Laboratorio de Inteligencia Ambiental AmILab. Dentro de este laboratorio se encuentra desarrollada una capa intermedia o *middleware* para entornos inteligentes soportada por ontologías [39]. Este sistema recibe el nombre de "*The Blackboard*" (la Pizarra), ya que está basado en la metáfora de la pizarra [40].

En la Pizarra se modelan objetos del entorno, cuya estructura se describe en un esquema. Este esquema contiene la descripción del entorno, en término de clases, propiedades y capacidades y las relaciones que puedan aparecer entre ellas, es decir, un modelo ontológico. Los objetos que se modelan de acuerdo a estos esquemas se conocen como entidades, y se almacenan en el repositorio de manera que toda la información es accesible desde la estructura global de la Pizarra. Las entidades pueden representar tanto objetos físicos (por ejemplo ordenadores o personas), como objetos virtuales (por ejemplo imágenes o información personal).

Por tanto, se decidió modelar las actividades de la vida diaria como entidades de la Pizarra (tratadas como objetos virtuales). De esta forma toda la información relativa a las actividades estaría controlada por la Pizarra y, por ello, accesible al personal médico para que realicen las modificaciones necesarias de una forma sencilla.

El esquema que utiliza aQRdate es el siguiente:

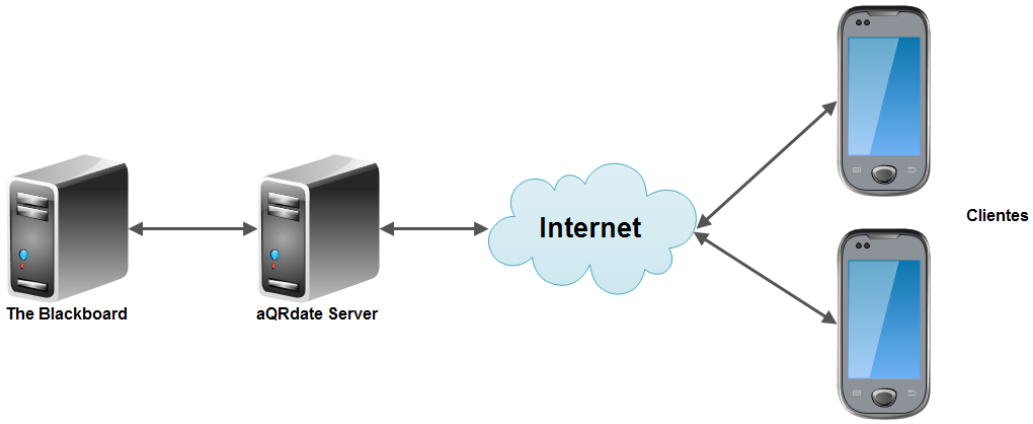


Figura 3.1: Arquitectura del sistema.

Como se puede ver en la Figura 3.1, entre la Pizarra y los clientes se introdujo un servidor intermedio; cuyo objetivo es reducir la cantidad de tráfico que se genera, lo que puede suponer altos costes en la conexión. Esto se debe a que se requiere varias conexiones para recuperar la información asociada a cada uno de los pasos de la actividad así como las relaciones de éstos con el resto (paso anterior y paso siguiente). Para solventar este problema se incluyó “aQRdate Server” en la arquitectura, entre la Pizarra y el cliente. Este servidor, al arrancar, pregunta a la pizarra por todas las actividades que tiene almacenadas y genera los árboles que representan las secuencias de pasos de cada actividad así como la relación entre ellos.

En resumen, podemos dividir el sistema en tres partes: la Pizarra que almacena la descripción de todas las actividades; aQRdate Server que resuelve las relaciones y genera los árboles; y los clientes a los cuales se les muestra en el teléfono móvil los pasos a realizar en cada tarea.

3.4. DESCRIPCIÓN DE LAS ACTIVIDADES

Para utilizar el sistema de rehabilitación propuesto, se deben de definir previamente las actividades y pasos que la componen. La descripción de las actividades en el esquema de la Pizarra se llevó a cabo a través de la sintaxis descrita en [39] (Anexo A). Para ello, se definen "Routines" (rutinas, que comprenden un activividad completa), "Tasks" (tareas, que son los pasos a seguir para completar una actividad y por tanto es el contenido que se le muestra al usuario) y "Fragments" (fragmentos, los cuales contienen porciones de información relacionado con las tareas).

Un ejemplo de representación de una actividad (Routine) en forma de árbol es:

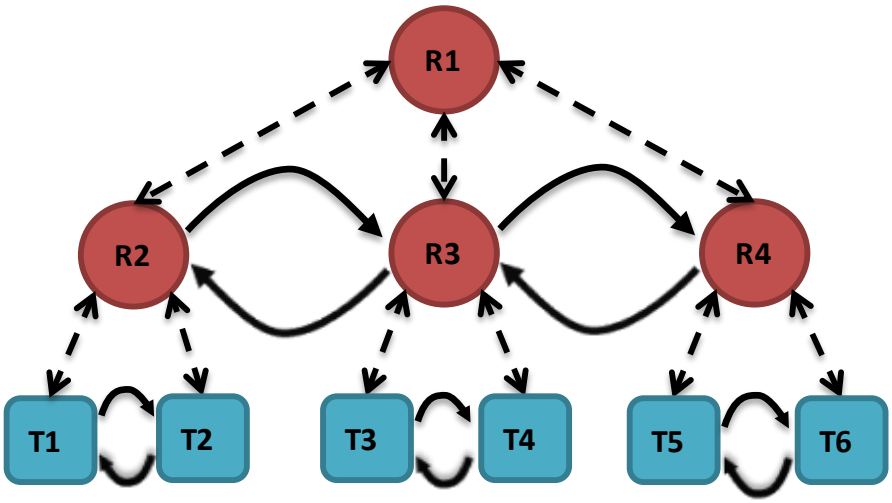


Figura 3.2: Ejemplo de Routine en forma de árbol.

En este esquema podemos ver que hay una actividad (Routine) R1 compuesta por tres subrutinas: R2, R3 y R4. Cada rutina, en este caso, tiene dos tareas (Tasks) cada una. El árbol completo constaría, a su vez de los dos fragmentos (Fragments) que componen las tareas, pero en el ejemplo se han omitido para simplificar la explicación ya que no son demasiado relevantes y harían la imagen menos clara. Las líneas discontinuas representan el parentesco (nodos padre-hijo) y las líneas continuas simbolizan la secuenciación (paso anterior y paso siguiente).

Para mostrar los pasos (tareas) en el móvil del usuario, se realiza una búsqueda en profundidad siguiendo el orden en que están dispuestas, en preorden [41]. Para este ejemplo (Figura 3.2) la aplicación haría el recorrido R1-R2-T1, obteniendo la primera tarea de la secuencia. La rutina R2 tiene dos tareas, luego la siguiente tarea es T2. A continuación se continúa recorriendo el árbol de la siguiente manera:

1. Comprueba si T2 tiene una tarea siguiente: no la tiene.
2. Obtiene por tanto el padre de la última tarea (T2): es el nodo R2.
3. Comprueba si R2 tiene una relación "siguiente": Sí la tiene, es R3.
4. Como R3 es una rutina, obtiene las tareas que la componen:
    1) Para obtener las tareas que tiene, mira todas sus relaciones "está compuesta de".
    2) Una vez encontradas todas estas relaciones, busca aquella tarea que no tenga una relación "anterior" puesto que es el primer paso de la subrutina: T3.
5. Por tanto, la siguiente tarea a T2 es T3.

Recorriendo el árbol completo siguiendo procedimiento explicado, obtendríamos que la secuencia de tareas que se le mostraría al usuario sería: T1-T2-T3-T4-T5-T6. La descripción completa del código generado para la creación del árbol mostrado en la Figura 3.2 se encuentra en el Anexo B.1.

Como se ha visto en la explicación del recorrido del árbol, se realizan varias preguntas a la Pizarra para obtener una tarea. Por ello, como se ha explicado en el apartado anterior, debido a la gran cantidad de tráfico que estas consultas generan, se situó un servidor intermedio en el sistema (llamado aQRdate Server). Este servidor construye todo el árbol de la actividad desde que el usuario ejecuta la aplicación y esto hace que todas las preguntas se reduzcan a una sola: pedir la siguiente tarea a la mostrada actualmente en el móvil (en el ejemplo anterior simplemente preguntaría por la tarea siguiente a T2).

A continuación se explica con más detalle los tres tipos de elementos que utiliza la Pizarra para realizar las actividades (Routine, Task y Fragment), las relaciones entre ellas y las capacidades que tiene:

Routine

Engloba el conjunto de pasos que los usuarios deben realizar para completar una actividad. Pueden ser representadas mediante códigos QR para que con el teléfono móvil sean escaneados por los usuarios y se les muestre los pasos que la componen. Además, una Routine (actividad) puede estar compuesta por varios pasos (Task) o por otras Routine (subactividades). Por tanto, cada Routine tiene las siguientes relaciones:

- hasAction: Representa la unión padre-hijo entre una Routine y otra Routine o Task.
- composesRoutine: Es la relación inversa a hasAction. Es la unión hijo-padre.
- PreviousAction, nextAction: Indican el paso siguiente o anterior (es decir, una Routine o Task).

El nombre de la actividad que será mostrado al usuario se almacena como una propiedad de esta entidad llamada “descriptedName”.

Task

Representa el contenido de cada paso que se muestra en la aplicación del usuario para completar la actividad. Mediante la relación “composesRoutine” se define la actividad a la que pertenece. Cada Task se divide en dos fragmentos: el primero de ellos es un título y el segundo puede tener información de tipo imagen, vídeo, audio o texto. Esta división se establece mediante relaciones “hasFragment”. Al igual que las Routines, cada Task puede ir precedida o seguida de otros pasos (Task o Routine). Por ello, como en el caso de las Routines, se emplean las relaciones “previousAction” y “nextAction”.Además las tareas pueden tener dos tipos de capacidades:

- hasToBeRepeated: Define si la tarea se va a repetir.
- isMultipleChoiceAction: Sirve para definir varias opciones en esta tarea. El usuario elegirá una para continuar con la actividad.

Estas capacidades se explicarán con más detalle en la sección 3.6 de este capítulo.

Fragment

Contiene la porción de información relacionada con un paso (Task) de la actividad. Para indicar la tarea a la que pertenece se utiliza la relación “composesTask”.

Tiene dos propiedades estrechamente relacionadas:

- fragmentType: Define la naturaleza de la información. Puede ser de tipo título, texto, imagen, vídeo o audio.
- framentContent: Es el contenido en sí. Es decir, las imágenes, vídeos, audios, títulos (nombre de la tarea) o textos (descripción textual de la tarea que el usuario tiene que realizar).

Relaciones

Como se acaba de ver, se han modelado los nexos entre los diferentes elementos para establecer relaciones. Estas relaciones tienen asociado el nombre de usuario o en su defecto el "usuario genérico" y por ello permiten la adaptación de las secuencias a las necesidades de los diferentes usuarios como se verá en la sección 3.5 (siguiente sección).

3.5. ADAPTACIÓN AL USUARIO

Diferentes usuarios pueden requerir diferentes conjuntos de instrucciones para realizar una misma actividad. Por ello, el sistema aQRdate tiene desarrollado un mecanismo que permite realizar esta adaptación. Una vez que el terapeuta ha definido una actividad al completo, puede establecer nuevas relaciones personalizando la secuencia a las necesidades de los usuarios.

Para explicar con mayor claridad la opción de adaptación que tiene el sistema, se muestra el siguiente ejemplo:

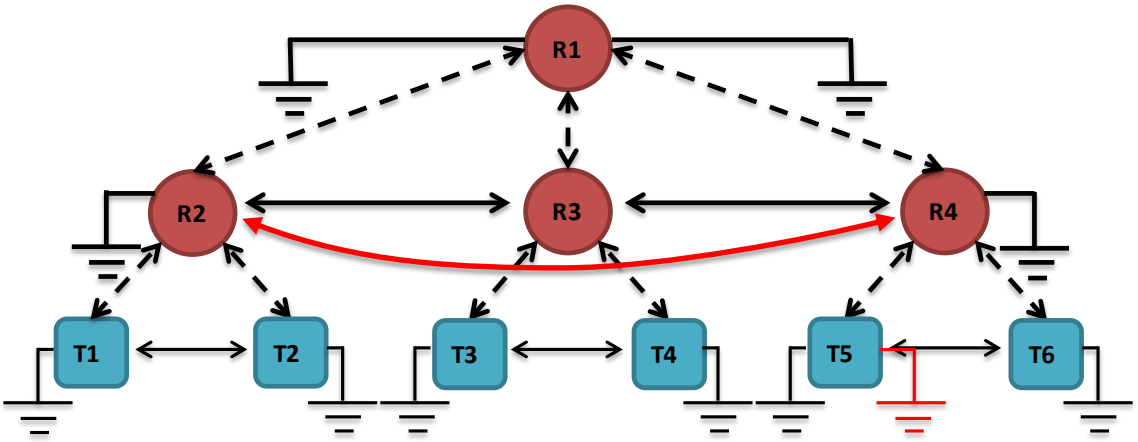


Figura 3.3: Ejemplo de adaptación de la actividad.

En esta figura se ha añadido el símbolo de conexión a masa electrónica para indicar que esa actividad o tarea no tiene paso anterior o paso siguiente. Con las líneas discontinuas se señala el parentesco y las líneas continuas muestran las relaciones de secuenciación (anterior y siguiente). Por otro lado, las líneas de color negro corresponden con las relaciones de un usuario genérico y las de color rojo son para un usuario concreto al que se le ha adaptado la secuencia.

Al igual que en el ejemplo de la Figura 3.2, se realiza el recorrido de la Routine R1 en preorden [41]. Si se simulase la ejecución entera del ejemplo, se obtendría la siguiente secuencia: R1 – R2 – T1 – T2 – R3 – T3 – T4 – R4 – T5 – T6. Si se tiene en cuenta que al usuario sólo se le mostrarán los elementos de tipo Task, la secuencia resultante sería: T1 – T2 – T3 – T4 – T5 – T6.

Por otro lado, una vez adaptada la actividad al usuario, habría que recorrer el árbol de la misma forma pero, en caso de existir relaciones específicas para ese usuario, seguirlas (en la figura, serían las líneas rojas). De esta forma, obtendríamos T1 – T2 – T5, subconjunto de la secuencia genérica.

Para crear la adaptación de secuencias, se han creado nuevas relaciones (en las Routines R2 y R4 y en la Task T5) para un cierto usuario (llamado en este ejemplo “usuario adaptado”). La descripción completa de la figura anterior se encuentra en el Anexo B.2 (en este código, el nombre de “usuario adaptado” se ha abreviado a “UA”).

3.6. CONTROL DE FLUJO DE LA SECUENCIA DE TAREAS

Generar interfaces ubicuas para un conjunto de tareas cuya secuencia de pasos es necesariamente lineal supone, en muchas ocasiones, un apoyo insuficiente. Normalmente, cualquier tarea de la vida cotidiana comprende hacer tareas que varían dependiendo de detalles concretos sobre lo que se desea hacer (por ejemplo, el uso de una fotocopiadora varía dependiendo de si el usuario desea hacer copias a una o a dos caras). Debido a ello, se realizó un Trabajo Fin de Grado (TFG) por parte de Juan Carlos Torrado Vidal [42], en el que se introdujo al sistema aQRdate dos nuevas opciones: tareas repetitivas y tareas de selección múltiple.

3.6.1. Tareas Repetitivas

La necesidad de introducir tareas repetitivas radica en la ventaja que supone el guiado de tipo mecánico para este tipo de personas. No hay que olvidar que se trata de una herramienta de aprendizaje, por lo que es de interés que las actividades que comprendan tareas que hay que repetir varias veces consecutivas durante la secuencia sean descritas al paciente como tales, en lugar de ofrecer una única tarea que dé esta información de forma descriptiva (esto es, “pulsa cierto botón" cuatro veces de forma consecutiva, en lugar de “pulsa el botón cuatro veces", y pasar a la siguiente tarea).
Además en este tipo de tareas, se muestra en la interfaz un indicador de progreso (este progreso es mostrado en forma de número de repeticiones que quedan por realizar).

El número de repeticiones que puede ir asociado a una tarea es, generalmente, definido por el tutor cuando diseña la actividad. Sin embargo, el sistema también permite que el tutor decida durante la planificación que se deje esa decisión en manos del usuario. Esto significa, que al llegar al paso en cuestión, la aplicación móvil preguntará al usuario cuántas veces desea repetir la tarea que acaba de realizar.

Como se comentaba en la sección 3.4, esta funcionalidad se traduce en una capacidad “hasToBeRepeated”, con una propiedad “nTimes” de tipo numérico, en la que se introduce el número de repeticiones que se desean hacer (si se desea que las elija el usuario, esta propiedad deberá tener el valor *-1*).

3.6.2. Tareas de Selección Múltiple (Bifurcaciones)

Muchas actividades que se realizan en la vida cotidiana son objeto de decisiones que se van tomando durante la realización de las mismas (por ejemplo, siendo una actividad hacer la colada, la secuencia de pasos varía dependiendo de si se decidiese lavar ropa blanca, lavar ropa de color o lavar ropa delicada).

El terapeuta, con aQRdate, puede planificar para el usuario tareas de selección múltiple, en las que habrá pasos de la secuencia en los que se muestren al usuario posibles opciones a escoger, variando las tareas sucesivas en función de la opción escogida (en el ejemplo que se comentaba anteriormente, la actividad global será

hacer la colada, y consistirá en una serie de pasos, entre ellos uno del tipo escoger el modo de lavado, en el que se mostrarán las opciones ropa blanca, ropa de color y ropa delicada, variando la sub-secuencia sucesiva según la opción).

Esta opción, en forma de árbol, quedaría representada de la siguiente manera:

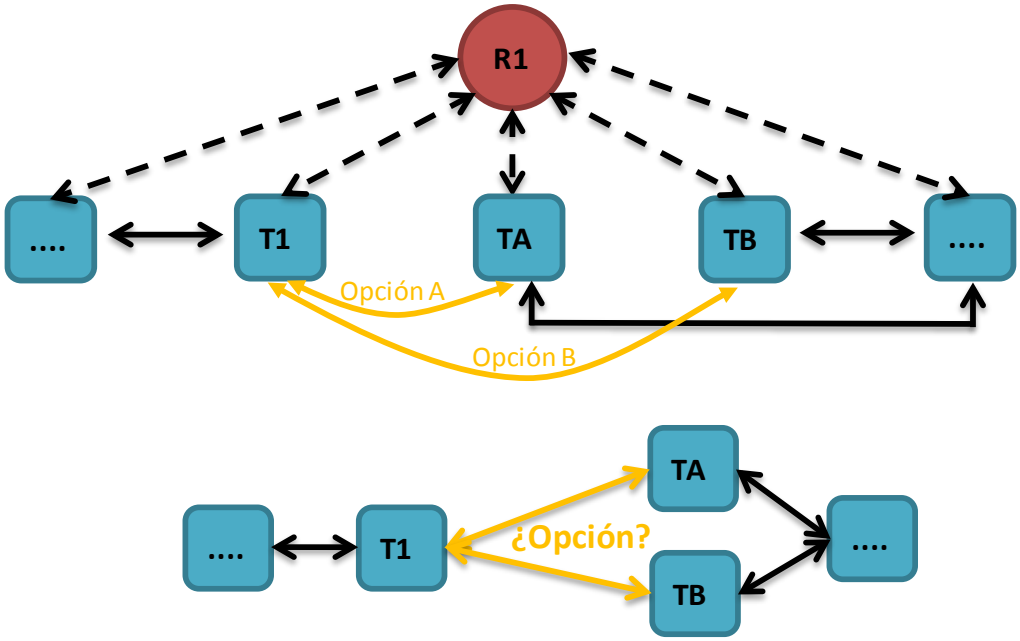


Figura 3.4: Ejemplo de tarea de selección múltiple.

En este ejemplo, una vez alcanzada la Tarea (Task) T1, el usuario tendría que elegir la opción que decide escoger: opción A (TA) u opción B (TB). Una vez que el usuario ha seleccionado la opción, se le muestra ésta y continúa con la secuencia habitual.

Todas las opciones que puede tener una tarea se encuentran recogidas en una propiedad multivaluada dentro de la capacidad mencionada en la sección 3.4; llamada "isMultipleChoiceAction". Estas opciones son las que se despliegan ante el usuario una vez alcanzada la tarea que contiene selección múltiple. A continuación, cuando se selecciona la opción, se envía al servidor y éste busca qué tarea del árbol debe ofrecer a continuación (aquella referenciada por la relación "nextAction" que contenga la propiedad "forOption" con el valor de la opción escogida por el usuario). En el apartado 3.8 se muestra un ejemplo de este tipo de tareas cuyo código completo se encuentra en el anexo B.3.

3.7. INTERFAZ DE USUARIO

Como se ha comentado al inicio de este capítulo, los usuarios ejecutarán esta aplicación en teléfonos móviles, gracias a ello pueden acceder a la información de forma ubicua. Se optó por teléfonos móviles equipados con sistema operativo Android[4].

Uno de los objetivos principales del sistema aQRdate era adaptar tanto la interacción como la interfaz a los usuarios a los que va destinado esta aplicación. Con respecto a la interacción se puede decir que se ha reducido al máximo la intervención del usuario. Una de las medidas tomadas es que para seleccionar la actividad, en lugar de buscar en un listado o escribir el nombre, tan solo tiene que apuntar con el teléfono al código QR correspondiente a la actividad que quiera realizar. La adaptación de la interfaz a las necesidades del usuario se ha tenido en cuenta tratando de seguir líneas minimalistas y que la información quedase lo más clara posible. Además la información tiene contenido multimedia (imágenes, vídeos, etc.) y la lectura en voz alta de las órdenes.

De acuerdo con las sugerencias del equipo médico del CEADAC, para cada una de las tareas que componen una actividad, se diseñó la siguiente interfaz:

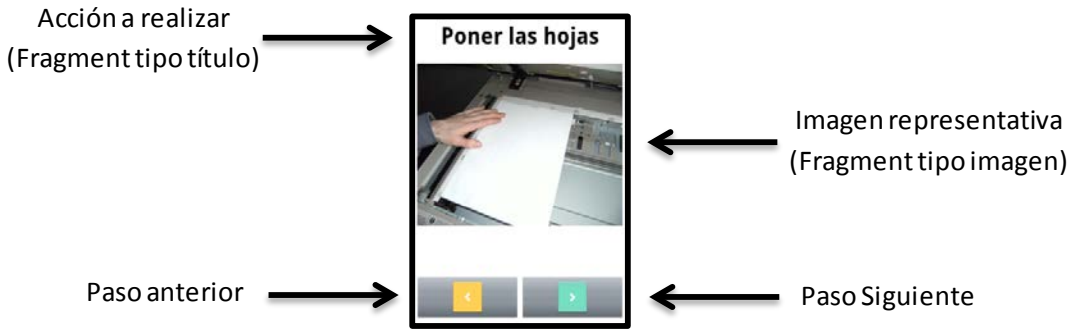


Figura 3.5: Ejemplo de la interfaz del teléfono móvil.

Esta captura corresponde con una actividad real (realizar fotocopias). Los elementos que componen esta interfaz son:

- La acción a realizar es un Fragment de tipo "título" de la tarea. Como se ha comentado anteriormente, la aplicación puede configurarse para que este texto se lea al cargar la tarea.

[4] Para más información ver: http://www.android.com

- En este caso, el terapeuta ha añadido un Fragment de tipo imagen (podría haber sido un vídeo, un audio o una descripción textual), que ofrece un fuerte apoyo visual ya que ayuda al usuario a entender la orden que se le está mostrando o a identificar el objeto a usar.
- Por último se muestran los botones para navegar hacia el paso anterior o el siguiente. Si se trata de la primera tarea, sólo es posible ir hacia delante (el botón de paso anterior no se muestra). Si por el contrario, es la última tarea, el botón de siguiente cambia a un icono de finalización correcta. Como el usuario tiene capacidad de influir en la secuencia de tareas (por ejemplo, en tareas de selección múltiple o repetidas), es más coherente que el teléfono móvil almacene un historial con todas las tareas ejecutadas y por tanto, pedir la tarea anterior consiste únicamente en navegar en este historial y obtener de él la tarea anterior.

3.8. EJEMPLO DE USO

Con este ejemplo se pretende exponer, de forma clara, el funcionamiento del sistema aQRdate descrito a lo largo de este capítulo. Para ello, se muestra la actividad “Hacer Fotocopias”. Los pasos principales que habría que hacer para realizar esta tarea serían:

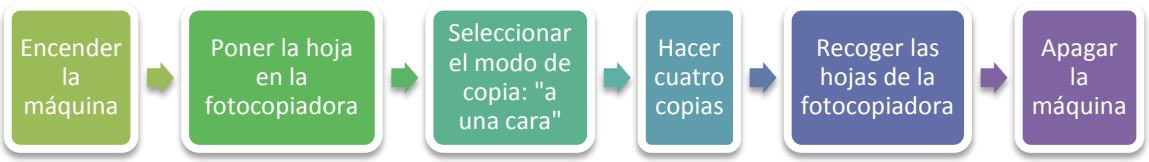


Figura 3.6: Fases de la actividad “Hacer Fotocopias”.

Para que esta actividad sea llevada a cabo en el sistema aQRdate, hay que utilizar la sintaxis definida en apartado 3.4 (descripción de las actividades). Las características que esta actividad tiene que cumplir son:

- Una Routine que englobe todos los pasos de la secuencia en forma de hijos.
- Una Task hija para cada una de las tareas descritas, excepto:
    - La tarea de selección múltiple (descrito en el apartado 3.6.2) en la que se pueda elegir el modo en el que se quiere hacer la fotocopia (a una cara o a dos caras).

- La tarea de hacer copias ya que será una única tarea del tipo tarea repetitiva explicada en el apartado 3.6.1. En este caso el terapeuta dejará al usuario elegir el número de copias luego la propiedad "nTimes" de la capacidad "hastToBeRepeated" valdrá *-1*.

- Cada una de las Task deberá contener Fragments con la información que se desee proporcionar con la tarea. En este ejemplo, se va a mostrar un Fragment de tipo título con el texto que define la tarea y otro Fragment con una imagen que explique cada paso.

El esquema de la figura anterior se puede expresar en forma de árbol de la siguiente manera:

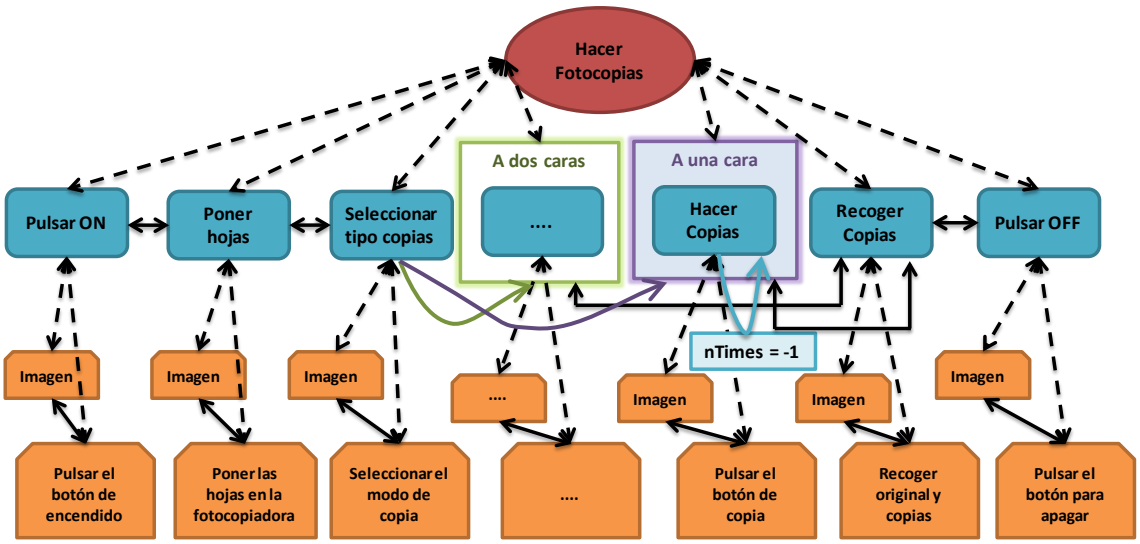


Figura 3.7: Esquema en forma de árbol de la actividad "Hacer Fotocopias".

En esta representación cada una de las Task (de color azul) está relacionada con una Task previa y una Task siguiente. Además éstas están relacionadas con un Fragment de tipo título con la descripción de la tarea y otro con la imagen explicativa de este paso (ambos de color naranja). Los Fragment también están enlazados por relaciones previo/siguiente. También se puede observar cómo la tarea "seleccionar tipo de copias" es de selección múltiple pues ella conduce o bien a "A dos caras" o bien a "A una cara" (para este ejemplo se ha obviado el contenido de la primera opción pues la opción escogida será la segunda). Otra de las cosas que se puede ver es que la tarea "Hacer Copias" (perteneciente a la opción "A una cara") es una tarea repetitiva cuyo valor ("nTimes") tiene un valor *-1*, luego este valor será elegido por el usuario.

Una vez creada toda la secuencia de pasos con su información correspondiente, se genera el código QR que contenga el nombre de la rutina creada "Routine:hacerFotocopias@amilab". Este es el código que se escaneará con el teléfono móvil por parte del usuario y cuya posición debe de estar lo más posiblemente ligada a la tarea a realizar. En este caso, una de las mejores opciones sería situar el código en la misma fotocopiadora.

La secuencia completa de pantallas que el usuario vería sería:

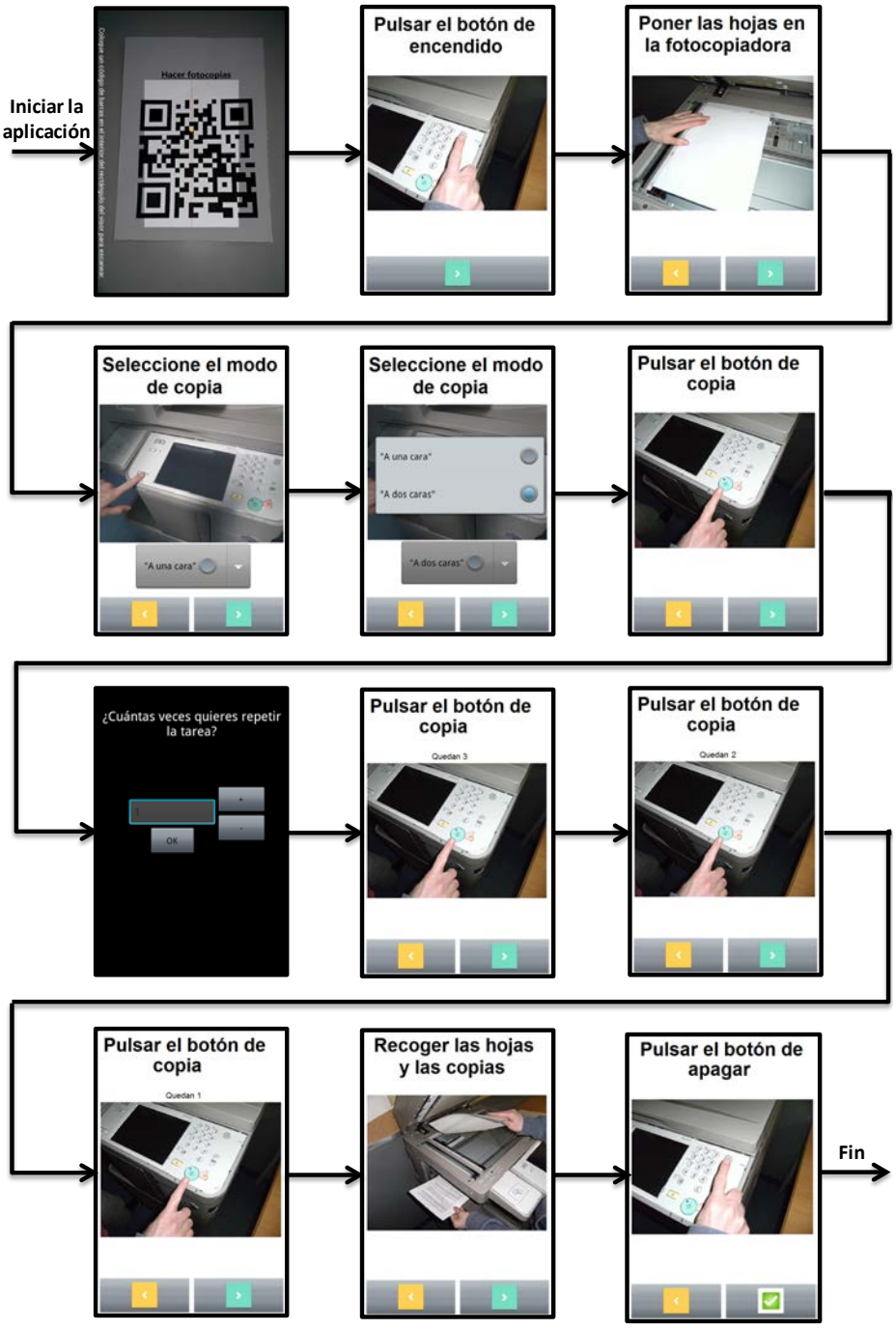


Figura 3.8: Secuencia completa de la actividad "Hacer Fotocopias".

¿Cómo está actuando el sistema para ir generando esos pasos?

1. El usuario apunta con el móvil al código QR de la tarea a realizar. En este caso quiere hacer fotocopias, luego acude a la máquina fotocopiadora donde se encuentra el código a escanear y lo escanea.
2. Se notifica al servidor de aQRdate que la tarea escogida es “hacerFotocopias” y éste construye el árbol.
3. El teléfono móvil pide la primera tarea. El servidor de aQRdate recorre el árbol y le devuelve la primera tarea que es “Pulsar ON”.
4. El usuario aprieta el botón de siguiente tarea. Como consecuencia el móvil pide la próxima tarea. El servidor devuelve “PonerHojas”.
5. Al igual que en el paso anterior, el servidor devuelve ahora “Seleccionar tipo copias”.
6. El usuario, en la tarea “Seleccionar tipo copias” tiene que elegir cómo desea hacer las fotocopias si “A dos caras” o “A una cara”. El usuario escoge hacer las fotocopias por una cara y aprieta el botón de siguiente tarea.
7. La siguiente tarea que devuelve el servidor es “Hacer Copias”. El usuario la realiza (una vez). Esta tarea es repetitiva y el número de repeticiones viene definido por el usuario. Cuando el usuario selecciona el botón de tarea siguiente, el servidor pregunta al usuario cuántas veces más quiere repetir la tarea que acaba de realizar. En este caso, como se quieren realizar cuatro fotocopias y ya se ha realizado una, se seleccionan tres.
8. Cada vez que se quiera avanzar en la secuencia de tareas, el servidor enviará la misma tarea (“Hacer Copias”) pero con un contador en el que figura el número de copias restantes.
9. Una vez realizadas las copias, la siguiente tarea que devuelve el servidor es “Recoger copias”.
10. Por último, la tarea que se obtiene es “Pulsar OFF” para apagar la fotocopiadora. En esta tarea desaparece el botón de tarea siguiente por un botón de actividad concluida.

4

aQRdate: HERRAMIENTA DE AUTOR. DISEÑO

En el proceso de rehabilitación de personas con discapacidad cognitiva participan expertos de diferentes áreas (terapeutas ocupacionales, fisioterapeutas, neuropsicólogos, etc.), que trabajan para ayudar a estos pacientes a adquirir un mayor nivel de autonomía en sus vidas. Sin embargo, este tipo de proceso supone unos costes logísticos y económicos muy importantes que se pueden reducir con el uso del sistema aQRdate, posibilitando que el personal médico asista a más usuarios y sin necesidad de una intervención directa e intrusiva en la vida del paciente.

En el momento de iniciar este trabajo, aQRdate sólo contaba con la aplicación móvil destinada al paciente (persona con daño cerebral adquirido), explicada en el capítulo anterior. Los profesionales que atienden a este tipo de pacientes no podían modificar actividades ya existentes en el sistema o introducir nuevas, pues no contaban con los conocimientos informáticos necesarios para generar el archivo de entidades en el que se describen ni tampoco podían actualizar la información en el servidor. Por ello, tenían que avisar a los programadores del sistema para que fueran ellos los que introdujeran la información mediante el código descrito en la sección 3.4. Como consecuencia de esta situación el contenido que se mostraba en el móvil al usuario no podía ser actualizado a tiempo real y, además, se producía un aumento en los costes, ya que cualquier cambio que se deseara introducir en el sistema llevaba aparejado un gasto en personal informático encargado de hacer efectivo este cambio en el servidor.

Por la situación que se acaba de describir, se consideró completar el sistema aQRdate (Figura 4.1) desarrollando una herramienta de autor para el diseño actividades de la vida diaria (conjunto de instrucciones para realizar actividades), por parte del personal

médico, de una forma sencilla e intuitiva; sin que sea necesario tener conocimientos informáticos avanzados (como hasta ahora). El diseño e implementación de esta herramienta constituye la parte central de este PFC.

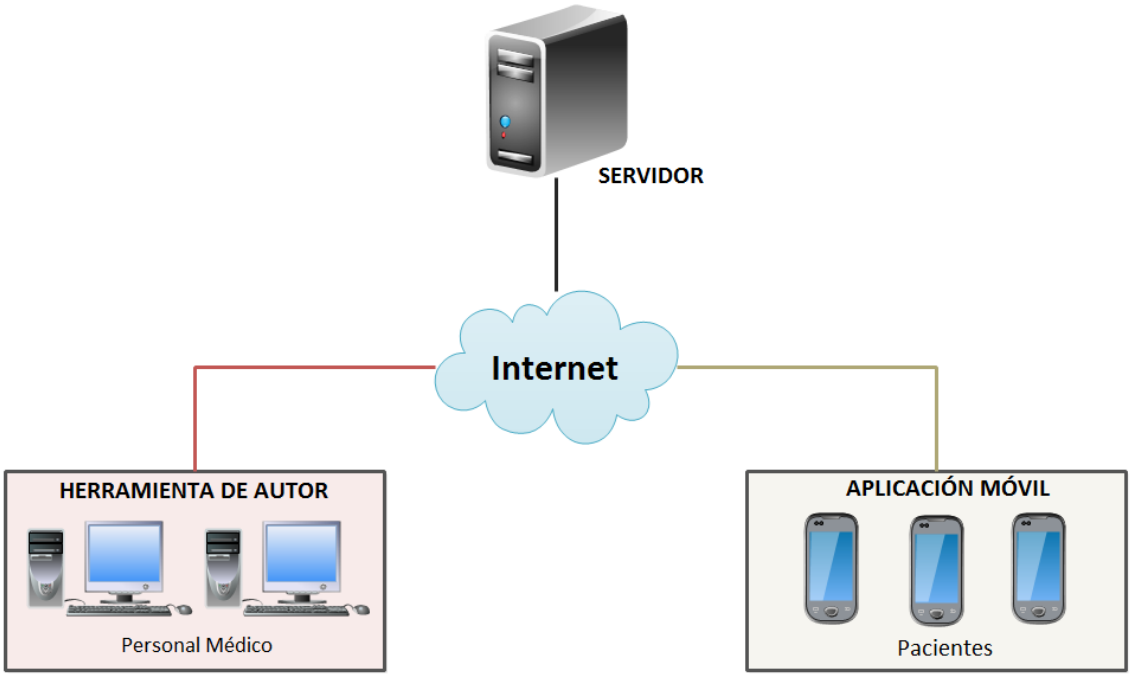


Figura 4.1: Esquema completo del sistema aQRdate.

4.1. REQUISITOS

Siguiendo la división que realiza Sommmerville [43] acerca de los requerimientos de la ingeniería de *software*, se tiene:

- Requisitos funcionales: Definición de los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular y cómo se debe comportar ante situaciones particulares.
- Requisitos no funcionales: Son las limitaciones sobre servicios o funciones que ofrece el sistema, tales como restricciones de tiempo, sobre el proceso de desarrollo, estándares, etc.

Los requisitos que se detallan a continuación son los requisitos finales, obtenidos tras realizar tres iteraciones en el proceso de diseño iterativo explicado en la sección 4.2.

4.1.1. Requisitos funcionales

Los requisitos funcionales (RF) que este trabajo debe cumplir son:

Interfaz

1. La interfaz del programa debe poder adaptarse al usuario.
    - Para ello, se definirá un tamaño mínimo (la vista no se puede reducir más) y, tomando éste como referencia, se podrá maximizar (en función del tamaño de la pantalla que tenga el usuario) y, posteriormente, minimizar la aplicación.
    - Se permitirá que la aplicación se mueva por toda la pantalla del ordenador.
    - Otra de las opciones que el usuario tendrá en la aplicación será la posibilidad de reescalar el tamaño del programa.

Creación de Actividades

2. Con el uso del sistema, se debe ser capaz de crear, modificar y borrar actividades (cuyo contenido hace referencia a actividades de la vida diaria, por ejemplo hacer el desayuno o hacer la compra en el supermercado).
    - Las actividades tendrán un nombre por defecto que podrá ser editado por el usuario.
3. Cada actividad estará compuesta por tareas y/o subactividades.
    - La actividad puede estar compuesta de una o varias tareas. Estas tareas representan los pasos intermedios que se efectúan para realizar la actividad.
    - También puede estar compuesta de una o varias subactividades (actividades contenidas en otra actividad).
4. Además, estas tareas o subactividades deben poder ordenarse de forma sencilla dentro de la actividad.
    - Para la ordenación de los pasos que componen una actividad, lo más sencillo sería ordenarlos arrastrándolos con el ratón a las posiciones en las que se desee que se muestren al paciente.

5. Una vez que la actividad ha sido creada, el usuario podrá generar, mediante el programa, el código QR correspondiente a esta actividad para posteriormente imprimirlo y colocarlo en el lugar donde se desarrolle.
6. Las actividades se pueden adaptar a los usuarios. Adaptar al usuario, en este contexto, se define como la posibilidad de eliminar algunos pasos que componen las actividades para un usuario concreto.
    - Por este requisito, el *software* de autor debe poder almacenar distintos usuarios y memorizar los pasos de las actividades personalizadas para ellos.

Creación de Tareas

7. La creación y edición de tareas será de tipo WYSIWYG (*What You See Is What You Get*). De esta forma, la persona que está diseñando la tarea ve en cada momento el resultado tal y como lo vería el paciente en el móvil.
    - Para ello, es fundamental mostrar, en la edición de tareas, una vista previa de la tarea tal y como se vería en la terminal. Además debe actualizarse con los cambios de forma automática.
    - Al igual que con las actividades, las tareas tendrán un nombre que el sistema pondrá pero que podrá ser modificado por el usuario.
8. Del mismo modo que en el caso de las actividades, con las tareas se pueden realizar operaciones de cortar y copiar, para finalmente pegarse en otra actividad con el objetivo de reusarlas.
9. En el caso de las tareas, su contenido estará formado por un título explicativo del paso que se tiene que hacer y un contenido multimedia.
    - El contenido multimedia que se muestre en una tarea puede ser: imagen, vídeo, audio o una descripción textual.
10. Cada tarea, durante la edición, aparte de poder introducir/modificar el título y el contenido multimedia, debe permitir también la creación de tareas repetitivas o de selección múltiple (nunca ambas a la vez).
    - Cada tarea se puede repetir. Por ello, el sistema debe dejar introducir el número de veces que el personal médico quiere que el

paciente repita esta actividad. Otra opción es que este personal decida que el número de repeticiones sea escogido por el paciente en el momento de su ejecución.

- Cada tarea puede tener la opción de mostrar una lista de selección múltiple. Con ello, cuando el usuario llegue a esta tarea, se le muestra la lista y escoge una de las opciones. La siguiente tarea será aquella que corresponda con la opción elegida. Para esta opción, la herramienta de autor debe proveer una edición de opciones de fácil uso.

Gestión de la ayuda al usuario

11. Este programa debe mostrar, en todo momento, un menú de navegación para situar el lugar en donde se encuentra el usuario dentro de la actividad (una actividad puede tener varias subacitividades, lo que puede complicar al usuario el saber en qué lugar se encuentra dentro de la actividad principal).
12. Aunque el sistema tiene que poderse manejar de una forma sencilla e intuitiva, pensado en las personas a las que va destinado, aQRdate proveerá al usuario de un menú de ayuda.
    - En él se podrá consultar diferentes manuales acerca de la funcionalidad de esta aplicación.
13. Otro de los requisitos que debe cumplir este programa es mostrar una descripción emergente, también llamado *tooltip*, cuya finalidad es mostrar al usuario información de un elemento cuando el cursor pase encima de éste.
    - Este tipo de información se mostrará principalmente en los botones contenidos en la herramienta de autor con el fin de que los usuarios conozcan la función que éstos poseen.
14. Los cambios realizados en el sistema se han de poder guardar en cualquier momento en el servidor. De esta forma, los cambios se verán actualizados en el terminal móvil cuando se ejecute la aplicación. También, cuando el usuario cierre la aplicación, se deberán guardar automáticamente.

4.1.2. Requisitos no funcionales

Los requisitos no funcionales (RNF) que este programa debe cumplir son:

15. Usabilidad: El sistema estará dotado de una interfaz altamente usable, de manera que su aprendizaje y manejo resulte fácil por cualquier tipo de usuario.
    - El sistema mostrará una interfaz con menús, ventanas, mensajes informativos y otras opciones familiares para el usuario.
16. Disponibilidad: Este programa debe de poder usarse en cualquier lugar y en cualquier momento.
    - El único requerimiento es disponer de internet debido a la conexión con el servidor donde se encuentra la información y la máquina virtual de Java (alta portabilidad).
    - Se tiene que garantizar una alta disponibilidad de la aplicación, de manera que el servicio esté siempre disponible.
17. Fiabilidad: El sistema que se pretende desarrollar ha de ser robusto. Intentando, en la medida de lo posible, que la tasa de fallos que presente el programa durante el uso por parte del personal médico sea el menor posible.
18. Mantenibilidad: Se deberán realizar revisiones periódicas del sistema, de manera que la aplicación se encuentre actualizada constantemente para evitar posibles fallos en los equipos o en la información y para garantizar un correcto funcionamiento del sistema.
    - Para cumplir este requisito se deberán de llevar a cabo tareas de tipo correctivo (modificaciones que se hacen en el sistema tras detectar defectos, ambigüedades o errores), preventivo (modificar el sistema con los cambios necesarios para mantener la eficiencia y fiabilidad del *software*), adaptativo (modificaciones en el sistema para acomodarlo a cambios físicos del entorno) y perfectivo (mejorar el sistema para cumplir con las nuevas necesidades/requerimientos de los usuarios).

19. Rendimiento: El tiempo de respuesta ante las diferentes operaciones que el usuario puede realizar debe ser breve.
    - Se debe de evitar esperas al usuario cuando realice operaciones sobre actividades y tareas o cuando se actualice la información el servidor.
20. Extensibilidad: Se ha de implementar la aplicación de tal manera que, en caso de que se encuentre la necesidad de introducir una nueva funcionalidad, se pueda añadir fácilmente sin un alto coste de programación.
    - Implementar la aplicación muy modularmente.
    - Generar una documentación (interna o externa) fácil de comprender y actualizar, que indique cómo y dónde modificar en caso de necesidad.
21. Interoperabilidad: La información que se genere con la herramienta de autor será almacenada en el servidor para posteriormente ser mostrada en el teléfono móvil del usuario. Es por ello que, como resultado del diseño de actividades, se tiene que generar el mismo lenguaje que el sistema aQRdate utiliza en los terminales móviles para que exista una buena interoperabilidad entre ambos equipos (herramienta de autor y teléfonos móviles).

4.2. PROCESO DE DISEÑO ITERATIVO

4.2.1. Introducción

Para el proceso de diseño de la herramienta de autor se ha seguido un proceso de diseño iterativo similar al que se observa en la Figura 4.2.

El primer paso que se realiza es establecer los requisitos que el *software* debe cumplir. Con esta información se realiza un prototipo del programa con el fin de valorar si se han cumplido los requisitos que se han detallado previamente o si a la vista del modelo planteado surgen nuevos. Otros de los aspectos a valorar son los relativos a la funcionalidad y la estética. Cuando alguno de estos puntos no ha sido superado, se

procede de nuevo a repetir el proceso hasta que el resultado final es el deseado. Es en este momento cuando termina esta fase dando paso a la implementación del diseño aprobado.

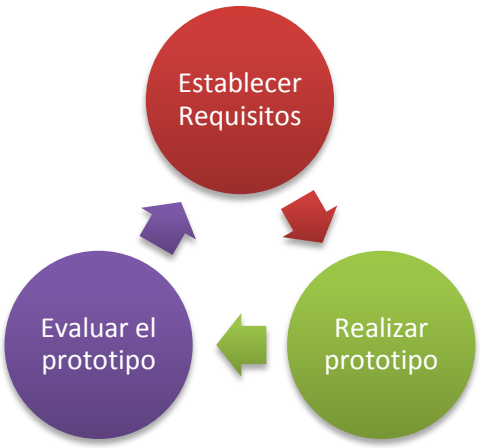


Figura 4.2: Esquema de diseño iterativo.

Para la realización del diseño de este trabajo fueron necesarias tres iteraciones del esquema mostrado en la Figura 4.2:

1. En primer lugar se fijaron los requisitos tras una explicación de los objetivos del proyecto y también teniendo en cuenta el estado del arte consultado. A continuación se diseñó el primer prototipo y se realizó una presentación a un grupo de expertos con el fin de valorarlo. Una vez valorado, se propusieron mejoras y modificaciones, lo que condujo a la segunda iteración.
2. El segundo prototipo sigue la línea del primero pues lo que se hizo fue rehacer el modelo creado en la primera iteración pero teniendo en cuenta todos los comentarios que fueron aportados por el grupo de expertos. Una vez estuvo listo para realizar una presentación, ésta se llevó a cabo y de nuevo se volvieron a obtener nuevas mejoras y comentarios para la creación de la herramienta de autor.
3. Tras la valoración obtenida se concluyó que lo mejor era crear un nuevo diseño (sin basarse demasiado en los prototipos anteriores) para intentar cambiar el aspecto de la interfaz ya que éste resultaba bastante pobre. La interfaz propuesta es más sencilla que las anteriores pues elimina pantallas intermedias y menús que complicaban su uso. El tercer prototipo fue el final y por tanto este diseño es el que implementó.

4.2.2. Primera Iteración

Una vez obtenidos los requisitos iniciales, mediante la librería Swing de Java [44], destinada al desarrollo de interfaces gráficas, se diseñó el primer prototipo a evaluar.

Figura 4.3: *Storyboard* del primer prototipo.

El resultado obtenido fue una presentación de catorce pantallas en las que se recorren los posibles caminos que puede tener el flujo del programa durante su ejecución en función de la interacción del usuario, también llamado *storyboard* [45].

A continuación se va a mostrar cómo sería una posible ejecución de este primer prototipo:

1. Cuando se inicia el programa se muestra una pantalla de inicio en la cual hay tres opciones: “Crear una actividad”, “Modificar una actividad” y “Borrar una actividad”.

   En este primer caso el usuario elige la opción “Crear una actividad” (Figura 4.4):

Figura 4.4: Primer Prototipo. Opción “Crear una actividad”.

2. Tras escoger esta opción, se muestra la siguiente pantalla que corresponde con la creación de actividades. Para crear la actividad se tiene que arrastrar los elementos del panel izquierdo (actividad y tarea), así como las relaciones entre padres e hijos (flecha de color verde). Para establecer el orden en el que se muestran las tareas y subactividades dentro de la actividad, se utilizan las flechas de color naranja. El resultado de utilizar este programa es el siguiente:

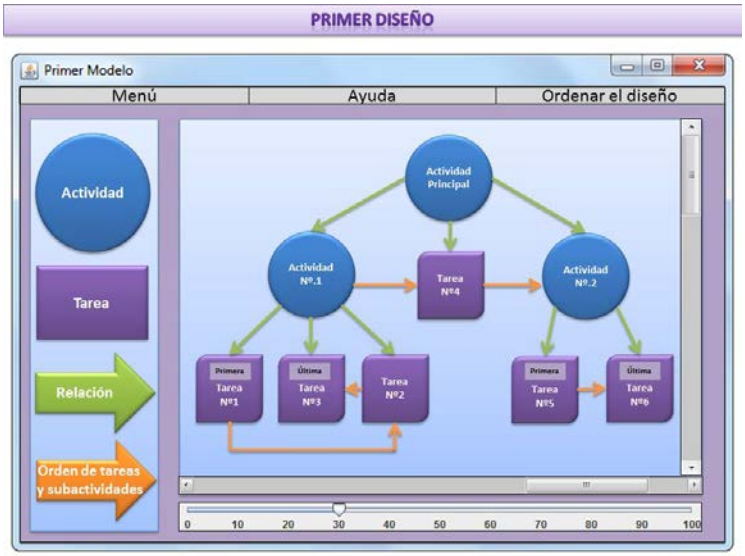


Figura 4.5: Primer Prototipo. Pasos para crear una actividad.

Como se observa en la imagen, se ha creado una actividad principal con dos subactividades y una tarea intermedia. La primera subactividad (llamada Actividad nº 1) tiene tres tareas ordenadas según muestran las fechas naranjas. Una vez concluida, siguiendo el orden que marcan las flechas de color naranja, se continúa con la Tarea nº4 y después con la segunda subactividad (llamada Actividad nº 2). Ésta tiene dos tareas ordenadas de la misma forma. Por tanto, la ejecución de esta actividad en el terminal móvil del paciente sería: Tarea nº1, Tarea nº2, Tarea nº3, Tarea nº4, Tarea nº5 y Tarea nº6.

Además, en esta pantalla de desarrollo se puede observar:

- En la parte superior se muestra una barra con las opciones de "Menú" para volver a la pantalla inicial del programa, "Ayuda" para que el usuario pueda consultar una guía acerca del manejo de esta herramienta y "Ordenar el diseño", pensado para ordenar, en forma de árbol, la actividad que el personal médico acaba de diseñar (en la Figura 4.5 se puede ver el diseño ya ordenado).
- En la zona inferior de la pantalla se muestra un *slider* (deslizador). Este elemento está diseñado para que se pueda ampliar o reducir la parte central de la pantalla en donde se diseña la actividad (zoom).

3. Una vez creada la actividad, se puede cambiar el nombre:

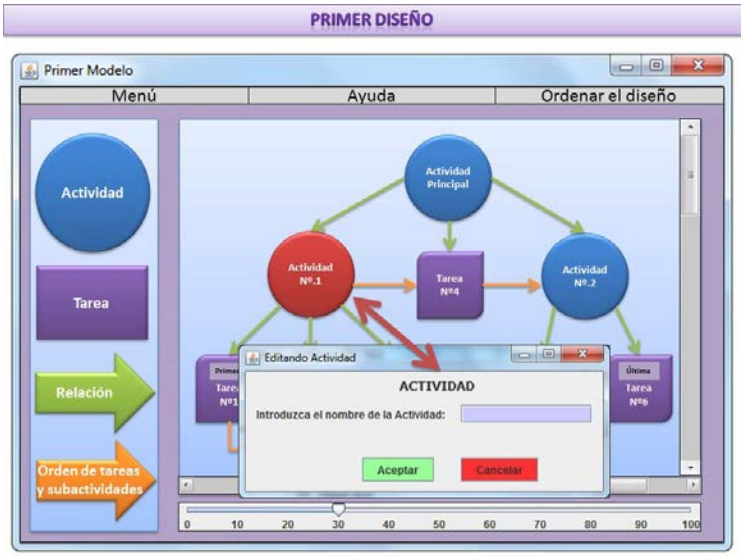


Figura 4.6: Primer Prototipo. Editar el nombre de una actividad.

Para realizar este cambio se hace doble clic sobre el icono que representa una actividad. Finalmente se introduce el nombre como muestra la figura anterior (Figura 4.6).

4. Ahora se muestra la forma de editar la información de una tarea (Figura 4.7).

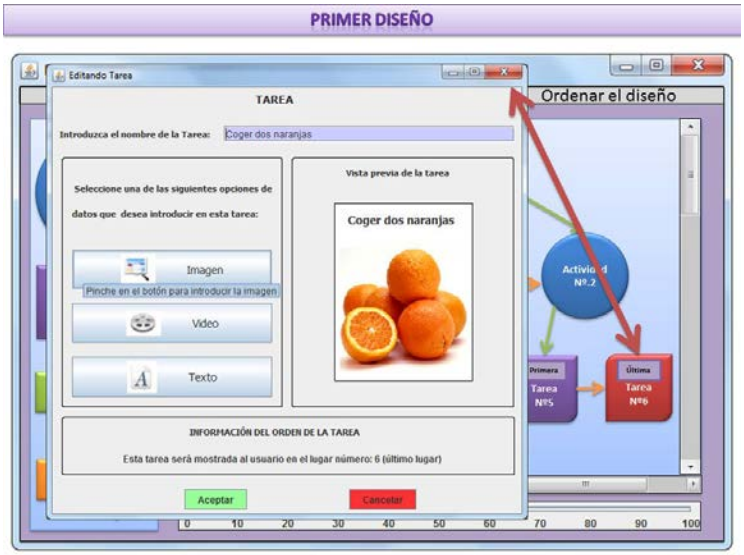


Figura 4.7: Primer Prototipo. Editar el contenido de una tarea.

Tiene tres partes claramente diferenciadas:

- Lo primero que se observa en esta figura, en la parte superior, es un campo de texto para poder introducir el nombre de la tarea que se quiere mostrar al paciente.
- En la mitad izquierda de la pantalla se muestra el tipo de datos que puede tener una tarea. En función de la opción escogida el contenido podrá ser imagen, vídeo o texto. Para el caso de la imagen, como se puede ver en la Figura 4.8, se muestra un panel de selección en donde el usuario selecciona la fotografía a mostrar. Si se prefiere mostrar un vídeo (Figura 4.9), simplemente se tiene que escribir en el cuadro de texto emergente la dirección web de este contenido. Si por el contrario se quiere mostrar al usuario una descripción textual, basta con redactar este texto en la ventana emergente que aparece cuando se presiona la opción de texto (Figura 4.10).

Además de estas tres opciones, se observa que cuando se sitúa el ratón por encima de una opción, se muestra la información de ella (en este caso se está mostrando la información sobre la opción de imagen).

En la mitad derecha aparece una vista previa de la tarea creada. Se muestra el nombre de la tarea así como el tipo de datos, que en este caso es una imagen.

- Por último, se indica la posición en que se sitúa esta tarea con respecto al resto de ellas dentro de la actividad a la que pertenezcan.

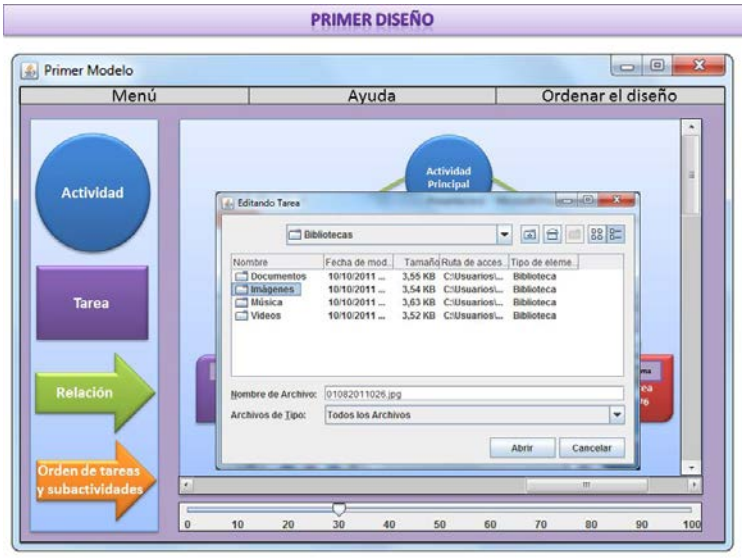


Figura 4.8: Primer Prototipo. Opción Imagen.

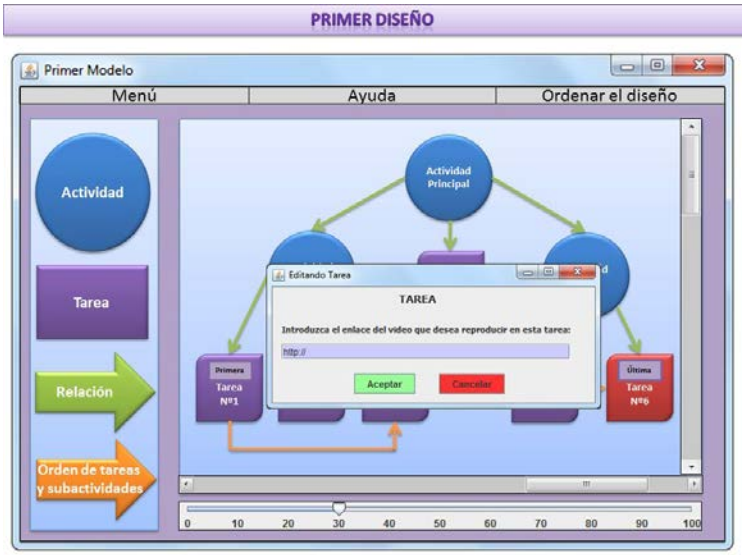


Figura 4.9: Primer Prototipo. Opción Vídeo.

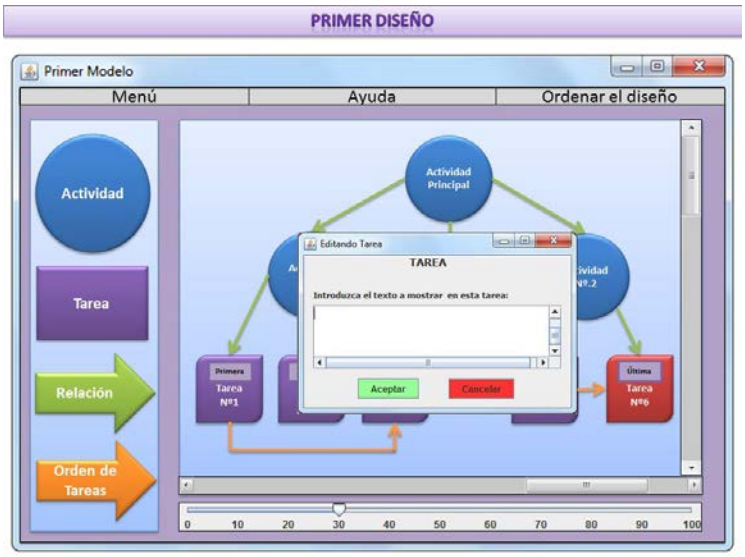


Figura 4.10: Primer Prototipo. Opción Texto.

5. Una vez concluida la opción de "Crear una actividad", se vuelve al menú inicial para mostrar las distintas pantallas que tiene la opción "Modificar una actividad" (Figura 4.11).

Figura 4.11: Primer Prototipo. Opción "Modificar una actividad".

6. Después de escoger la opción de modificar, se muestra, en forma de lista, todas las actividades que tiene el sistema. El usuario de la herramienta de autor tiene que elegir la actividad a modificar y pinchar en el botón de color verde llamado "Aceptar". Si por el contrario presionase el botón de color rojo cuyo texto es "Cancelar" volvería al menú de inicio (Figura 4.12).

Figura 4.12: Primer Prototipo. Lista de las actividades a modificar.

7. En el momento en que ya ha sido elegida la actividad que el usuario quiere modificar, se muestra, como en el caso de la creación de una nueva actividad, todo el contenido de la misma. Para cambiar cualquier subactividad o tarea existente sólo tiene que hacer doble clic sobre ella. También se puede introducir nuevos contenidos arrastrando los objetos (subactividades, tareas y relaciones) que se deseen del panel de la derecha de la imagen. Para eliminar cualquier contenido hay que seleccionarlo y apretar la tecla suprimir.

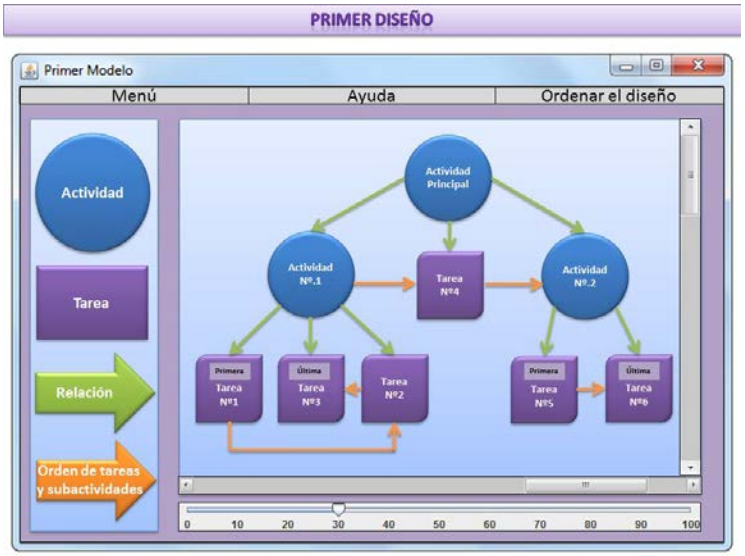


Figura 4.13: Primer Prototipo. Modificación de la actividad.

8. La última opción que se muestra en el menú inicial es la de borrar una actividad.

Figura 4.14: Primer Prototipo. Opción "Borrar una actividad".

9. Al igual que en caso de la modificación, para borrar una actividad se presenta, al usuario de este *software*, una lista de todas las actividades que tiene el sistema (Figura 4.15). Se selecciona la actividad a borrar y se presiona "Aceptar". En el caso en que se quisiera cancelar el borrado se pulsaría "Cancelar".

   A continuación, se muestra una ventana emergente en la que se vuelve a preguntar al usuario sobre si quiere borrar la actividad que previamente ha seleccionado de la lista (Figura 4.16). En caso afirmativo pulsaría "Sí" y en caso negativo apretaría "No". Tras ello, se vuelve al menú inicial.

Figura 4.15: Primer Prototipo. Lista de las actividades a borrar.

Figura 4.16: Primer Prototipo. Confirmación del borrado de la actividad.

Una vez explicado en profundidad toda la funcionalidad de este primer prototipo (tal como se acaba de realizar en esta memoria), fue analizado por personas expertas en el desarrollo de *software* para tecnologías asistivas. Para ello, se realizaron una serie de preguntas acerca de esta herramienta cuya respuesta fue la siguiente:

1. ¿Es la mejor opción mostrar al usuario un menú inicial con las tres opciones descritas anteriormente (crear actividad, modificar actividad y borrar actividad)?

   Las opciones de crear y borrar actividad son correctas y se entiende su funcionamiento. En cuanto a la posibilidad de modificar una actividad sería mejor mostrar como opción "consultar actividades". Así el personal médico puede ver las actividades que el sistema ya posee y decidir, una vez mostrada, si sobre ella realiza alguna modificación. Además, en esta pantalla de inicio se ha de especificar que la persona que esté usando el programa tiene que escoger una de estas tres posibles opciones.

2. En cuanto a la opción de crear actividades, ¿es intuitivo y fácil de utilizar el editor de actividades descrito en la Figura 4.5?

   Para la representación del contenido de la actividad se está utilizando un modelo de árbol, similar al que crea el servidor de aQRdate para establecer el orden de las tareas que se han de mostrar en el teléfono móvil del paciente.

Sin embargo, es más sencillo, de cara al usuario de la herramienta, utilizar un modelo *Wizard*; un modelo de pregunta - respuesta con flechas de navegación, en cada pantalla, de anterior y siguiente (así el usuario puede modificar los datos previamente introducidos). De esta forma, para crear la actividad se pregunta por cuántas subactividades y tareas quiere que tenga y, una vez creadas, le pide al usuario que rellene cada objeto con la información necesaria. Finalmente es de utilidad mostrar la actividad en forma de árbol para ver, de una forma gráfica, las relaciones que el contenido tiene entre sí.

3. Para las tareas, ¿el editor descrito en la Figura 4.7 se ajusta a los requisitos que tiene que cumplir la edición de tareas?

   Sí, se muestran las diferentes opciones de contenido que puede tener una tarea (imagen, texto, etc.), se puede poner el nombre que se desee a la tarea y además se muestra una vista previa del contenido que se está creando.

4. A la hora de ordenar el contenido (tareas o subactividades) dentro de una actividad, ¿es la mejor opción que el usuario utilice flechas de ordenación para establecer el orden?

   Al igual que en la pregunta anterior, la idea del árbol con las flechas, una vez creada la actividad, aporta mucha información. Sin embargo, puede resultar un gran problema de comprensión, para personas no especializadas en el uso de esta herramienta, especialmente en la etapa de creación de actividades. Por ello, lo mejor sería mostrar en una lista las subactividades y tareas que contiene una actividad y, mediante el arrastre de cada entrada de esta lista, cambiar el orden.

5. ¿Es suficiente el diagrama en forma de árbol del contenido de la actividad de cara al usuario o hay que mostrar una opción de vista previa de ella?

   Es mejor mostrar una opción que sea “Vista Previa”. El diagrama muestra la relación existente entre los distintos elementos (subactividades y tareas) pero no se puede ver de una forma sencilla el contenido de cada uno de ellos (la información que se muestra al paciente en el dispositivo móvil en cada tarea).

6. Para la opción “Modificar una actividad”, ¿la forma en la que la aplicación guía al usuario para manejar esta opción es sencilla?

   Sí, el usuario solamente tiene que elegir una actividad de la lista y, una vez abierta, editarla. Como se ha comentado en la primera cuestión, lo mejor sería que esta opción tuviese como nombre “Consultar Actividades”. De esta forma el usuario puede ver la actividad y, si lo desea, modificarla.

7. Para la última de las posibilidades que el menú de inicio muestra: “Borrar una actividad”. ¿Hay otra forma mejor para borrar una actividad del sistema?

   Es una opción bastante sencilla, luego con la lista que se muestra para que se seleccione la actividad a borrar es suficiente. Es una buena idea la ventana emergente preguntando de nuevo acerca de la eliminación de la actividad previamente seleccionada (similar a lo que ocurre en otros sistemas operativos como Windows cuando se pretende eliminar algún documento).

   Otra de las opciones sería que, cuando se listan las actividades a consular (segunda opción), se dé la posibilidad de eliminar la actividad seleccionada. Así el menú inicial pasaría a tener dos opciones en vez de tres.

8. Sobre el tema de la información que se debe suministrar al usuario para cada elemento que tiene esta aplicación. ¿Es suficiente el *tooltip* (información emergente)?

   Sí, en cada botón de la aplicación, cuando se deslice el ratón sobre él, se debe mostrar una descripción con la información más descriptiva posible sobre la funcionalidad de este elemento.

   Además, como se muestra en la presentación de este primer prototipo, el usuario debe tener a su disposición un menú de ayuda para consultar la información relativa a la utilización del sistema.

Tras esta valoración, se diseña un nuevo prototipo teniendo en cuenta los comentarios aportados por los expertos.

4.2.3. Segunda Iteración

Tras esta valoración, se diseña un segundo prototipo teniendo en cuenta los comentarios aportados por los expertos. El resultado obtenido es el siguiente:

Figura 4.17: *Storyboard* del segundo prototipo.

Al igual en la primera iteración se utiliza la librería Swing de Java [44] para crear este segundo diseño.

Una vez creado el prototipo, se realiza nuevamente una presentación de la ejecución que haría el programa en función de las opciones que escogiese el usuario:

1. Al igual que con el modelo anterior, la pantalla de inicio tiene tres opciones:
    - Crear una actividad.
    - Consultar actividades (en vez de "Modificar actividades" como se comentó en respuesta a la primera cuestión planteada al grupo de expertos).
    - Borrar una actividad.

    Además, esta pantalla contiene otra modificación que se sugirió durante la evaluación: se le informa al usuario que debe de escoger entre una de las opciones que se acaban de mencionar (Figura 4.18).

Figura 4.18: Segundo Prototipo. Opción "Crear una actividad".

2. Se selecciona la opción de "Crear una actividad". En esta pantalla (Figura 4.19) se puede ver el modelo *Wizard* (modelo de pregunta y respuesta con flechas de navegación hacia atrás y hacia delante para que el usuario pueda moverse entre pantallas de una forma sencilla) implementado, en vez del modelo en forma de árbol que tenía el diseño inicial.

En concreto, se le pide al usuario que introduzca el nombre de la actividad y seguidamente se le pregunta por el número de subactividades y de tareas que quiere que tenga esta actividad (en el anterior prototipo este procedimiento se realizaba de forma gráfica arrastrando estos objetos al panel de edición).

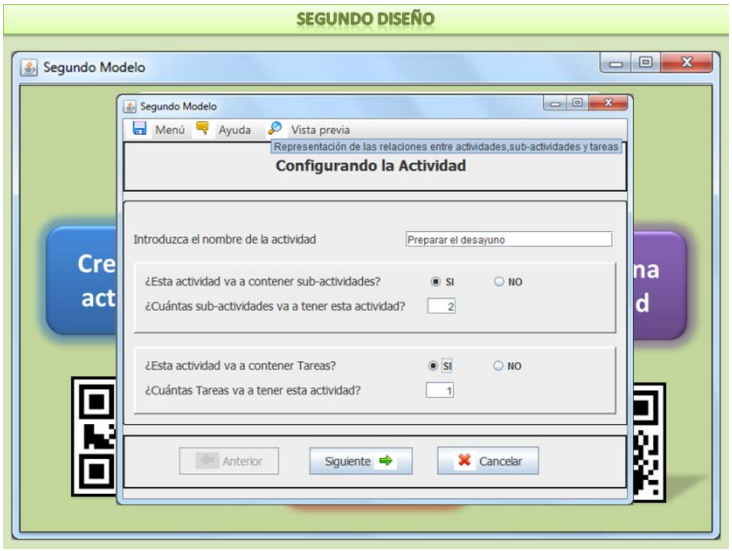


Figura 4.19: Segundo Prototipo. Pasos para crear una actividad.

3. Una vez introducido el número de elementos que componen la actividad, el programa pide que se definan.

   Para el caso de las subactividades hay que definir el nombre que tendrá, así como el número de tareas que la compondrán (Figura 4.20):

Figura 4.20: Segundo Prototipo. Pasos para editar las subactividades.

Para el caso de las tareas, la interfaz es similar a la descrita en el primer prototipo. En ella se introduce el nombre de la tarea y el tipo de datos multimedia que se quiera

mostrar (al tratarse de la misma interfaz de edición, se han omitido las pantallas que corresponden con cada una de las opciones de los datos). A la derecha de la pantalla se muestra la vista previa del contenido introducido (Figura 4.21).

Figura 4.21: Segundo Prototipo. Pasos para editar las tareas.

El resultado de completar una subactividad es el siguiente:

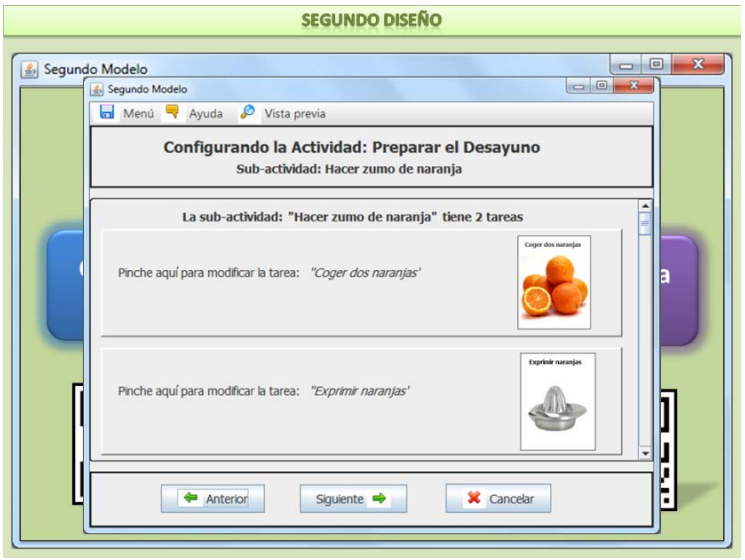


Figura 4.22: Segundo Prototipo. Subactividad completada.

Como se puede observar, se muestra, en forma de lista, cada una de las tareas que están contenidas en la subactividad. Para volver a editar cualquier tarea hay que hacer doble clic sobre ella. Si en cambio se quiere cambiar de orden, se debe arrastrar la tarea a la posición deseada.

Además, como se comentó en la valoración del primer diseño, se ha incluido (en la barra superior de esta pantalla) una "Vista Previa" de la subactividad, similar a la que vería el usuario, tarea a tarea, en el dispositivo móvil. El resto de opciones como el menú de ayuda o la posibilidad de volver al menú inicial se mantienen.

4. Tras definir toda la información que contiene una actividad, el programa lista todas las subactividades y tareas por si se desea cambiar el orden de éstos. Para ello se plantearon dos diseños (Figura 4.23 y Figura 4.24).

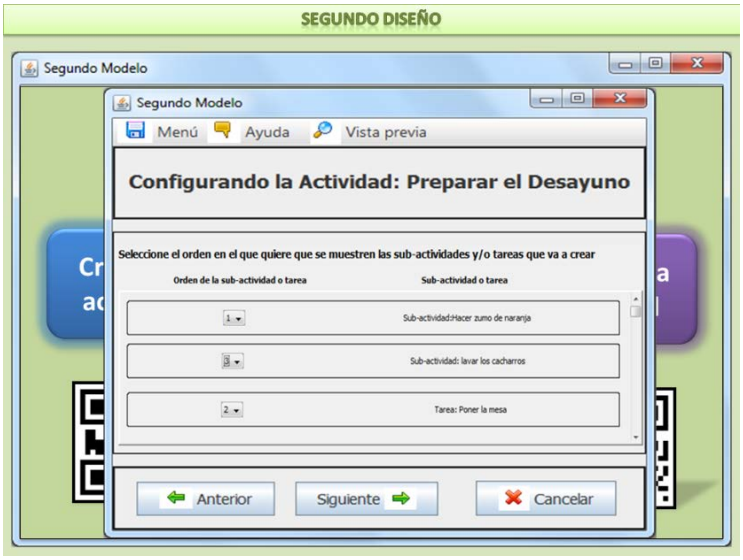


Figura 4.23: Segundo Prototipo. Primera opción para ordenar la actividad.

Figura 4.24: Segundo Prototipo. Segunda opción para ordenar la actividad.

En la primera opción de ordenación, las modificaciones con respecto al orden se harían seleccionando de la lista la posición deseada dentro de la actividad para cada uno de los objetos que la componen. En cambio, la segunda opción, para modificar la disposición de los elementos, basta con arrastrar cada uno de ellos a la posición deseada. Finalmente, por resultar más simple la interacción con el usuario se decidió usar el modelo de la Figura 4.24 (segunda opción).

5. Una vez definida toda la actividad, ésta se representa en forma de árbol:

Figura 4.25: Segundo Prototipo. Actividad completada en forma de árbol.

6. Otra de las opciones del menú principal es “Consultar actividades” (Figura 4.26).

Figura 4.26: Segundo Prototipo. Lista de las actividades a consultar.

Para ello, al igual que en el primer diseño, el usuario elige de una lista una actividad. Una vez que es escogida se muestran todos los pasos de la opción "Crear actividad", pero esta vez todos los datos aparecen rellenados con la información que se introdujo cuando se creó esta actividad (Figuras comprendidas en el intervalo 4.19 - 4.25). Si sólo se desea ver el contenido, mediante las flechas de navegación (anterior y siguiente) se puede ir avanzando hasta llegar a la representación de la actividad en forma de árbol. Si por el contrario, lo que se desea es cambiar algún dato, es posible hacerlo modificando los datos mostrados.

7. Por último, se muestra en el menú inicial la opción de "Borrar un actividad". El funcionamiento de esta opción es similar al descrito en la primera interacción. Se selecciona de la lista la actividad a borrar y se presiona el botón de "Aceptar". En el caso en que se quiera cancelar esta operación se pulsaría el botón de "Cancelar":

Figura 4.27: Segundo Prototipo. Lista de las actividades a borrar.

A continuación, se muestra una ventana emergente en la que se vuelve a preguntar al usuario sobre si quiere borrar la actividad que previamente ha seleccionado de la lista (

Figura **4.28**). En caso afirmativo se aprieta el botón de "Sí " y en caso negativo se selecciona el botón "No".

Figura 4.28: Segundo Prototipo. Confirmación del borrado de la actividad.

Para este segundo diseño las preguntas que se plantearon al grupo de expertos fueron:

1. ¿Resulta más fácil de usar el modelo *Wizard*?

   A priori, este modelo de pregunta y respuesta parece ser más sencillo pues el usuario, poco a poco, rellena la información de cada actividad cuando el programa se lo solicita. Sin embargo, debido al gran número de pantallas que se necesitan para crear una actividad resulta complejo, incluso más que el primer diseño.

2. Para ordenar las subactividades y tareas dentro de una actividad, ¿la ordenación de estos elementos resulta sencilla?

   En la Figura 4.24, el poder arrastrar cada elemento a la posición deseada resulta algo bastante intuitivo de utilizar. El problema de esta opción radica en que la ordenación sólo es posible realizarla una vez creado todo el contenido de la actividad. Lo ideal sería que en cualquier momento se pueda tener opción a modificar el orden establecido.

3. En cuanto al diagrama en forma de árbol, ¿Se comprende la información que representa?

   Este tipo de diagramas pueden resultar difíciles de entender si van dirigidos a todo tipo de público. Por ello, puede resultar una mejor opción mostrar la información que contiene una actividad de forma lineal. Así sería más sencillo

de comprender el orden en el que se mostraría el contendido, al usuario, en el teléfono móvil.

Por otro lado, si bien se ha incluido en este diseño la vista previa, ésta sólo está disponible para ver el contenido de las subactividades. Sería de mayor utilidad el poder ver la vista previa de toda la actividad (similar a lo que vería el paciente) o si se desea sólo de una subactividad.

4. Para la opción de "Consultar actividades", ¿las mejoras introducidas en esta opción del menú inicial son útiles?

   Al tener esta opción se puede tanto modificar actividades como simplemente consultarlas. Sin embargo, para consultar una actividad previamente creada y almacenada en el sistema, se tiene que pasar por todas las pantallas de configuración de esta actividad hasta llegar a ver su representación. Este recorrido resulta innecesario. Una vez seleccionada una actividad se debería mostrar su contenido mediante alguna representación fácil de entender, como puede ser en forma lineal y, si se quiere editar, que se dé la opción a ello también posteriormente.

Una vez concluidas las preguntas, se comentaron posibles mejoras de este programa:

1. Eliminar el menú inicial que ha estado presente en los dos diseños planteados para simplificar el uso de la interfaz. Para ello:
   - Cuando se ejecute el programa se mostrarán las actividades que tiene almacenadas el sistema.
   - Si se desea consultar cualquier actividad, se hará doble clic sobre ella y se mostrará su contenido mediante un diagrama lineal que recoja los elementos que ésta contiene.
   - Si se quiere crear una nueva actividad, se tendrá que apretar un botón que represente esta opción mediante un icono (que será el mismo que tienen las actividades que ya se encuentran dentro del sistema). Para editarla, al igual que en el caso de consultar la actividad, se realizará un doble clic sobre ella. Una vez dentro se mostrarán botones que representen a las tareas y subactividades a añadir.

- Para ordenar las tareas y subactividades que componen una actividad, se arrastrará cada elemento a la posición deseada dentro del diagrama lineal que representa a la actividad.
- Para eliminar la actividad, bastaría con seleccionar el icono que la representa y, o bien mediante un menú secundario o bien pulsando la tecla suprimir, eliminarla del sistema.

2. En los dos diseños planteados realizados mediante Swing de Java, la estética de la aplicación resulta pobre. Por ello, se recomienda usar otro programa de diseño de interfaces que permita crear un diseño más atractivo.

   Hasta ahora las tareas podían ser de tipo: imagen, vídeo o descripción textual. Sin embargo, según los requisitos, faltaría un cuarto tipo de datos: el audio.

   Se comentó también que, cuando se elige que el contenido de la tarea sea de tipo vídeo, se solicita al usuario la dirección web del lugar donde se encuentra alojado dicho contenido. Para simplificar la inserción de esta información en el sistema se recomendó que el vídeo y el audio se encontrasen en el mismo ordenador del usuario en vez de en internet (al igual que las imágenes).
3. Dentro de cada actividad se debe dar la opción de ver una vista previa de toda la actividad. En esta opción se debe mostrar únicamente las tareas en el orden en el que serán vistas por los pacientes.
4. Por cada actividad el sistema debe de crear el código QR, cuestión que hasta ahora no se ha mostrado en los diseños implementados.
5. Otro de los requisitos es el poder personalizar las actividades. Por ello, por cada actividad se debe de poder personalizar, para cada paciente, el conjunto de tareas y subactividades que se quiera que éste realice. Para que sea posible esta opción hay que poder ingresar y borrar nuevos pacientes al sistema.
6. Una de las opciones que se tienen que seguir manteniendo, en cualquiera de las pantallas en las que se encuentre el usuario, es el menú de ayuda. En él se explicará paso a paso cómo funciona el programa para realizar todas las operaciones necesarias con las actividades.

4.2.4. Tercera Iteración

Con todas las cuestiones y comentarios planteados en la segunda iteración, se crea el diseño definitivo. En este caso, para su representación, se usó un nuevo programa llamado Balsamiq Mockups[5], ya que permite crear interfaces con una estética más cuidada que la librería Swing de Java y permitiendo además un ahorro de tiempo considerable.

La ejecución de este diseño, que el usuario vería usando el programa es la siguiente:

1. La primera pantalla que se muestra, una vez cargada la herramienta de autor, es la que corresponde con la siguiente imagen (Figura 4.29). En ella se puede ver, en la parte central, todas las actividades (representadas por un icono; en este caso se ha usado un libro) que existen en el sistema.

   Encima de estas actividades se encuentra el botón de agregar una nueva actividad y su *tooltip* asociado (que aparecería cuando se mueve el ratón encima de este botón). Para añadirla hay que pinchar sobre este icono y arrastrar la nueva actividad al panel que contiene el resto de actividades.

   Para acceder al contenido de las actividades que ya existían en el sistema así como para las que se crean de nuevo, basta con hacer doble clic sobre ellas.

   Además se puede observar, en la parte superior, una barra de navegación para conocer, en todo momento, la situación actual en la que se encuentra ubicado el usuario. Adicionalmente, a la izquierda del elemento que se acaba de describir, se encuentran las flechas de anterior y siguiente para facilitar la movilidad entre pantallas.

   A esta misma altura pero en el lado derecho de la pantalla aparece un botón cuyo icono (un *diskette*) hace referencia a la opción de guardar. Cuando la persona que está usando este programa aprieta este botón, los cambios son actualizados (se guardan) en el servidor donde se almacena toda la información. A partir de este momento, el paciente que ejecute el programa en el teléfono móvil verá el programa actualizado.

[5] http://balsamiq.com/products/mockups/

En la parte izquierda de la interfaz se muestra un menú de selección con cuatro posibles opciones; “Todo”, “Actividades”, “Tareas” y “Ayuda”. En este caso, (Figura 4.29) la opción que se encuentra seleccionada es “Todo”. Es por ello que esta opción muestra en forma de menú desplegable todo el contenido de este programa (las actividades, todas las tareas que se encuentran almacenadas en el sistema así como un menú de ayuda).

Por último, cabe señalar la barra superior de esta aplicación. En ella aparecen los iconos de cerrar, minimizar (nunca por debajo del tamaño mínimo que permita la correcta visualización del contenido) y maximizar (se adapta al tamaño de la pantalla que tenga el usuario) el programa.

A la izquierda de estos tres botones aparece el icono de ayuda (representado por una bombilla), que estará presente en todas las pantallas del programa. Este botón, al igual que la opción “Ayuda” del menú izquierdo, conducen al menú de ayuda donde se detalla todo el funcionamiento de esta herramienta.

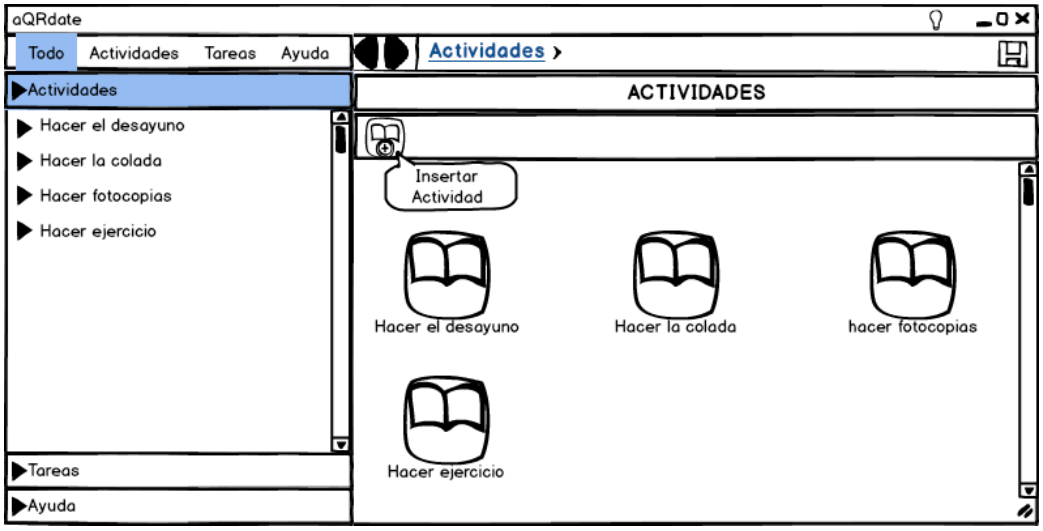


Figura 4.29: Tercer Prototipo. Pantalla de inicio.

2. Como se acaba de comentar en el primer paso de la ejecución de esta tercera iteración, para editar una actividad (modificar el contenido) hay que hacer doble clic sobre el icono que la presenta. En este caso, se selecciona la actividad “Hacer el desayuno”. El contenido de esta actividad se muestra en la Figura 4.30.

En esta pantalla se puede observar que:

- Esta actividad está compuesta por cinco subactividades (“preparar café”, “preparar zumo”, “preparar tostadas”, “calentar la leche” y “preparar café con leche”) representadas por el mismo icono que una actividad y por cinco tareas (“apagar la cafetera”, “desenchufar la cafetera”, “colocar la taza en la mesa”, “colocar en la mesa el vaso de zumo” y “colocar el plato de tostadas en la mesa”) cuyo icono, en este caso, es un folio.
- Otra forma de ver los elementos que tiene esta actividad es mediante el menú desplegable de la izquierda. En él se puede observar el nombre de las tareas y subactividades en el orden en el que se ha diseñado. Las subactividades en este menú, haciendo clic sobre ellas, se despliegan mostrando a su vez el contenido que las componen.
- Encima de los iconos de las subactividades y tareas que componen esta actividad, se pueden ver seis botones. Empezando por la izquierda, se muestra el botón de crear subactividades dentro de esta actividad y también el botón de insertar nueva tarea. Para ello hay que pinchar en estos iconos y arrastrarlos a la posición deseada dentro de la actividad (por ejemplo, si se quiere crear una subactividad después de la subactividad llamada “preparar café”, se arrastraría el ratón desde el botón de crear nueva subactividad hasta el hueco comprendido entre las subactividades “preparar café” y “preparar zumo”). El tercer botón tiene como función eliminar toda la actividad. El resto de botones (vista previa, personalizar y generar el código QR asociado a esta actividad) que aparecen es esta barra se explicarán más adelante.
- Para cambiar el nombre de esta actividad, se tiene que hacer doble clic sobre el nombre de la actividad (que se encuentra escrito en letras mayúsculas justo debajo de la barra de navegación del programa) y escribir el nombre deseado. Por defecto, las nuevas actividades creadas llevarán el nombre de “Nueva Actividad (x)”, donde x es el número que representa las nuevas actividades que han sido creadas.

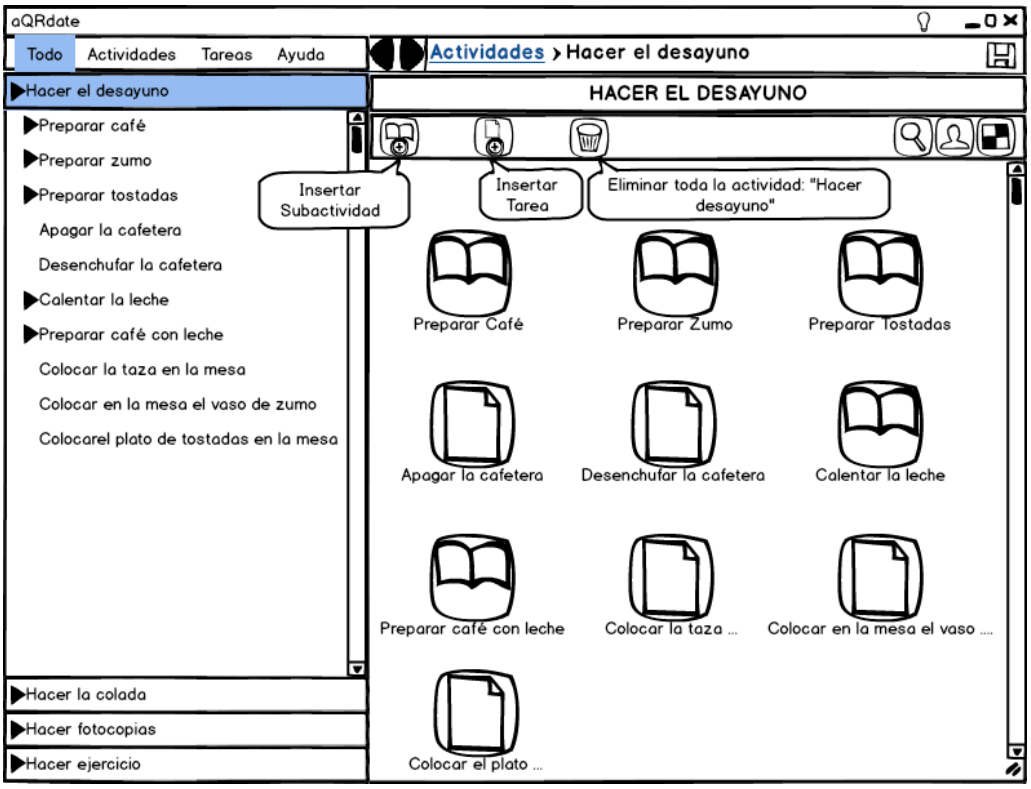


Figura 4.30: Tercer Prototipo. Contenido de la actividad "Hacer el desayuno".

3. Además se puede cortar, copiar y eliminar un objeto (actividad, subactividad o tarea) haciendo clic encima de éste con el botón secundario del ratón:

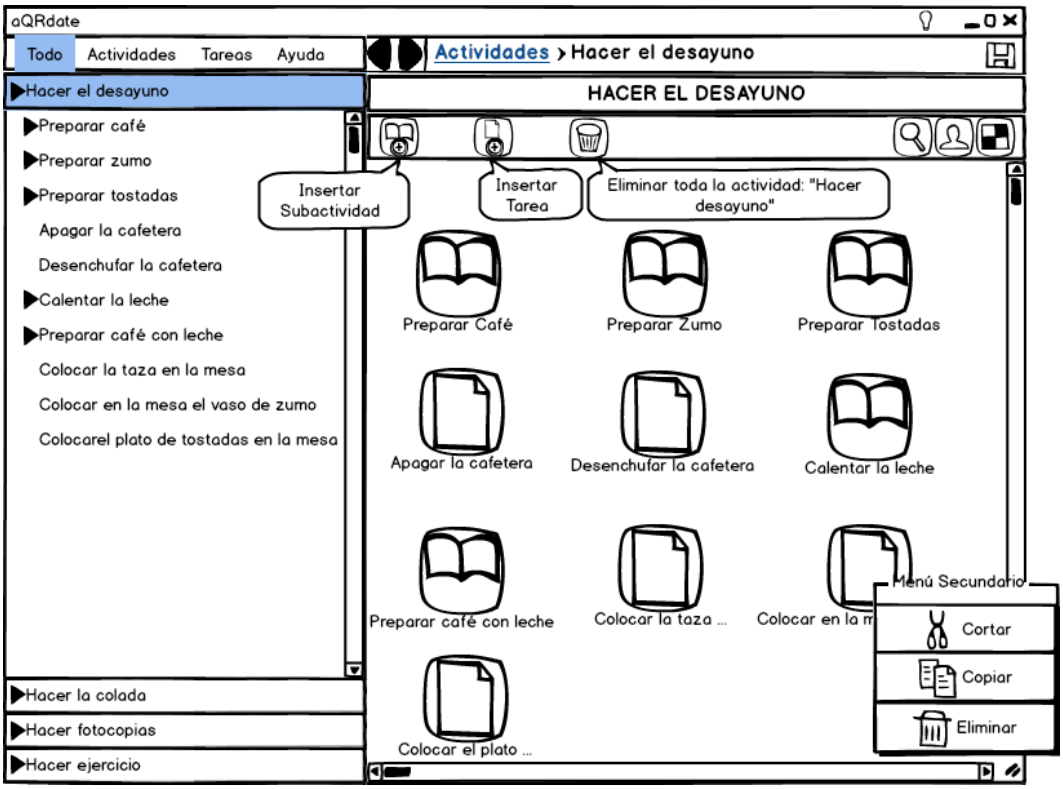


Figura 4.31: Tercer Prototipo. Menú secundario.

En ese momento aparece el menú secundario que se puede ver en la figura anterior (Figura 4.31). Para este ejemplo, se ha seleccionado la tarea “Colocar en la mesa el vaso de zumo”.

4. Al igual que para acceder al contenido de una actividad, para ver la información que contiene una tarea hay que pulsar dos veces seguidas el botón principal del ratón. Para ilustrar los cuatro tipos de tareas posibles (imagen, vídeo, audio y descripción textual) se explica a continuación un ejemplo de cada uno de ellos, todos pertenecientes a la actividad “Hacer el desayuno”.

   Para el caso de la tarea de tipo Imagen:

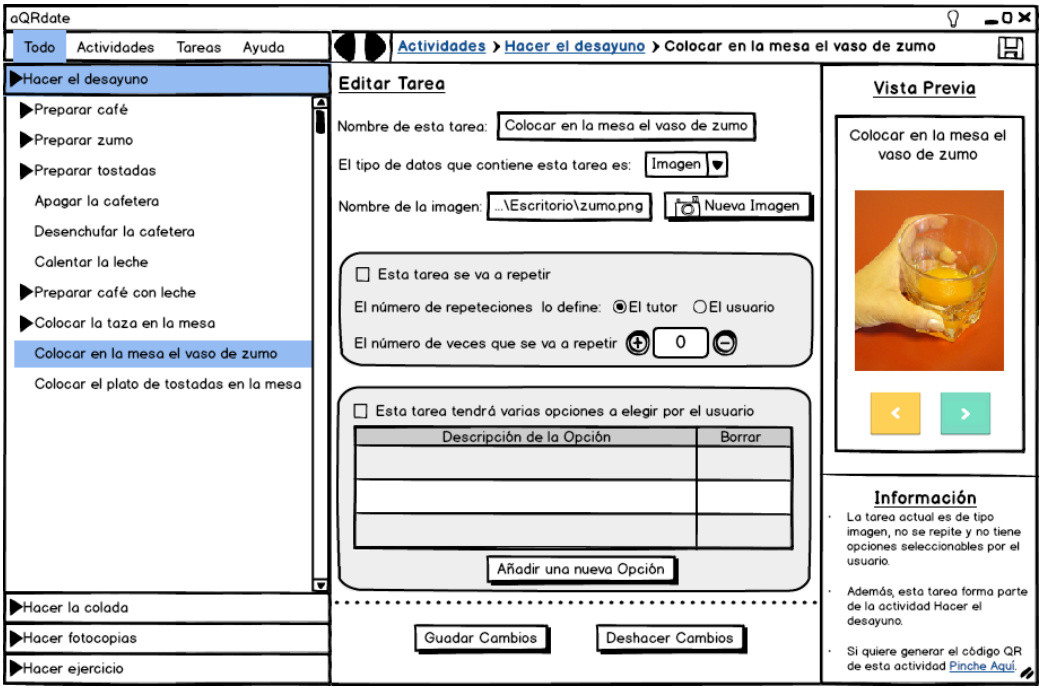


Figura 4.32: Tercer Prototipo. Tarea tipo imagen.

Esta tarea tiene por nombre “Colocar en la mesa el vaso de zumo” que ha sido previamente introducido en el cuadro de texto que se encuentra en la parte central de la pantalla. Después del nombre, el programa permite elegir el tipo de tarea. En este caso se ha escogido “Imagen” de la lista seleccionable. El siguiente campo para rellenar es el lugar en el que se encuentra la imagen que se quiera mostrar en esta tarea. Para llevar a cabo este paso se aprieta el botón que tiene por nombre “Nueva Imagen”. Una vez pulsado, emerge una pantalla para que se busque la foto en el ordenador. Cuando la foto ha sido

seleccionada, la dirección de este contenido aparece escrito en el cuadro de texto tal y como muestra la figura anterior (Figura 4.32).El resto de opciones (tareas repetitivas y opciones a elegir por el usuario), que se muestran debajo de los campos rellenados se explicarán más adelante.

Por último, se pulsa el botón de ”Guardar Cambios”. Es en ese momento cuando a la derecha se actualizaría la vista previa de esta tarea. Con ella el usuario se puede asegurar de que el contenido introducido es el correcto. Además, justo debajo de la vista previa, se incluye una información de la tarea que puede resultar de gran utilidad: se explica sus contenidos, se informa de la actividad a la que pertenece y se permite la generación del QR de la actividad.

Tarea cuyo contenido es un vídeo:

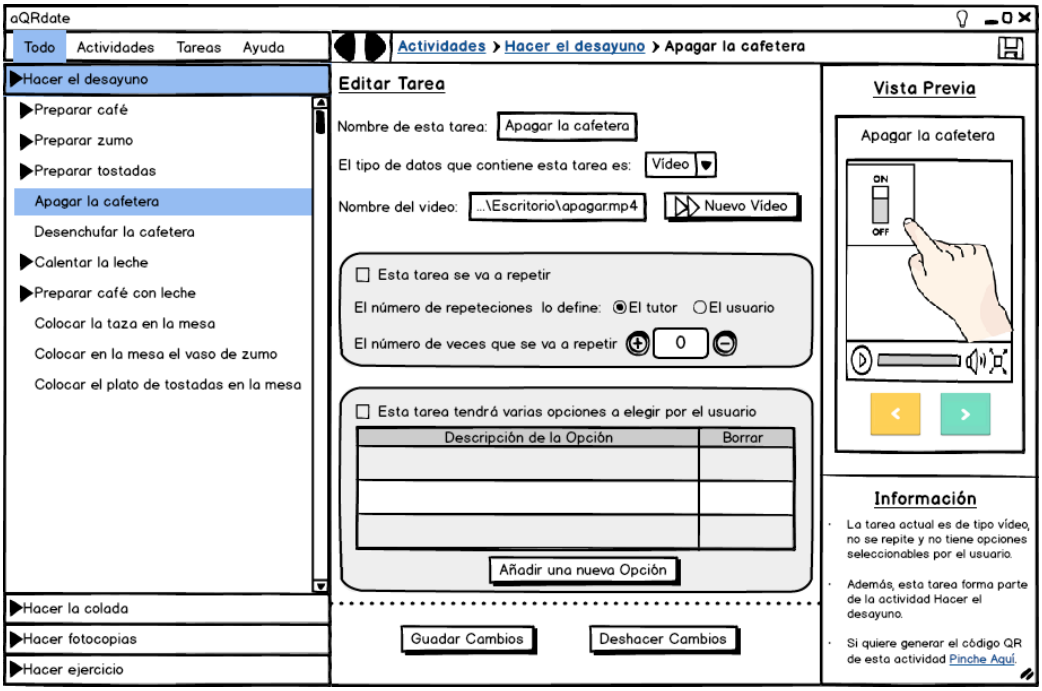


Figura 4.33: Tercer Prototipo. Tarea tipo Vídeo.

Para rellenar los campos iniciales (nombre de la tarea y tipo de datos), se procede de la misma forma que en el caso anterior. La diferencia radica en que ahora el tipo de datos seleccionado es un vídeo. Por tanto hay que pulsar sobre “Nuevo Vídeo” y buscar este contenido en el ordenador (y no como en las iteraciones pasadas en las que el programa pedía la dirección web del vídeo). La vista previa de esta tarea se actualiza una vez guardados los cambios y es en ese momento cuando, de forma automática, se reproduce el vídeo.

Tarea de tipo audio:

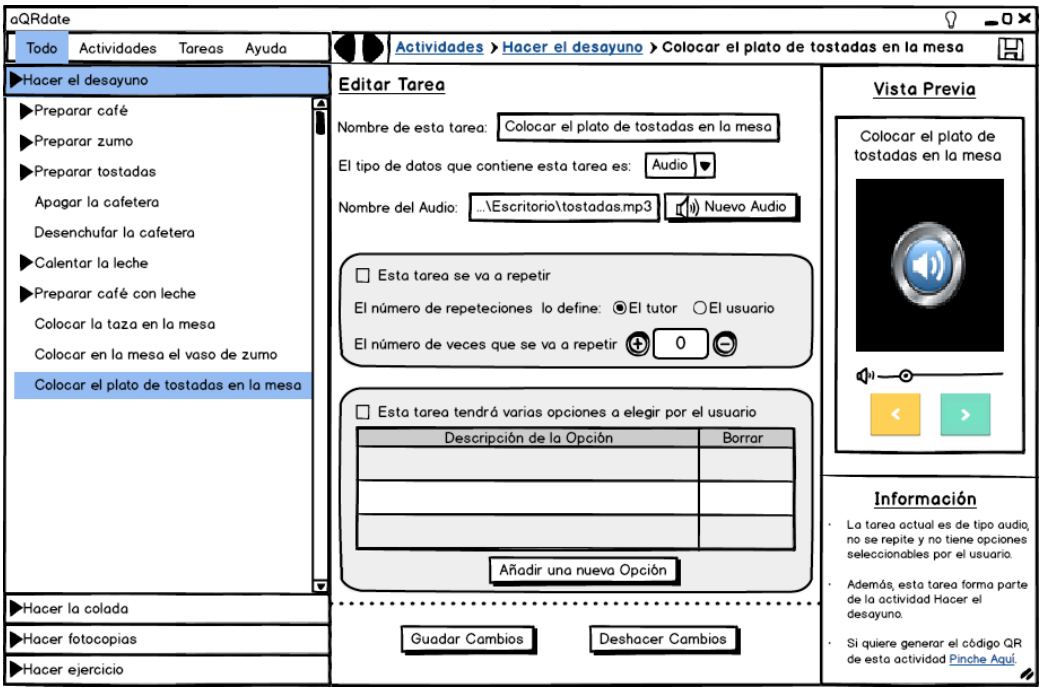


Figura 4.34: Tercer Prototipo. Tarea tipo Audio.

La única diferencia con la tarea tipo vídeo es que el archivo a introducir debe de ser de tipo audio. Cuando se guardan los cambios, se reproduce este contenido.

Tarea cuya información es una descripción textual:

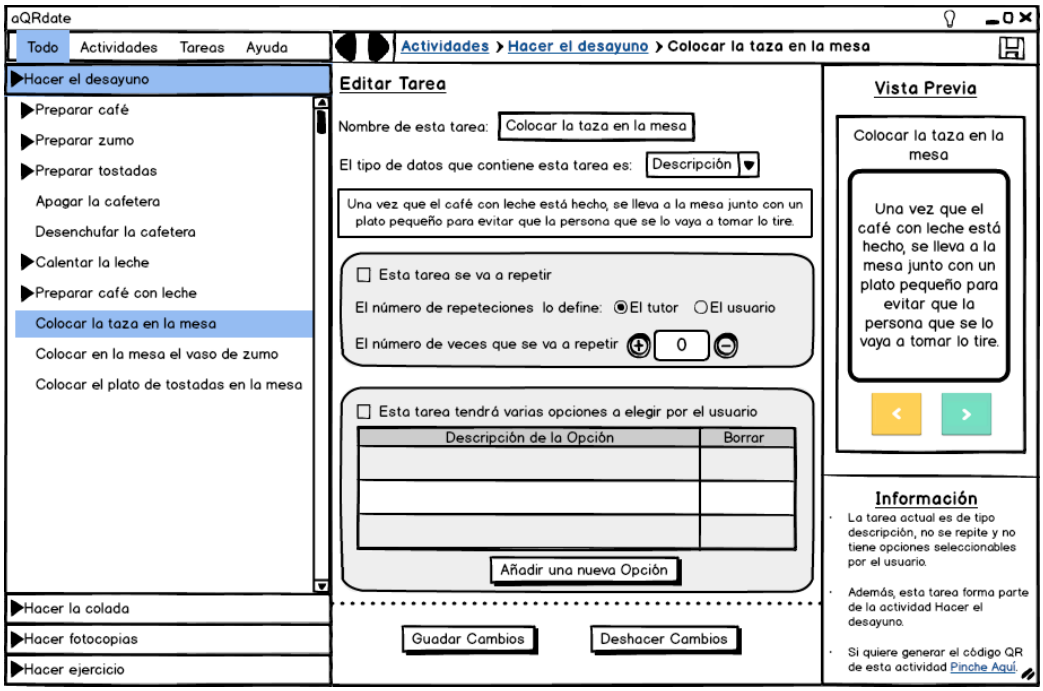


Figura 4.35: Tercer Prototipo. Tarea tipo descripción textual.

Es este caso, cuando se selecciona el tipo de datos, se muestra, justo debajo de esta lista de selección, un cuadro de texto para introducir la explicación que se desee expresar en esta tarea. Del mismo modo que con los tipos anteriores, cuando se pulsa el botón "Guardar Cambios", aparece reflejada esta información en la vista previa.

5. Como se ha visto en cada una de las tareas anteriores, a parte del tipo de tareas que se quiera seleccionar, se puede elegir que la tarea se repita (conocido como tarea repetitiva explicado en el apartado 3.6.1) o bien que la tarea tenga diferentes opciones y que el usuario elija una de ellas (llamado tarea de selección múltiple visto en la sección 3.6.2).

   Tarea repetitiva:

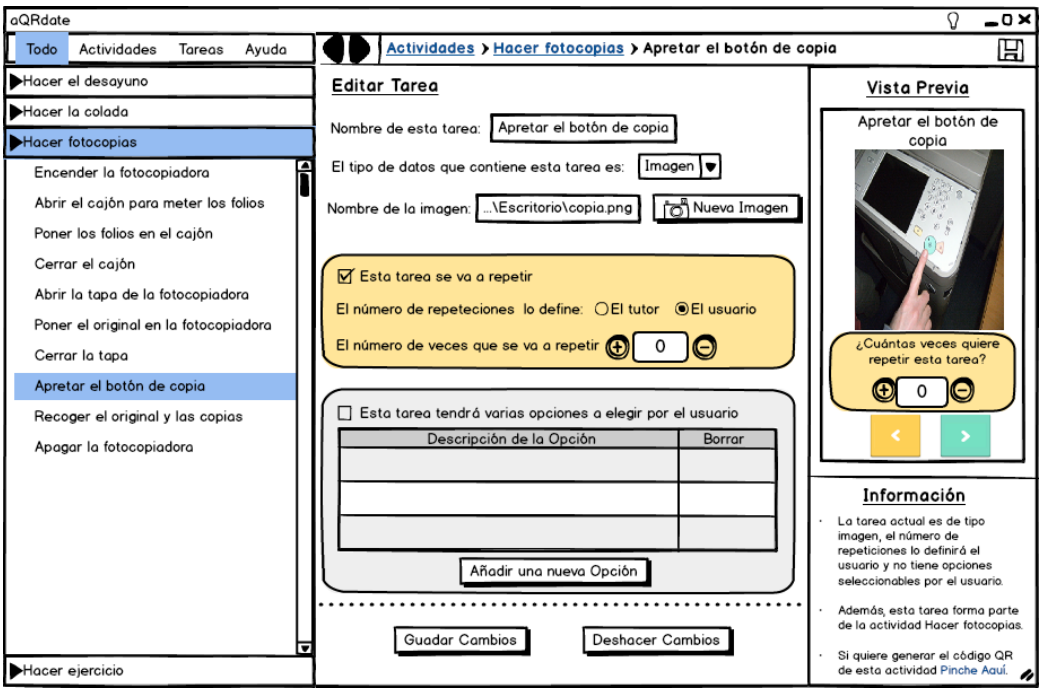


Figura 4.36: Tercer Prototipo. Tarea repetitiva.

Una vez marcada la tarea como repetitiva, hay que establecer la persona que va a fijar el número de repeticiones: el tutor o el usuario final desde el teléfono móvil (como en este ejemplo). Finalmente se introduce las veces que se quiera volver a realizar esta tarea en el caso de que sea el tutor quien fija esta cantidad o bien será el paciente durante la ejecución de la actividad.

Con motivo de significar que esta tarea llamada “Apretar el botón de copia”, que forma parte de la actividad “Hacer fotocopias”, se va a repetir, aparece en ella un icono que hace referencia a esta opción:

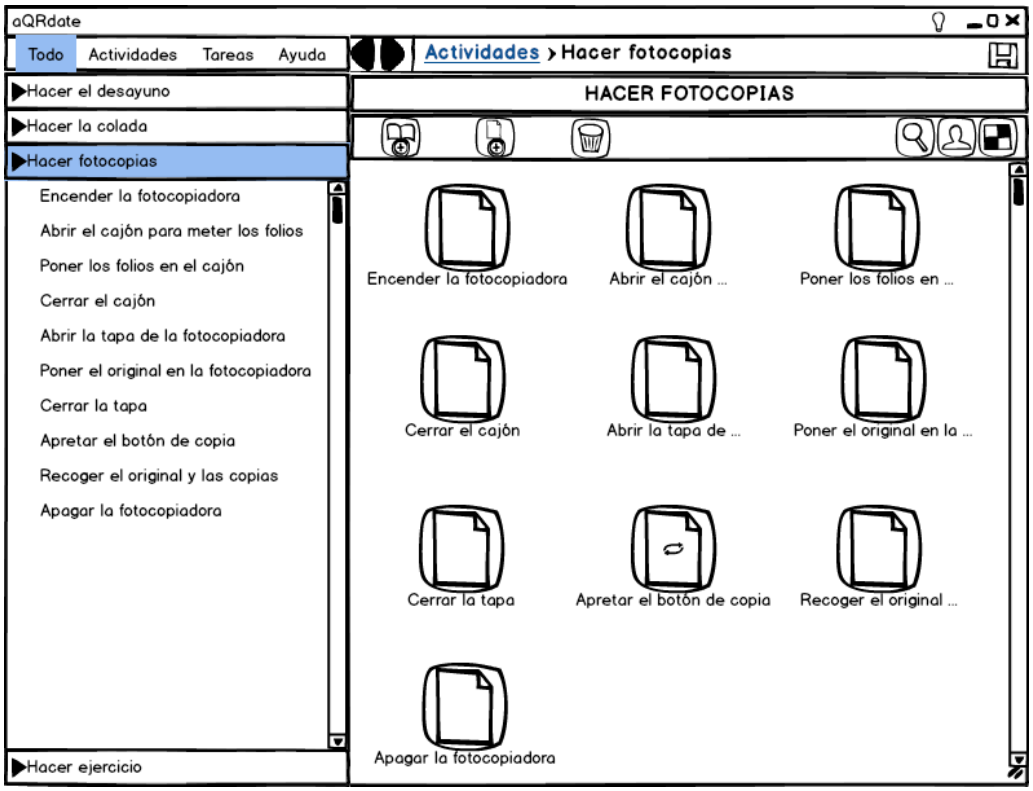


Figura 4.37: Tercer Prototipo. Icono de tarea repetitiva.

Tarea de selección múltiple:

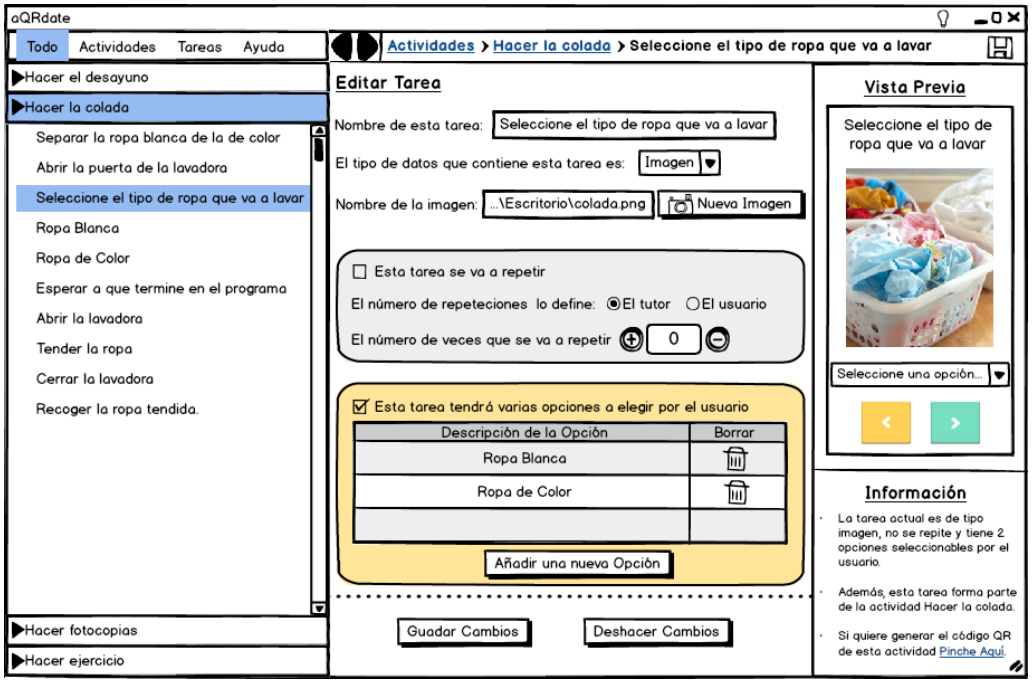


Figura 4.38: Tercer Prototipo. Tarea de selección múltiple.

En esta ocasión, para crear las opciones que posteriormente el usuario elegirá en esta tarea, se tiene que seleccionar el botón “Añadir una nueva Opción” (Figura 4.38). A continuación en una de las filas de la tabla aparecerá el mensaje de “Introduzca la descripción de esta opción”. En ella se tiene que pinchar e introducir el texto. Estos pasos se repetirán tantas veces como opciones se quieran mostrar.

En el ejemplo de la Figura 4.38 se puede ver que dentro de la tarea “Seleccione el tipo de ropa a lavar” se han creado dos opciones: “Ropa Blanca” y “Ropa de Color”. El usuario, una vez haya llegado a esta tarea durante la ejecución de la actividad “Hacer la colada”, tendrá que escoger una de esas dos opciones planteadas.

Dentro de la actividad, la creación de estas opciones se refleja de la siguiente manera:

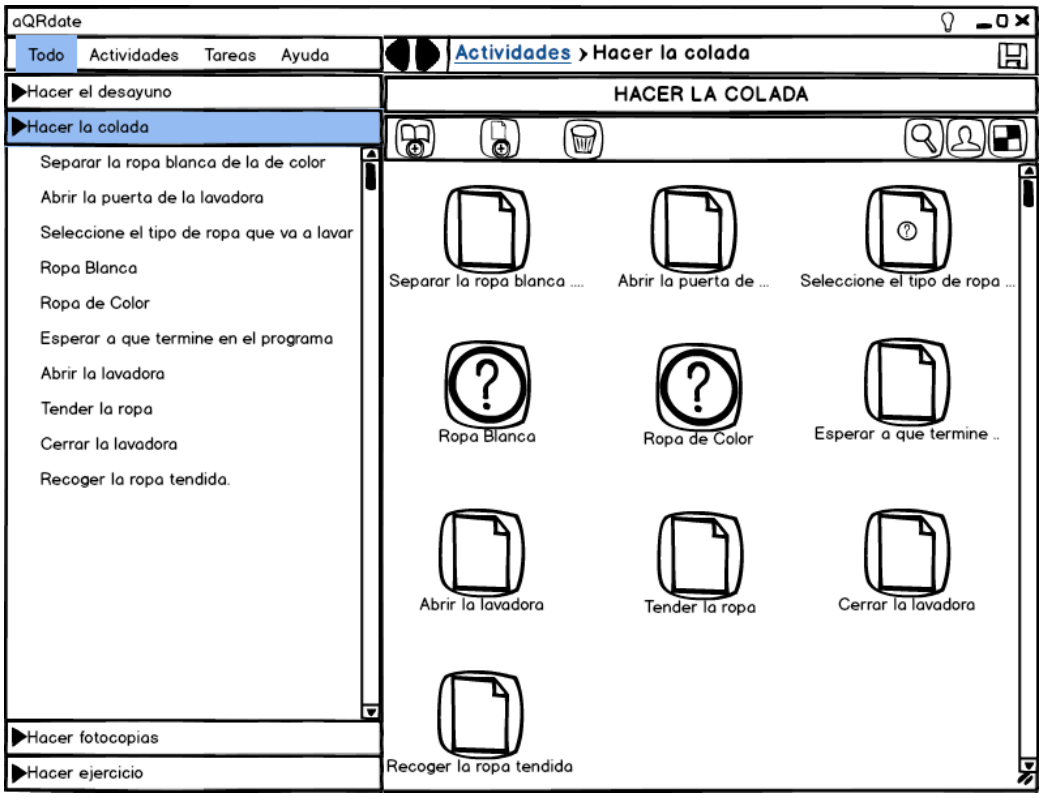


Figura 4.39: Tercer Prototipo. Icono de la tarea y opciones de selección.

Por una parte, aparece un icono dentro de la tarea para hacer referencia a la selección múltiple, similar al icono que aparece cuando la tarea es repetitiva (Figura 4.39).

Por otro lado se crean, a continuación de esta tarea, tantos iconos de selección como opciones se hayan creado (Figura 4.39). En ellos se puede introducir todas las subactividades y tareas que se deseen (a efectos prácticos cada opción se puede tratar como una subactividad). Sin embargo, el contenido que haya en su interior sólo será mostrado al usuario si éste escoge esa opción.

Durante la ejecución de la actividad por parte del paciente, una vez que se ha mostrado toda la información de la opción seleccionada, se continúa con el resto del contenido que haya en la actividad (en este caso se seguiría con la tarea "Esperar a que termine el programa").

6. Dentro del panel de las actividades, en la barra que contiene los botones, aparecen tres: vista previa, personalizar y generar el código QR de la actividad que aún no se ha detallado su funcionamiento. En la siguiente figura se pueden observar estos botones con sus *tooltips* (textos emergentes que aparecen cuando se desliza el ratón encima de ellos) correspondientes.

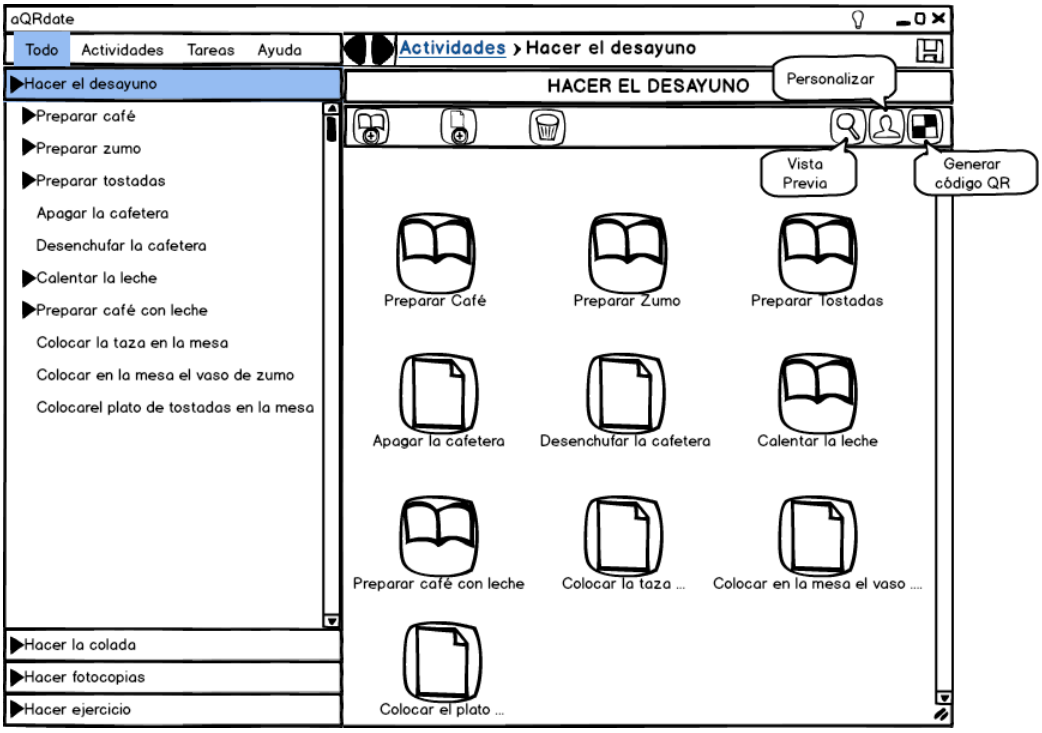


Figura 4.40: Tercer Prototipo. Interfaz de la herramienta de autor.

Vista Previa

Esta opción muestra, tal y como se vería en el teléfono móvil del paciente, el contenido de la actividad tarea a tarea. Para realizar la navegación entre tareas, el usuario dispone de flechas de navegación (flechas de color naranja y azul en la Figura 4.41). También existe la posibilidad de escoger una opción en el caso de una tarea de selección múltiple así como de indicar el número de repeticiones que se quiere realizar en el caso de que sea una tarea repetitiva cuya cantidad debe ser especificada por el usuario.

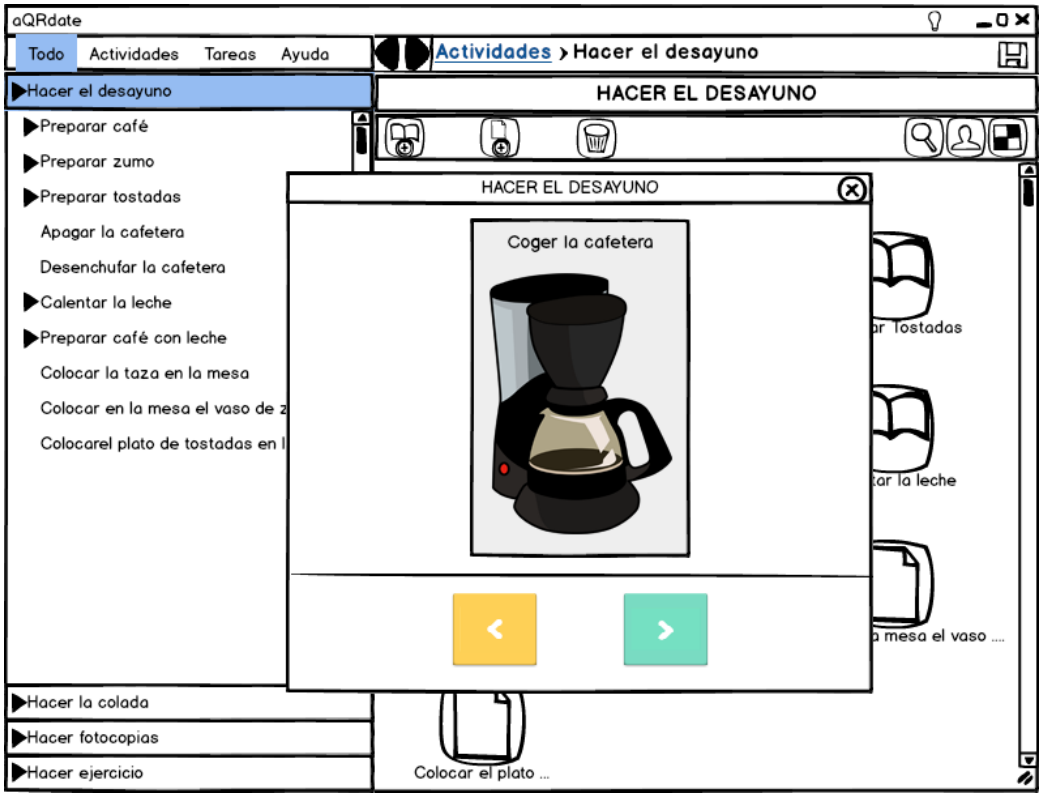


Figura 4.41: Tercer Prototipo. Vista Previa.

Personalización

Por personalizar se entiende eliminar, para un paciente concreto, uno o varios pasos (tareas o subactividades) de la actividad estándar. Por tanto, lo primero que debe permitir el sistema es añadir y eliminar pacientes por cada actividad.

En la figura que se muestra a continuación se puede ver la interfaz que se muestra cuando se selecciona la opción de personalizar. A la derecha se encuentran los usuarios registrados para esta actividad y el llamado “Usuario genérico” el cual contiene todos los pasos de la actividad.

Para introducir un nuevo paciente al sistema, hay que pulsar el nuevo botón que aparece en esta opción por primera vez: “Nuevo usuario".

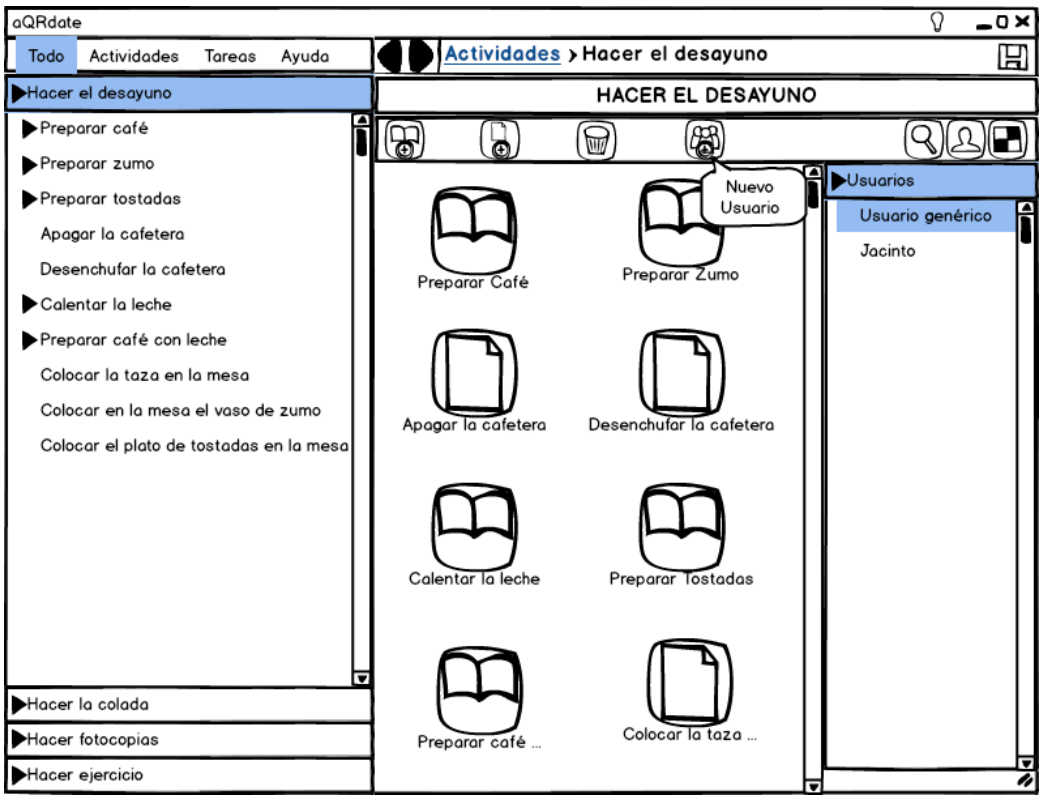


Figura 4.42: Tercer Prototipo. Personalizar la actividad.

Una vez creado y seleccionado, el resultado es el siguiente:

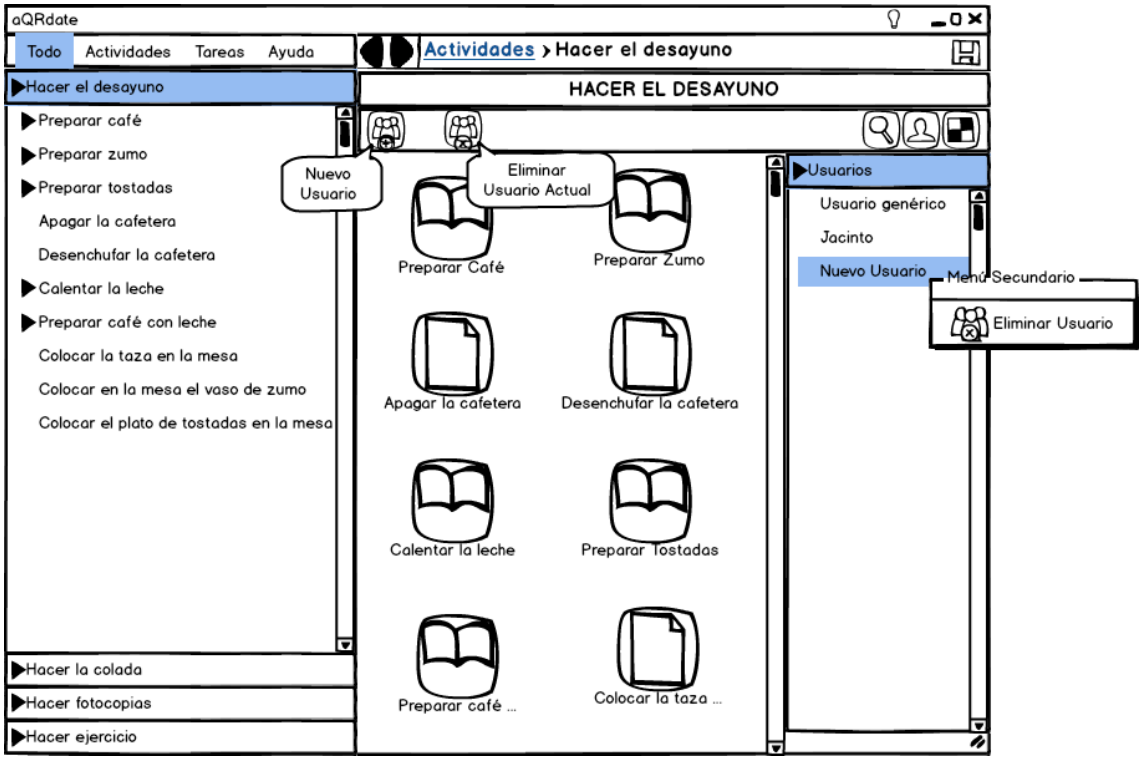


Figura 4.43: Tercer Prototipo. Introduciendo usuarios en la opción de personalizar.

Para cambiar el nombre al usuario, se hace doble clic consecutivo sobre el nombre (en el caso de que se acabe de crear pondrá “Nuevo Usuario”). Ahora, de la parte central de la pantalla, se pueden eliminar aquellas tareas o subactividades que no se quieran mostrar a este usuario.

Por último, para eliminar a un paciente se aprieta el botón secundario del ratón encima de éste. En el menú secundario emergente se selecciona la opción de “Eliminar Usuario” (Figura 4.43). Otra opción es pulsar el botón que aparece en la barra superior cuyo *tooltip* es “Eliminar Usuario Actual” (viene reflejado en la imagen anterior).

Generación de código QR de la actividad

Para poder ver una actividad en el teléfono móvil, como se vio en el capítulo anterior, es necesario que el paciente apunte con la cámara al código QR. Para que el personal médico pueda obtener estos códigos, la herramienta de autor los tiene que generar de manera sencilla.

Una vez seleccionada esta opción, se elige el nombre que va a tener la imagen que contenga el QR así como su ubicación en el ordenador:

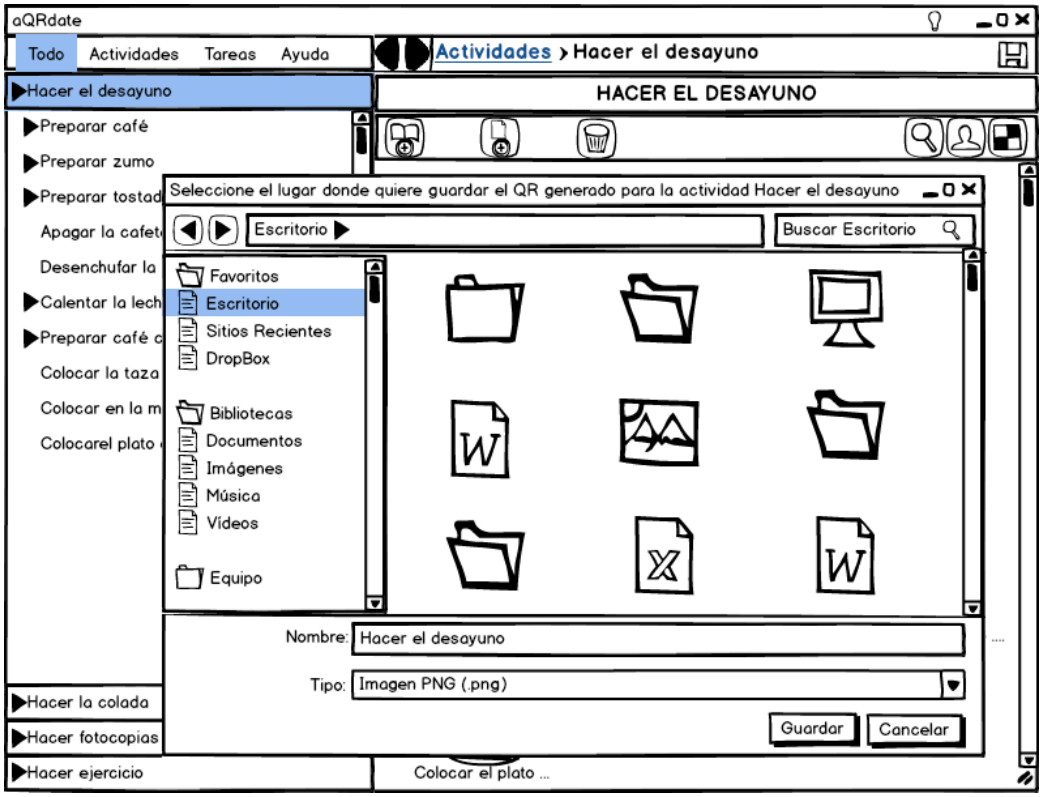


Figura 4.44: Tercer Prototipo. Generación del código QR de la actividad.

La imagen que se obtiene es la siguiente:

Figura 4.45: Tercer Prototipo. Código QR generado.

7. Para acabar la ejecución de esta tercera iteración, se detalla el diseño del menú de ayuda. Para acceder a este menú, se puede hacer pulsando encima del botón con forma de bombilla que se encuentra situado en la parte superior al lado de los botones de cerrar, minimizar y salir de la aplicación, o bien seleccionado "Ayuda" en el menú izquierdo.

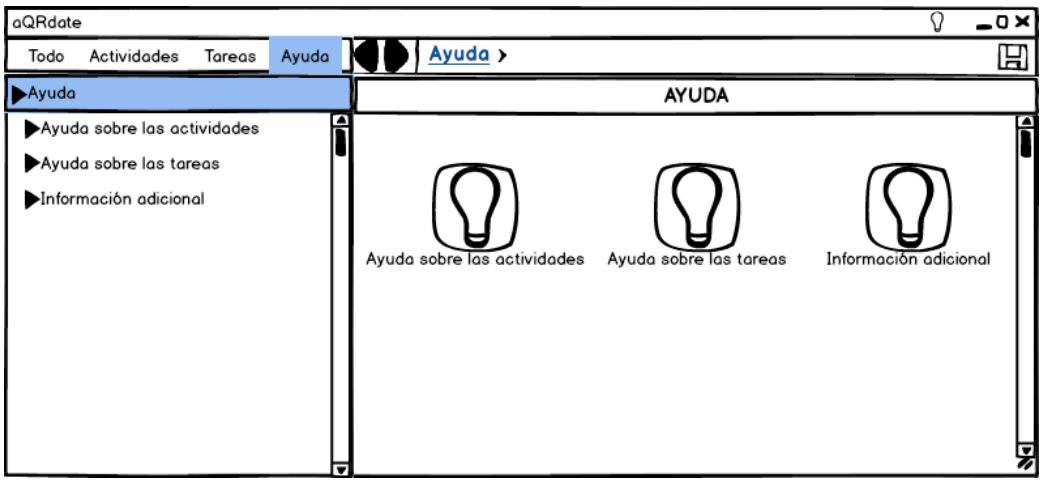


Figura 4.46: Tercer Prototipo. Menú de ayuda al usuario.

Este menú de ayuda al usuario contiene tres grupos (Figura 4.46): "Ayuda sobre las actividades", "Ayuda sobre las tareas" e "Información adicional". A su vez, cada uno de ellos, contiene varios contenidos en los que se explica las funcionalidades de este programa.

Un ejemplo de la información que contiene un grupo (en este caso “Ayuda sobre las actividades”) se muestra en la siguiente figura:

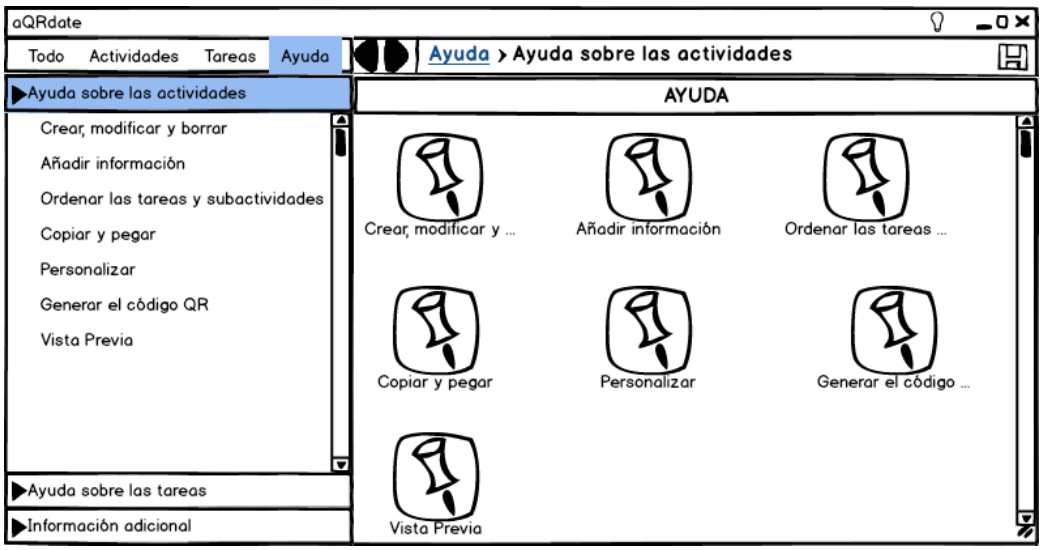


Figura 4.47: Tercer Prototipo. Ejemplo dentro del menú de ayuda al usuario.

Una vez concluida la presentación de esta tercera iteración al grupo de expertos, se dio por finalizada la fase de diseño de la aplicación ya que en esta ocasión el tercer prototipo cumplía todos los requisitos planteados.

5

aQRdate: HERRAMIENTA DE AUTOR. IMPLEMENTACIÓN

Una vez que han sido establecidos los objetivos, requisitos y diseño que ha de tener la herramienta de autor, se realiza la implementación del proyecto. En este capítulo se describe la creación del prototipo desde dos puntos de vista diferentes: la arquitectura del sistema y la interacción del usuario.

El primero de ellos explica el funcionamiento y composición de la aplicación mediante diagramas de bloques. En especial se centra en los módulos que conforman el menú que existe en la pantalla de inicio con tres opciones: “Actividades”, “Tareas” y “Ayuda” (este menú se explica en la sección 4.2.4), en la barra de navegación de la aplicación, en la conversión de actividades del sistema a código de entidades y en la conexión que se establece con el servidor. Finalmente se explican los elementos gráficos que han sido usados para crear la interfaz del usuario y se incluye un ejemplo de la hoja de estilo (CSS) utilizada para modificar la apariencia de los mismos.

El segundo describe cómo es la interacción que el usuario tiene con esta herramienta paso a paso. Para ello se explica, pantalla a pantalla, seis ejemplos tales como: crear una actividad, editarla, personalizarla, visualizar su vista previa, generar el código QR asociado a ésta y finalmente se muestra el resultado de navegar por alguno de los contenidos del menú de ayuda.

Antes de profundizar en cada uno de estos puntos, este capítulo, en su primera sección, comienza hablando del lenguaje escogido para el desarrollo esta herramienta: JavaFX. En este primer punto se define, a grandes rasgos, este lenguaje basado en Java así como la justificación de la elección del mismo.

5.1. JAVAFX Y NETBEANS

Como se acaba de comentar, se utiliza JavaFX para la creación de la herramienta de autor. En este apartado, se justifica su uso y se da a conocer las funcionalidades que tiene el lenguaje.

5.1.1. Justificación del lenguaje escogido

Inicialmente, este proyecto había sido diseñado para realizarse en el lenguaje de programación Java ya que las librerías encargadas de realizar la conexión con el servidor estaban creadas en este lenguaje (para más información acerca del servidor consultar la sección 3.3).

Sin embargo, al empezar a hacer los diseños usando la librería gráfica Swing de Java se vio que éstos resultaban bastante pobres en cuanto a la estética de la interfaz: las pantallas que se crean vienen definidas por defecto sin apenas poder realizar cambios y los elementos como pueden ser botones, listas, menús, etc. no pueden prácticamente modificar su estilo. Estas limitaciones hacían que la herramienta de autor fuera algo menos cómoda de desarrollar, haciendo que programar la misma pudiera llevar más tiempo del estimado con anterioridad.

Debido a este problema se pensó en usar otros lenguajes como puede ser Flash de Adobe, pero el problema de éstos era aún peor que el anterior: como se ha comentado anteriormente las librerías que se utilizan para la conexión con el servidor estaban creadas en Java y, para usar este tipo de lenguajes, habría que realizar bastantes cambios o desarrollar nuevas para que fueran compatibles.

Por todo ello, se optó por un lenguaje que pudiera trabajar sin problemas con las librerías que fueron diseñadas dentro del proyecto aQRdate. En este caso, se seleccionó una de las últimas tecnologías de Sun Microsystems para el desarrollo de aplicaciones RIA: JavaFX. Este lenguaje permite crear interfaces con una estética muy cuidada y altamente definible por el programador, además de atractiva al usuario. Asimismo, al estar basado en Java, es totalmente compatible con programas y librerías que hayan sido creadas en este mismo lenguaje.

5.1.2. Definición de JavaFX

JavaFX es una familia de productos y tecnologías de Sun Microsystems basados en Java para el desarrollo de RIAs (*Rich Internet Applications*), aplicaciones web que tienen características y capacidades similares a las aplicaciones de escritorio, incluyendo aplicaciones multimedia interactivas. Este producto es la contrapartida de Sun para las herramientas de desarrollo de este tipo de aplicaciones, como Flash de Adobe y Silverlight de Microsoft. Fue presentado en mayo de 2007 y lanzado oficialmente la versión 1.0 en diciembre de 2008, luego se trata de un lenguaje relativamente nuevo [46].

Las características generales que presenta JavaFX son [47]:

- Permite el desarrollo de interfaces visuales para escritorio, web, dispositivos móviles y TV bajo un perfil común que permite utilizar la misma aplicación en todos estos ambientes sin la necesidad de reescribir código.
- JavaFX utiliza un lenguaje declarativo que permite describir fácilmente qué debe hacer la aplicación y cómo debe ser su aspecto gráfico.
- Compatibilidad con la tecnología Java: Permite el uso de cualquier librería Java desde una aplicación JavaFX.
- Otro de los rasgos a destacar es que también favorece el trabajo conjunto de desarrolladores y diseñadores con una serie de herramientas que permiten crear diseños tan complejos como atractivos de una forma sencilla y coordinada.
- Mercado más amplio: Distribuir RIAs fácilmente a través de miles de millones de dispositivos valiéndose de todo el poder de la Plataforma Java. Aprovecha por tanto la extrema ubicuidad, el poder y la seguridad de la Máquina Virtual Java (JVM).

Con respecto al diseño de interfaces gráficas (GUIs), las características principales que se pueden destacar son [47] [48]:

- Nuevo acelerador gráfico.

- Nuevo *toolkit* para la gestión de ventanas.
- Este lenguaje contiene una amplia variedad de librerías para manejar contenidos multimedia tales como: vídeo, audio e imagen en formatos populares (ejemplos de ellos son: MP3, MP4 o JPG), así como texto enriquecido con una gran calidad. También contiene librerías para hacer uso de dispositivos multi-touch (como tabletas).
- Las animaciones son creadas utilizando líneas de tiempo (*timeline*), algo bastante parecido a lo que usa Flash.
- Soporte a alto nivel de maquetación para hacer la creación de gráficos más simple: sombras, degradados, difuminados, reflexiones, transformaciones 2D y 3D.
- La forma y el estilo de todos los componentes de la interfaz son fácilmente personalizables mediante el uso de las hojas de estilo en cascada (CSS).

Para ejecutar las aplicaciones realizadas en este lenguaje en cualquier ordenador, al igual que en Java, se necesitará tener instalado el JRE (*Java Runtime Enviroment)* que es un conjunto de utilidades que permite la ejecución de estos programas, entre ellos la JVM (máquina virtual de Java). La versión usada de JavaFX es la 2.2.3.

En el Anexo C se explica, a grandes rasgos, la creación de una interfaz usando JavaFX mediante un sencillo ejemplo.

5.1.3. Entorno de desarrollo (IDE): Netbeans

JavaFX dispone de soporte para el desarrollo de aplicaciones en los dos entornos de desarrollo integrado más importantes: Eclipse y Netbeans. Sin embargo, se opta por el uso de Netbeans pues este entorno era conocido por el alumno al haber sido utilizado previamente y porque en la mayoría de libros consultados sobre programación en JavaFX se recomendaba [48] [49] [50] .

La versión usada en este trabajo es: Netbeans IDE 7.2.1.

5.2. DESCRIPCIÓN DE LA ESTRUCTURA

En este apartado se va a describir la aplicación creada desde un punto de vista estructural. De esta manera, se parte de un esquema general de la interfaz (Figura 5.1) y, a partir de él, se definen con más detalle los diferentes bloques que conforman esta herramienta.

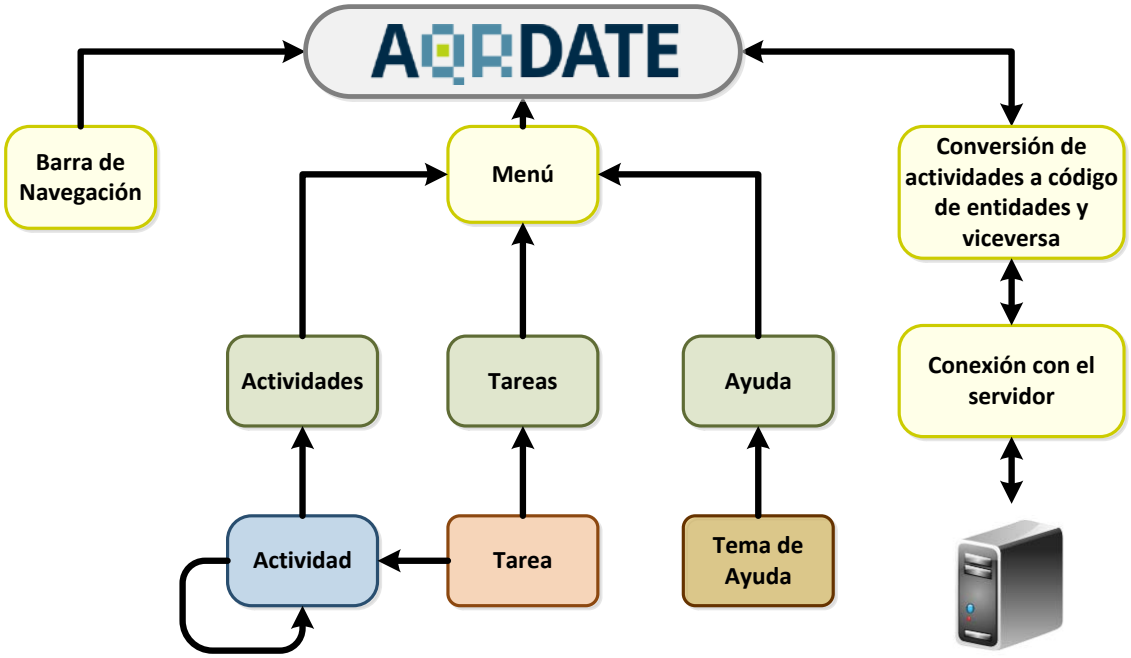


Figura 5.1: Esquema de la interfaz gráfica.

Como se aprecia en la figura, la aplicación de usuario está formada por tres bloques principales:

- Barra de navegación: Con ella se podrá retroceder y avanzar entre pantallas. Consiste en un sencillo historial que simplemente almacena las actividades, tareas y ayuda que el usuario ha consultado.
- Menú: Es la parte principal del programa. En función de la opción escogida se muestra el listado de las actividades, de tareas o el menú de ayuda. En este apartado se explicará también los módulos llamados “Actividad” y “Tarea” puesto que son parte esencial del programa.
- Conexión con el servidor y conversión de las actividades al archivo de entidades: Su trabajo consiste en cargar todos los datos de las actividades al inicio así como de guardarlas en el servidor cuando el usuario lo desea o cuando se cierra la aplicación.

Además, en la última sección de este apartado se explican los elementos gráficos de JavaFX que han sido utilizados y se incluye un pequeño ejemplo de código usado en la hoja de estilo en cascada (CSS) para personalizarlos.

5.2.1. Barra de navegación

Como se ha comentado anteriormente, esta barra de navegación permite al usuario moverse por el programa navegando entre pantallas que ya ha consultado anteriormente. Este historial tiene una estructura de datos cuyo modo de acceso a los elementos es de tipo LIFO (*Last In First Out*, último en entrar, primero en salir), más conocida como Pila. Su esquema de funcionamiento es el siguiente:

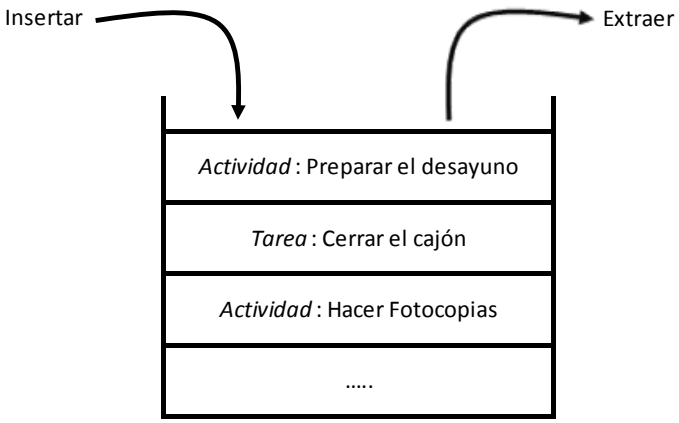


Figura 5.2: Esquema de la barra de navegación. Estructura de datos tipo Pila.

Este módulo se encarga de almacenar las pantallas por las que el usuario ha ido navegando. Para el ejemplo de la Figura 5.2, el usuario ha visitado justo antes de la pantalla en la que se encuentra la actividad “Preparar el desayuno” y previamente a ésta consultó la tarea “Cerrar el cajón”.

Cabe destacar en este punto que, cuando se retrocede usando la barra de navegación, es necesario guardarse en otra estructura de datos en forma de Pila, las pantallas a las que se va regresando pues luego éstas serán necesarias para cuando el usuario desee avanzar para situarse en el lugar en el que se quedó antes de retroceder.

Adicionalmente a estas flechas de desplazamiento (flechas de anterior y siguiente), en esta misma barra de navegación se muestra un *breadcrumb* que consiste en una línea de texto en la que se indica el recorrido seguido (en este caso entre actividades, tareas

y opciones de ayuda) y la forma de regresar. De esta forma, se da a conocer al usuario el punto en el que encuentra dentro de la ruta así como la posibilidad de volver a cualquier punto de ella pinchando sobre un elemento del recorrido mostrado. Un ejemplo de este tipo de contenido es la siguiente figura:

Figura 5.3: Ejemplo de *breadcrumb* en la página web de la Universidad Autónoma de Madrid.

Para poder crear el *breadcrumb* se tiene que obtener la ruta completa de la actividad o tarea se quiere consultar y finalmente se divide en tantos elementos como tenga la misma (en el ejemplo anterior, el resultado de la división de la ruta daría como resultado: “Inicio”, “La UAM”, “Tecnologías de la Información”). Cada uno de ellos forma parte del *breadcrumb* creado y pueden ser seleccionados por el usuario.

5.2.2. Menú

Es la parte central del programa. Este menú, como se puede ver en la Figura 5.1, está formado por tres opciones: Actividades, Tareas y Ayuda (en realidad hay una cuarta opción que engloba a todas ellas llamado “Todo” pero que no se incluye en la figura para simplificar el esquema y mostrar una imagen más clara de éste). En función de la opción que escoja el usuario se mostrará una u otra.

ACTIVIDADES

Esta opción contiene todas las actividades que hay en el sistema. Por tanto cuando esta opción es escogida, muestra todas las posibles actividades que se pueden consultar. El esquema de bloques que representa a esta opción es el siguiente:

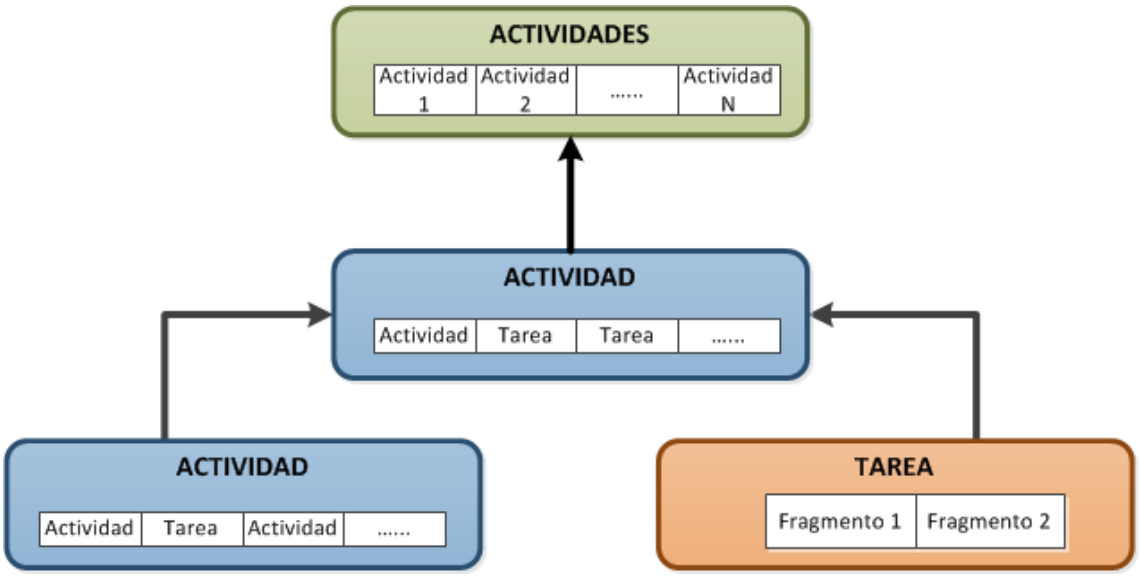


Figura 5.4: Esquema de bloques de la opción Actividades.

En el esquema se muestra que esta opción del menú contiene, como se acaba de comentar, las actividades y que éstas a su vez pueden estar compuestas por una o varias actividades (también llamadas subactividades) y/o tareas (para más información acerca de los contenidos de las actividades, consultar el apartado 3.4 y siguientes).

TAREAS

En esta opción se almacenan todas las tareas. El esquema sería:

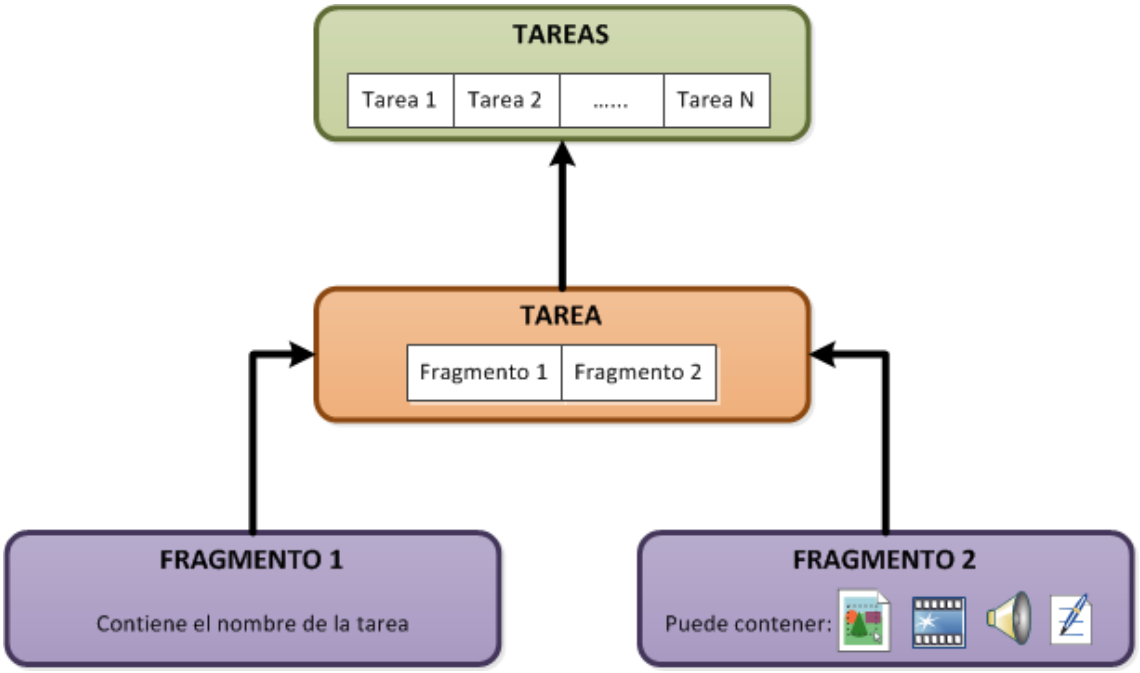


Figura 5.5: Esquema de bloques de la opción Tareas.

Es decir, esta opción contiene todas las tareas (que forman parte de las distintas actividades). A su vez, cada tarea está formada por dos fragmentos: el primero que contiene el título de la tarea con un tamaño no superior a 70 caracteres y el segundo que puede contener una imagen con formato .png o .jpg, un vídeo con formato .mp4, un audio con formato .mp3 o una descripción textual de un tamaño inferior a 650 caracteres.

AYUDA

Es la última de las tres opciones que tiene el menú. En su interior contiene, ordenados por categorías, los temas sobre los cuales el usuario puede consultar la información que necesite.

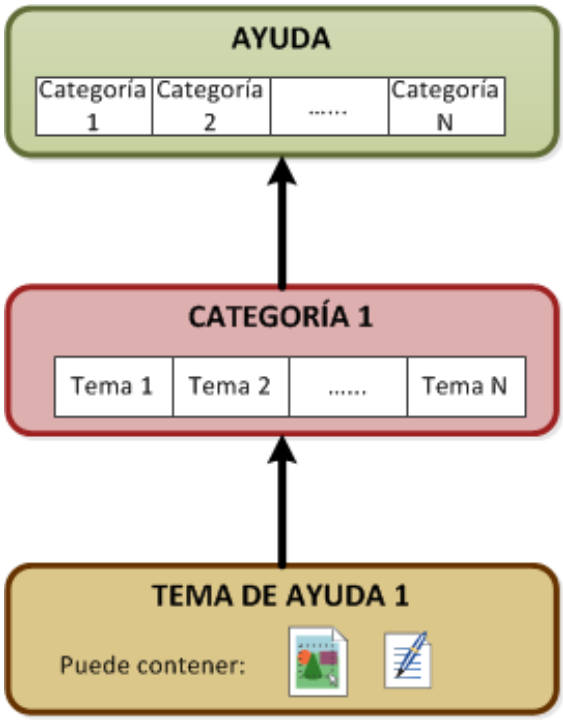


Figura 5.6: Esquema de bloques de la opción Ayuda.

En este caso, el menú de ayuda dispone de tres categorías: “Ayuda sobre Actividades” que contiene 7 temas, “Ayuda sobre tareas” formada 4 temas e “Información Adicional” que posee 2 temas. Cada uno de los temas explica algún aspecto del funcionamiento de la herramienta de autor (relacionado, obviamente, con la categoría del que forma parte). El contenido de los mismos está formado por textos e imágenes explicativas.

5.2.2.1. Actividad

El módulo encargado de trabajar con las actividades tiene la siguiente estructura:

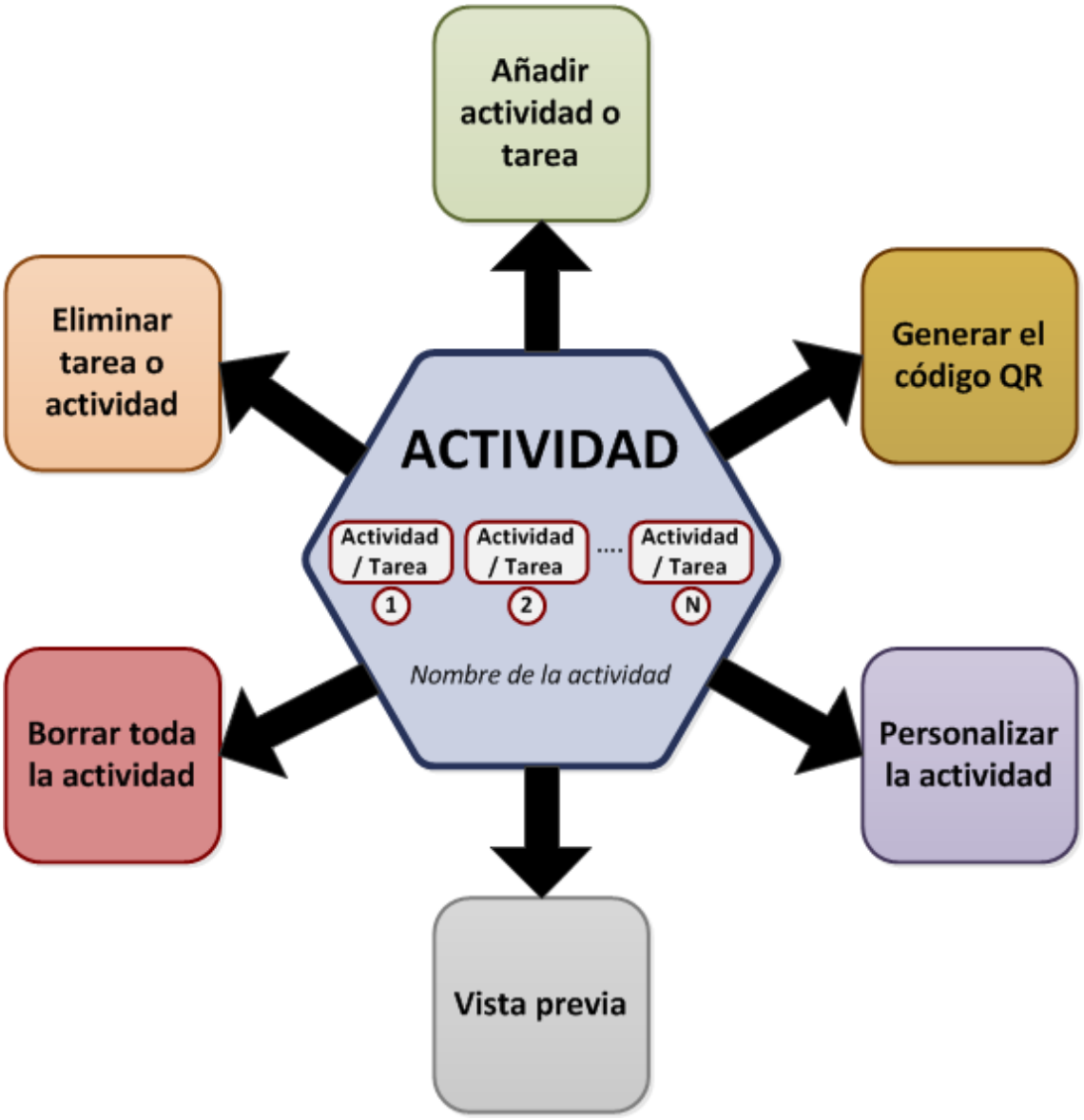


Figura 5.7: Esquema del bloque encargado de la gestión de cada Actividad.

En primer lugar se puede observar que cada actividad tiene un nombre que puede ser modificado por el usuario en cualquier momento.

También posee una lista ordenada de todos los elementos que la conforman, pueden ser tanto tareas como actividades (también llamadas en esta memoria subactividades). Cuando el usuario desea cambiar el orden de estos objetos, simplemente se cambia el elemento de posición. Si lo que se desea es borrarlo, este elemento es eliminado y la lista se desplaza hacia delante rellenado el lugar de la actividad eliminada.

A continuación se detalla el funcionamiento de cada uno de los seis bloques que permiten realizar diferentes acciones con las actividades (Figura 5.7):

Añadir Actividad (subactividad) o tarea

Para completar el contenido de una actividad puede ser necesario añadir una o más actividades o tareas. También puede darse este caso porque se haya copiado un nuevo elemento (actividad o tarea) a la actividad. Para ello, el único cambio que se realiza es introducir en la lista de objetos, la nueva actividad en aquella posición que el usuario desea.

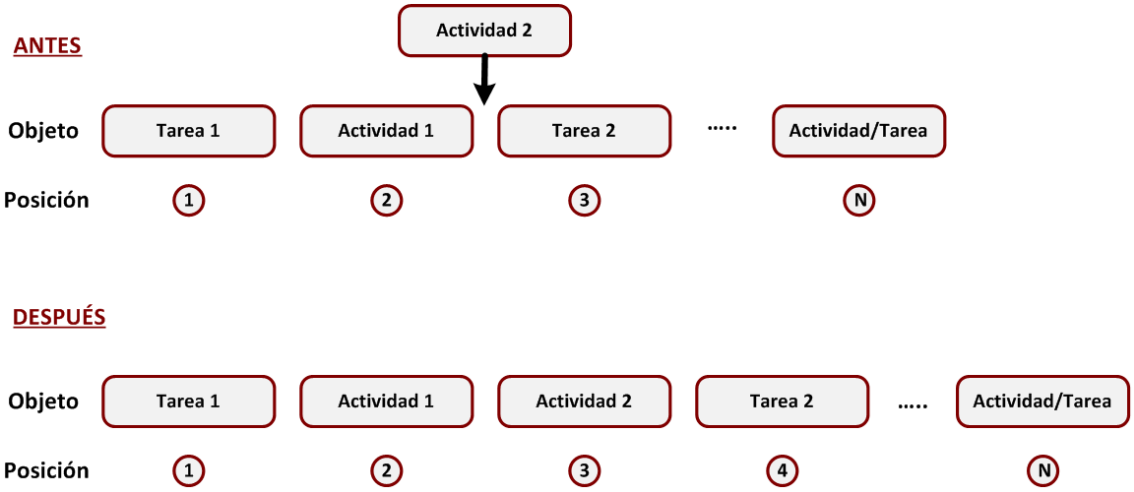


Figura 5.8: Ejemplo para añadir una subactividad a la actividad.

La nueva actividad estará vacía inicialmente (no tiene ningún elemento en la lista de objetos) y, como nombre inicial seguirá el modelo: "Nueva Actividad (X). Nombre_padre". Donde X será el número de actividades nuevas que se hayan creado y Nombre_Padre es el nombre la actividad a la que pertenece esta subactividad.

Eliminar Actividad (subactividad) o tarea

Al igual que se pueden introducir tareas y actividades a la actividad que se está editando, también se pueden eliminar. Otra de las posibilidades que se da para eliminar un objeto es que éste haya sido cortado y pegado en otra actividad. En ese caso, cuando se pega, previamente se elimina. Sea cual sea el caso, para borrar un elemento se quita de la lista y se desplaza al resto de elementos para rellenar el hueco creado.

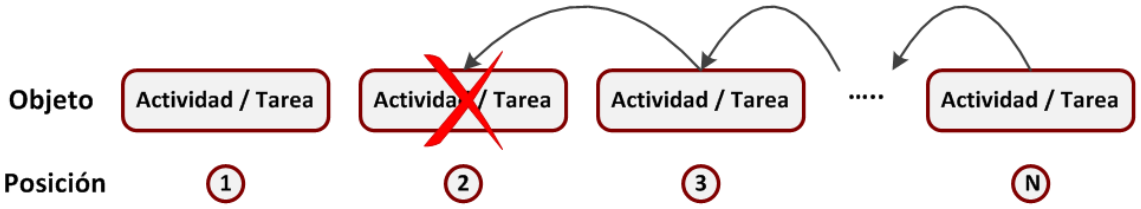


Figura 5.9: Eliminación de un elemento de la lista de objetos de la actividad.

En el caso de que la eliminación se trate de una actividad, los pasos a seguir se detallan en el bloque en el que se explica la funcionalidad de "Borrar toda la actividad". Si se trata de una tarea, no sólo hay que eliminarla de la lista de objetos de la actividad sino también de la lista general de tareas que se encuentra en la opción "Tareas".

Generar el código QR

Toda actividad se representa por un código QR que será colocado en el lugar en el que se realiza. Este código será utilizado por el teléfono móvil del paciente para indicarle al servidor de aQRdate la actividad que solicita ver paso a paso. El funcionamiento que hace posible la generación de estos códigos es el siguiente:

Figura 5.10: Esquema de funcionamiento del generador de códigos QR.

El identificador de las actividades tiene la siguiente sintaxis: "Routine:routineX@amilab", donde X es el número de actividades que había cuando ésta se creó. Se utiliza este texto y no el nombre completo que introduce el usuario y ve en la interfaz porque puede contener caracteres especiales como espacios o tildes los cuales generan problemas a la hora de trabajar con ellos.

Una vez obtenido este identificador, esta cadena de caracteres es codificado usando la librería Zxing[6]. El resultado es el código QR que el personal médico deberá de guardar e imprimir para que los usuarios puedan realizar la actividad.

[6]Librería para la generación de códigos QR Zxing: https://code.google.com/p/zxing/

Personalizar la actividad

Como se definió en el apartado 3.5, personalizar la actividad o adaptarla al usuario consiste en eliminarle ciertos pasos (subactividades y/o tareas). Para llevar a cabo esta opción, hay que almacenar todos los usuarios que el personal médico quiera inscribir. A cada uno de ellos se les asigna una copia de la lista de elementos genérica (la que figura en la Figura 5.7), y es en esta copia en donde se eliminarán aquellas subactividades o tareas que no se deseen que un usuario en concreto realice. Para entender mejor el funcionamiento de esta opción se adjunta el siguiente ejemplo:

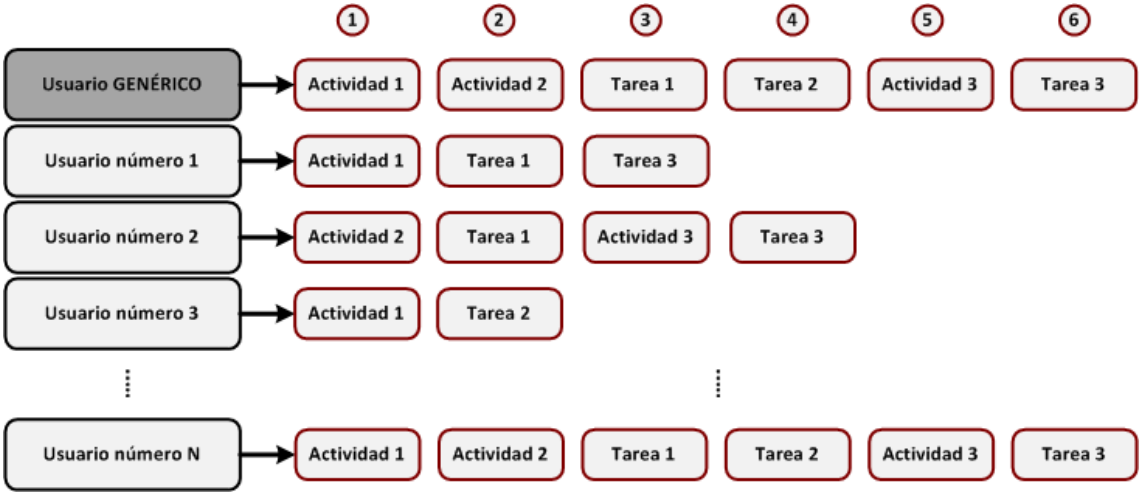


Figura 5.11: Ejemplo del funcionamiento de la personalización de la actividad.

En primer lugar, en este ejemplo se puede ver la secuencia de tareas y actividades para el usuario genérico, es decir, la lista de contenido para todos los pacientes que no hayan sido inscritos en la personalización de esta actividad.

Después se encuentran el resto de usuarios que hayan sido introducidos en el sistema. En los tres primeros se han eliminado algunos pasos, luego la secuencia que se verá en el teléfono móvil, para cada uno de ellos, será diferente.

Por último, en el caso del usuario N, tiene aún la secuencia intacta. En este caso se ha ingresado al usuario en el sistema pero todavía no se le ha eliminado ningún contenido. Si lo que se quiere es que este paciente realice la actividad completa, la forma más óptima sería borrarle de la lista de usuarios de esta actividad. Así, cuando el teléfono móvil pregunte por este paciente al servidor, al no figurar en la lista de usuarios, le devolverá la secuencia de contenidos del usuario genérico.

Vista Previa

Gracias a la vista previa puede verse la secuencia de tareas tal y como las vería el paciente en el teléfono móvil. Para mostrarlo se recorre toda la lista de tareas y actividades (de una actividad) para el usuario especificado (en caso de no encontrarse el usuario, se muestra la lista del usuario genérico). Cuando el elemento encontrado es una tarea, ésta se muestra al usuario. En caso de que el elemento sea una actividad, se repite de nuevo el proceso; se recorre la lista de contenidos en busca las tareas. Si se vuelven a encontrar nuevas subactividades se repetirá el proceso descrito hasta que se encuentren las tareas que componen todas las subactividades. Esta forma de proceder para obtener la vista previa de una actividad queda plasmada en el siguiente ejemplo:

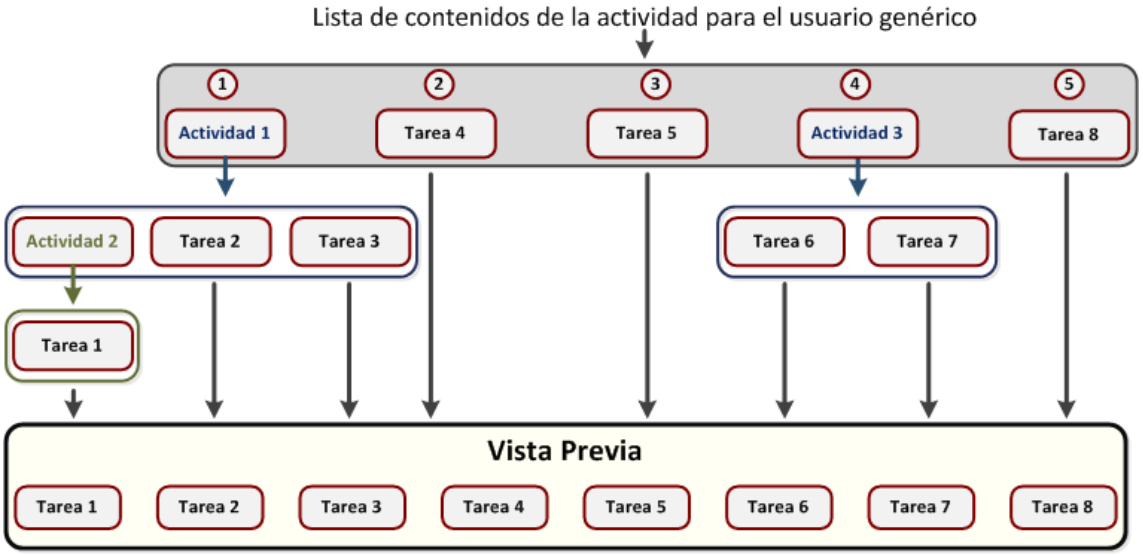


Figura 5.12: Esquema de funcionamiento de la vista previa.

Borrar toda la actividad

Una de las opciones que tiene este programa es borrar una actividad. Para llevar a cabo esta opción, hay que diferenciar dos posibles escenarios:

- Esta actividad puede formar parte de otras (puede estar copiada) y por tanto no se puede eliminar por completo. Lo que se hace en este caso es borrar solamente la actividad del lugar que el usuario desea.
- Si por el contrario esta actividad no se encuentra copiada en otras actividades: primero se borran todas las tareas (se borran de la lista de tareas almacenadas en la opción “Tareas”) y finalmente se borra la actividad de la lista que contiene todas las actividades del sistema (del listado de la opción “Actividades”).

5.2.2.2. Tarea

El módulo de tareas tiene la siguiente arquitectura:

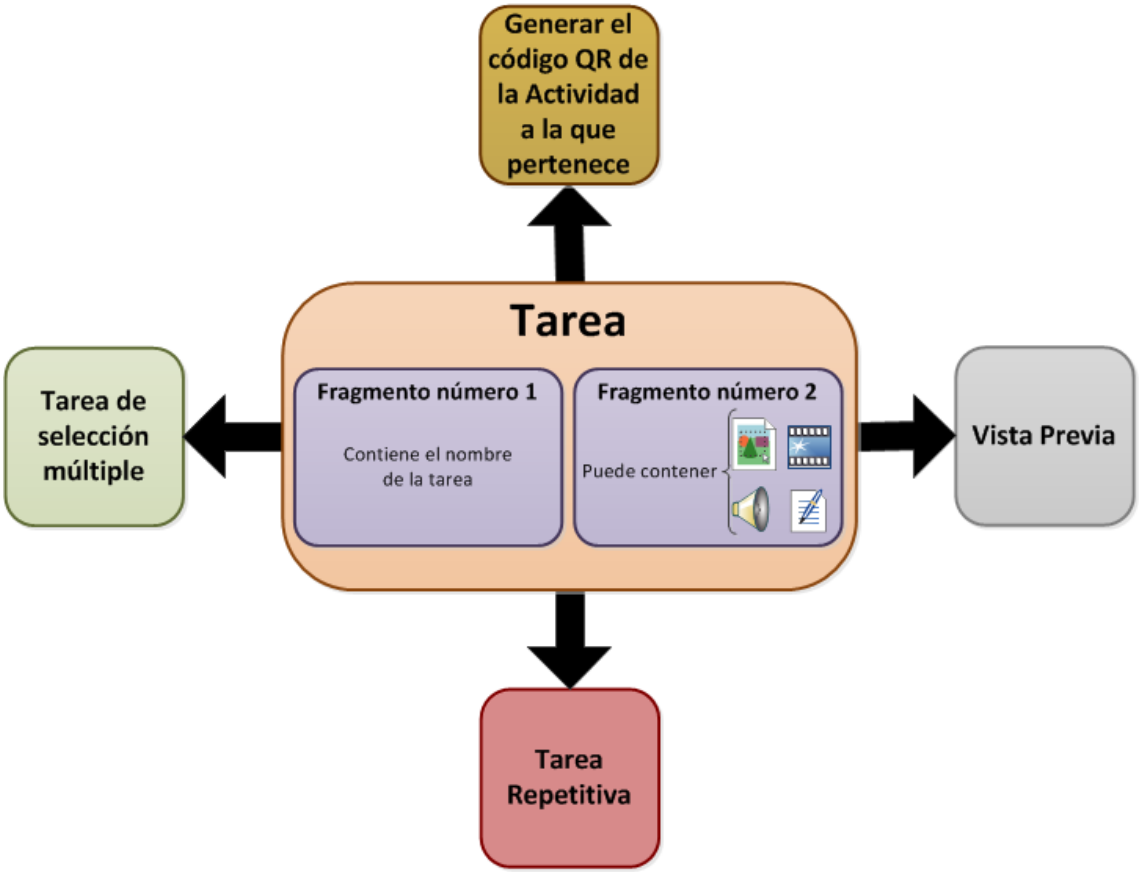


Figura 5.13: Arquitectura del módulo encargado de la gestión de cada tarea.

Como se ha comentado anteriormente, cada tarea está formada por dos fragmentos; el primero contiene el nombre de la tarea y el segundo el contenido que se quiera mostrar al usuario (imagen, vídeo, audio o descripción textual).

Principalmente, existen cuatro funciones que pueden ser realizadas sobre cada tarea:

Generar el código QR de la actividad a la que pertenece

Una vez obtenido el identificador de la actividad a la que pertenece la tarea, el código QR es generado de la misma forma que se describe en la Figura 5.10.

Vista Previa

Para la edición de las tareas, se estipuló en el requisito funcional número 7, que este trabajó debía realizarse teniendo presente, en todo momento, una vista previa del aspecto de la tarea (edición tipo WYSIWYG). Por tanto, este bloque se encarga de

mostrar el contenido de la tarea, es decir, el contenido de los dos fragmentos, cada vez que se guardan los cambios durante la edición de la misma.

Tarea Repetitiva

Una de las opciones que puede tener una tarea es que se repita un número de veces definido o por el personal médico o por el usuario durante la realización de la actividad. Este bloque se encarga por tanto de modificar la cantidad de veces que se repite la tarea en función de los datos introducidos en el sistema.

Tarea de selección múltiple

Otra de las posibilidades que puede albergar la tarea es la posibilidad de mostrar al usuario una lista de opciones y que él elija una de ellas. Esta elección modifica el cambio a seguir en el trascurso de la actividad.

Para conseguir la selección múltiple, en la etapa de diseño, se definió que por cada opción que el usuario introdujese, se crease un nuevo elemento en la actividad. En el siguiente ejemplo se añaden dos nuevas opciones a una tarea ("Tarea 1"). El resultado final es que, las opciones se han introducido en la lista de tareas y actividades a continuación de la tarea que las contiene.

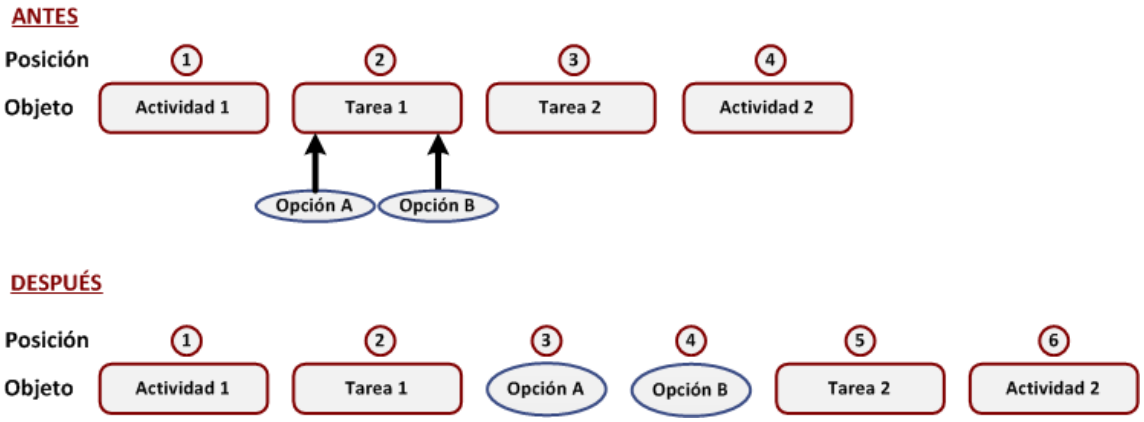


Figura 5.14: Esquema de funcionamiento de las tareas de selección múltiple.

Una vez alcanzada la "Tarea 1" durante el visionado de la actividad en el dispositivo móvil, en función de la opción que escoja el usuario (A o B) se le conducirá a la posición número 3 o la posición número 4.

Además dentro de cada opción, se debe poder introducir aquellas subactividades o tareas que se quisieran mostrar sólo si se elige esa opción. Por similitud con el contenido que tiene una actividad, aunque el aspecto con el que se representa una opción no lo parezca, se decidió que fueran de tipo actividad.

5.2.3. Conexión con el servidor

Este bloque se utiliza al inicializar la herramienta de autor y también en el caso de guardar los cambios que se realizan en el contenido de las actividades.

Cuando el programa se inicia, éste se conecta al servidor y se descarga el archivo de entidades donde se encuentran todas las actividades, tareas y fragmentos siguiendo la sintaxis descrita en la sección 3.4 de esta memoria (Figura 5.15).

Figura 5.15: Esquema de la carga y el almacenamiento de los datos en el sistema.

A continuación la información de este archivo es tratado por el bloque encargado de la traducción del código de entidades a los elementos gráficos (sección 5.2.5) que se muestran en la herramienta de autor. El resto de datos que son cargados desde el servidor corresponden con el contenido de las tareas: imágenes, vídeos y audios (las descripciones textuales así como el título de las tareas y actividades se encuentran en este mismo archivo de información).

Una vez que el usuario desea guardar los cambios o bien la aplicación se cierra, se guardan los datos en el sistema. El proceso es similar al que se acaba de describir pero en sentido contrario. Se reescribe el fichero de entidades con todos los datos de las actividades, tareas y fragmentos, y se guardan también todos los archivos que necesitan las tareas en el servidor.

5.2.4. Conversión de las actividades diseñadas al archivo de entidades

Como se ha comentado en el apartado anterior, una vez que se tienen implementadas las actividades mediante la herramienta de autor, antes de almacenar esta información en el servidor hay que traducirla al lenguaje descrito en la sección 3.4. El proceso que se describe en este apartado es inverso al que se realiza cuando se obtiene del servidor el archivo de entidades al iniciar la aplicación, motivo por el cual no se explica de nuevo esta conversión.

Para explicar el funcionamiento de este módulo se parte una actividad ya diseñada usando el programa (Figura 5.16).

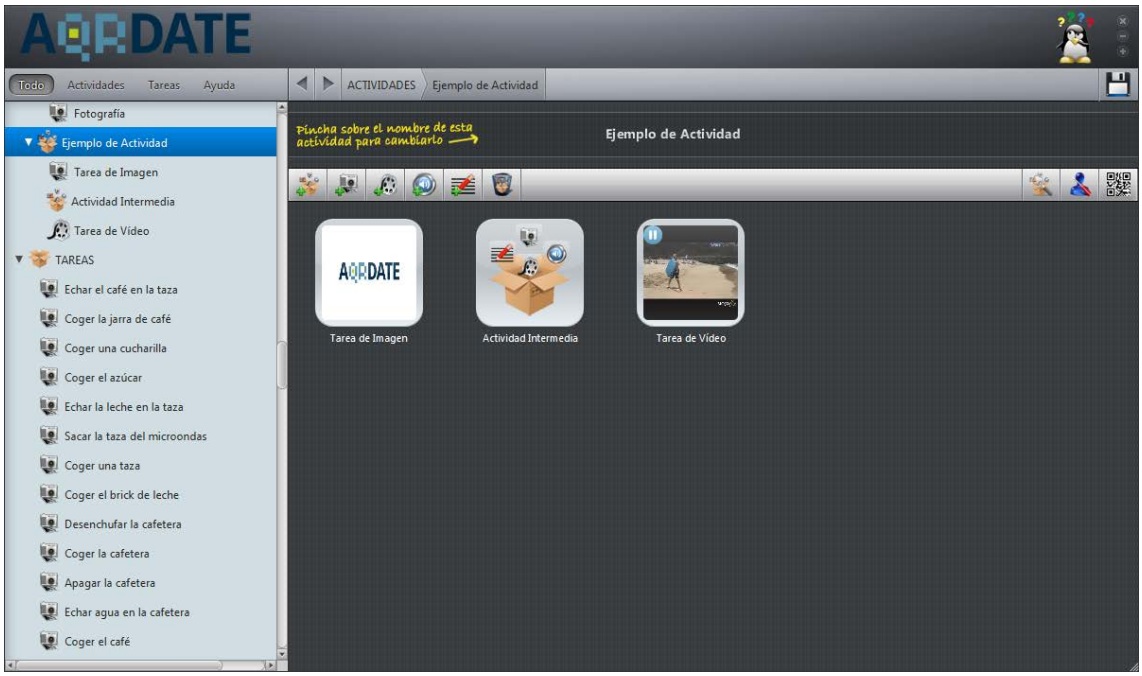


Figura 5.16: Ejemplo de actividad para la conversión al archivo de entidades.

La actividad creada tiene por nombre, otorgado por el usuario, “Ejemplo de Actividad” y contiene dos tareas (“Tarea de Imagen” y “Tarea de Vídeo”) y una subactividad (“Actividad Intermedia”) como se puede ver en la Figura 5.16.

El resultado de la conversión de objeto gráfico a código de ontologías se muestra en la Figura 5.17. En él se puede observar como el identificador que el sistema ha definido a esta actividad es “routine19” y el nombre que el usuario ha introducido se encuentra especificado en la propiedad “descriptedName”.

A continuación, en el lenguaje creado por este bloque de conversión se observan las relaciones. La primera de ellas, “composesRoutine” determina el padre de la actividad. En este caso esta actividad no pertenece a otra (no es una subactividad), por ello su padre es “RoutineNull” (no tiene padre). Seguidamente se definen mediante la relación “hasAction” el contenido de la actividad: una tarea con nombre “task115”, una actividad llamada “routine20” y otra tarea que el sistema define con el nombre de “task116”.

Por último hay que definir las relaciones de anterior/siguiente. Estas relaciones se usan cuando la actividad en cuestión pertenece a otra (tiene un padre), para saber el orden en el que se sitúa dentro de ella. En este caso se definen mediante “TaskNull” puesto que como se ha comentado anteriormente ésta carece de padre.

```
entity Routine:routine19@amilab{
   properties{
      descriptedName = "Ejemplo de Actividad";
   }
   relations{
      composesRoutine = Routine:RoutineNull@amilab;
      hasAction = Task:task115@amilab;
      hasAction = Routine:routine20@amilab;
      hasAction = Task:task116@amilab;
      prevAction = Task:TaskNull@amilab;
      nextAction = Task:TaskNull@amilab;
   }
}
```

Figura 5.17: Código generado de la actividad mostrada en la Figura 5.16.

Una vez generado el código de la actividad, este bloque continúa generando el código necesario para la primera tarea mostrado en la Figura 5.16. Al igual que para el caso de la actividad, el identificador de la tarea lo define el sistema; “task115”.

La primera de las relaciones, hace referencia a la actividad a la que pertenece esta tarea, que es la actividad que se acaba de definir (“routine19”).

Como se explicó en éste y en el capítulo 3, cada tarea está formada por dos fragmentos. Para especificar este dato se usa la relación “hasFragment”. En concreto esta tarea está formada por los “fragment230” y “fragment 231”.

Para esta tarea, al ser la primera dentro de la actividad a la que pertenece (“routine19”), el paso previo (“prevAction”) es nulo, es decir, no existe y el paso siguiente (“nextAction”) corresponde con una actividad llamada “routine20”.

```
entity Task:task115@amilab{
   relations{
      composesRoutine = Routine:routine19@amilab;
      hasFragment = Fragment:fragment230@amilab;
      hasFragment = Fragment:fragment231@amilab;
      prevAction = Task:TaskNull@amilab;
      nextAction = Routine:routine20@amilab;
   }
}
```

Figura 5.18: Código generado de la tarea llamada “task115”.

Por último se incluye el código de los fragmentos que conforman la tarea que se acaba de definir.

```
entity Fragment:fragment230@amilab{
   properties{
      fragmentType = "text";
      fragmentContent="<H1>Tarea de Imagen</H1>";
   }
   relations{
      composesTask = Task:task115@amilab;
      prevFragment = Fragment:FragmentNull@amilab;
      nextFragment = Fragment:fragment231@amilab;
   }
}
```

Figura 5.19: Código del “fragment230”.

El primer fragmento llamado “fragment230” (Figura 5.19) corresponde con el título de la tarea, por ello tiene la propiedad (“fragmentType”) de tipo texto. El título introducido por el usuario se recoge en la propiedad “fragmentContent”.

La primera relación del fragmento, “composesTask”, sirve para conocer la tarea a la que éste pertenece. Las otras dos relaciones sirven para conocer el orden de los fragmentos dentro de la tarea. En este caso, para el “fragment230”, no hay un fragmento anterior y el siguiente corresponde con el fragmento “fragment231”.

El último de los dos fragmentos se define de manera similar al explicado. La única diferencia es que el segundo fragmento siempre contiene el tipo de datos que el usuario ha definido durante su diseño. En este caso es de tipo imagen pero podría haberse optado por un vídeo, un audio o una descripción textual. El contenido en este caso es el nombre que tiene la imagen el servidor.

```
entity Fragment:fragment231@amilab{
   properties{
      fragmentType = "image";
      fragmentContent="Logo_aQRdate.png";
   }
   relations{
      composesTask = Task:task115@amilab;
      prevFragment = Fragment:fragment230@amilab;
      nextFragment = Fragment:FragmentNull@amilab;
   }
}
```

Figura 5.20: Código del “fragment231”.

Una vez analizado toda la estructura que presenta la herramienta de autor de este trabajo, se puede concluir que cumple con todos los requisitos (funcionales y no funcionales) que fueron establecidos.

5.2.5. Objetos gráficos

Para crear la interfaz gráfica se han usado varias clases de JavaFX que implementan diferentes objetos gráficos. Además a cada uno de ellos se le añade los "*Listener*" apropiados para que cuando el usuario interactúe con ellos, el programa reconozca la acción que se ha realizado y actúe en consecuencia.

Los objetos gráficos que se han usado son:

Stage

Para JavaFX la *stage* es el lugar donde se muestra todo el contenido gráfico (es la ventana que se mostrará al usuario). Para más información consultar el Anexo C.

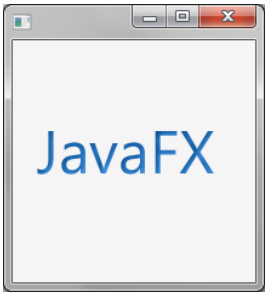


Figura 5.21: Ejemplo de una *stage* de JavaFX.

Tool Bar

Esta clase implementa una barra de herramientas, formada normalmente por botones o controles que incluyen iconos y que aparecen organizados.

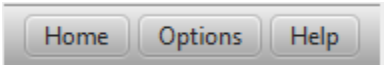


Figura 5.22: Ejemplo de Tool Bar.

Tree View

Esta clase se utiliza para mostrar la información organizada en forma de árbol. En ella se puede ver el típico árbol de datos en el que se puede abrir cada uno de los nodos para ver qué tiene dentro, cerrarlos, etc. Similar al árbol de directorios que se muestran en algunas aplicaciones para elegir un fichero.

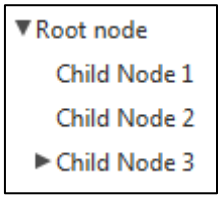


Figura 5.23: Ejemplo de Tree View.

Button

Un botón en un elemento que tiene dos estados: seleccionado y no seleccionado.

Figura 5.24: Ejemplo de Button.

Label

Se utiliza para mostrar en la interfaz un texto o un mensaje estático para proporcionar al usuario algún tipo de información.

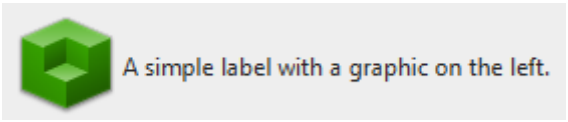


Figura 5.25: Ejemplo de Label.

Check Box

Se utiliza para implementar un cuadro de selección. Básicamente es un botón que tiene dos estados: seleccionado y no seleccionado.

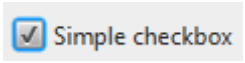


Figura 5.26: Ejemplo de Check Box.

Radio Button

Es muy parecido al Check Box con la salvedad de que en esta clase el estado de un botón depende del resto ya que sólo uno de ellos puede estar seleccionado al mismo tiempo.

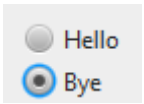


Figura 5.27: Ejemplo de Radio Button.

Hyperlink

Un hiperenlace consiste en una o más palabras que al dar clic sobre ella con el cursor permite navegar a un documento diferente que ampliará la información de las palabras del hiperenlace.

Figura 5.28: Ejemplo de Hyperlink.

Combo Box

Este objeto gráfico se utiliza para mostrar una lista de elementos (*ítems*) y que el usuario escoja uno de ellos.

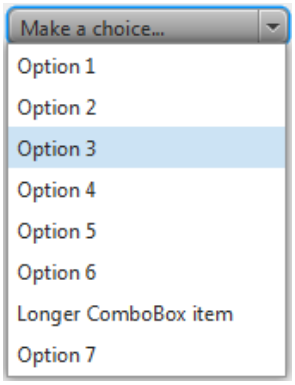


Figura 5.29: Ejemplo de Combo Box.

Text Field

Es una clase de objeto gráfico en el cual el usuario puede introducir una cadena de caracteres por el teclado.

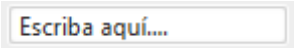


Figura 5.30: Ejemplo de Text Field.

Table View

Elemento utilizado para crear tablas con sus respectivas filas y columnas en donde se ingresan los datos que se desean.

First	Last	Email
Jacob	Smith	jacob.smith@example.com
Isabella	Johnson	isabella.johnson@example.com
Ethan	Williams	ethan.williams@example.com
Emma	Jones	emma.jones@example.com
Michael	Brown	michael.brown@example.com

Figura 5.31: Ejemplo de Table View.

Tooltip

Es una herramienta de ayuda visual, que funciona al situar el cursor sobre algún elemento gráfico, mostrando una ayuda adicional para informar al usuario de la finalidad del elemento sobre el que se encuentra.

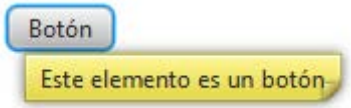


Figura 5.32: Ejemplo de Tooltip.

Image View

Es la clase que se dedica a cargar las imágenes (formato .png y .jpg).

Figura 5.33: Ejemplo de Image View.

Media Player

Es el reproductor de audio (formato .mp3) y vídeo (formato .mp4) que ofrece JavaFX.

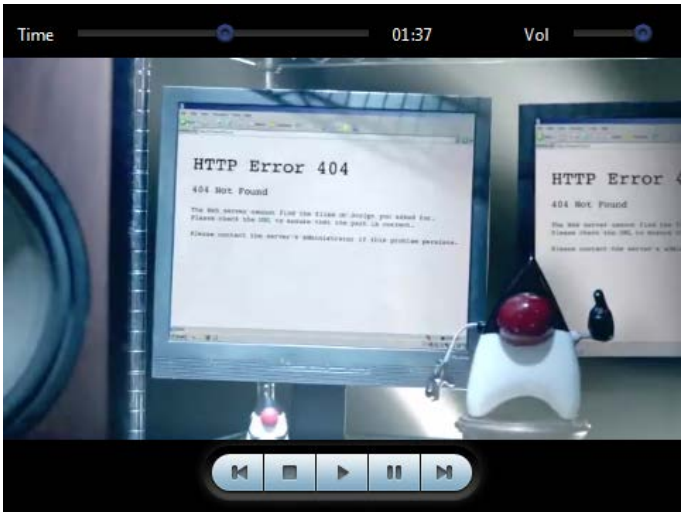


Figura 5.34: Ejemplo de Media Player.

Filechooser

Filechooser es una clase que permite mostrar fácilmente una ventana para guardar y/o seleccionar un fichero.

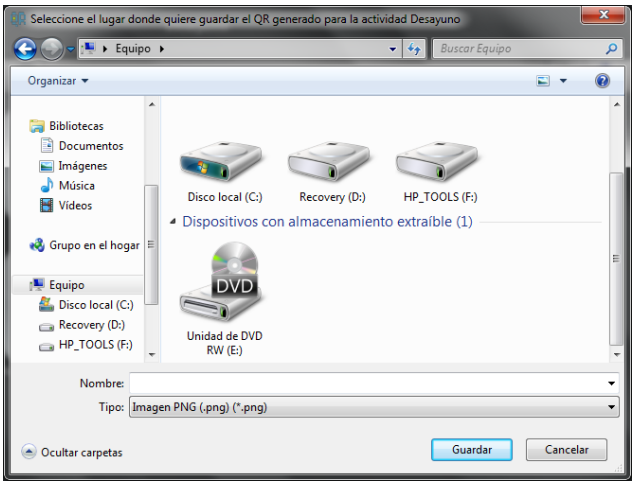


Figura 5.35: Ejemplo de Filechooser.

Spinner

Es un contador incremental/decremental y sirve para seleccionar un valor numérico en un rango fijado.

Figura 5.36: Ejemplo de Spinner.

BreadCrumb Bar

Es una línea de texto, explicada en la sección 5.2.1, en la que se indica el recorrido realizado y la forma de regresar. De esta forma, se da a conocer al usuario el punto en el que encuentra dentro de la ruta así como la posibilidad de volver a cualquier punto de ella pinchando sobre un elemento del recorrido mostrado.

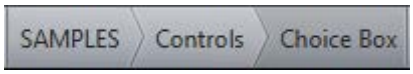


Figura 5.37: Ejemplo de BreadCrumb Bar.

Context Menu

Es un menú emergente que se muestra cuando se aprieta el botón secundario del ratón. En él se muestran las opciones propias que tiene el objeto que se ha seleccionado.

Figura 5.38: Ejemplo de Context Menu.

Option Pane

Son ventanas modales ya predefinidas en las cuales se muestra un mensaje de información y el usuario escoge una de las opciones que se le plantea (diálogos simples).

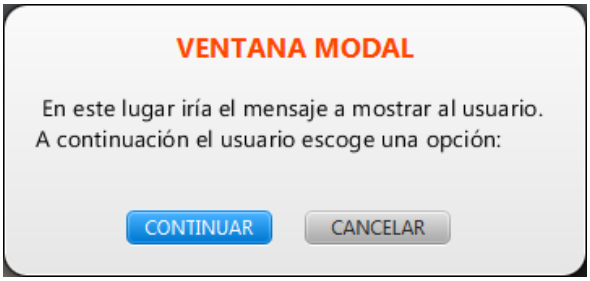


Figura 5.39: Ejemplo de Option Pane.

Para cambiar el aspecto, la posición y el estilo a estos objetos gráficos que se acaban de explicar, se ha utilizado una hoja de estilo en cascada (CSS). Un ejemplo de código utilizado en este trabajo ha sido:

Figura 5.40: Ejemplo de la hoja de estilo en cascada (CSS).

En este ejemplo se puede apreciar que la ventana mostrada tiene un fondo de color negro y dos botones: el de la izquierda con el aspecto que por defecto trae un botón en JavaFX, con el texto "AQRDATE", y en la derecha otro similar al primero pero al que se le ha modificado su estética siguiendo el siguiente código:

```
#ventana {
   -fx-background-image: url("texture.png"); //Imagen de la textura usada
   -fx-padding: 8px; //Relleno
}

.boton_derecho {
   -fx-background-image: url("icono.png"); //Icono usado en el botón derecho
   -fx-background-repeat: no-repeat; //No se repite la imagen de fondo
   -fx-background-color: "#c9e3f6"; //Color usado de fondo en el botón
   -fx-background-radius: 22; //Curvatura de los bordes del botón
   -fx-skin: "com.sun.javafx.scene.control.skin.ButtonSkin"; //Estilo escogido
   -fx-padding: 4px 4px 3px 4px; //Relleno
   -fx-alignment: center; //Alineación del contenido del botón
   -fx-text-fill: white; //Color de las letras del botón (en este caso no hay texto)
```

```
    -fx-font-size: 11px; //Tamaño del texto del botón
    -fx-font-weight: bold; //Formato "Negrita" al texto del botón
}

. boton_derecho:hover {//Cuando se pasa el ratón por encima del botón derecho
    -fx-border-color: derive(-fx-accent,80%), -fx-accent, derive(-fx-accent,80%); //Borde
    -fx-border-insets: -1,0,1; // Hay tres bordes en esas posiciones con respecto al botón
    -fx-border-radius: 7,6,5;  //Curvaturas de los tres bordes mostrados
    -fx-effect: dropshadow( three-pass-box , white , 8 , 0.2 , 0 , 0 ); //Efecto sombra
}

. boton_derecho:focused{//Cuando se selecciona el botón aparece otro efecto sombra
    -fx-effect: dropshadow( three-pass-box , derive(-fx-focus-color, 20%), 12, 0.6 ,0,0 );
}
```

Figura 5.41: Código de la hoja de estilo en cascada (CSS) de la Figura 5.40.

5.3. DESCRIPCIÓN DE LA INTERACCIÓN DEL USUARIO

En este apartado se describe el funcionamiento de la herramienta de autor desde el punto de vista del usuario. Para ello se recorrerán las diferentes pantallas con varios ejemplos:

5.3.1. Primer Ejemplo: Crear una nueva actividad

Para este ejemplo se va a crear la actividad "Hacer el desayuno". Los pasos a tener en cuenta son los siguientes:

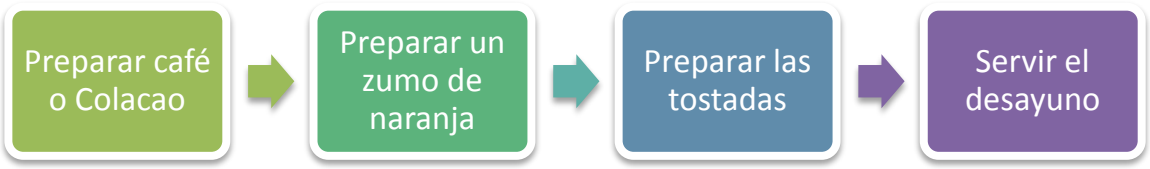


Figura 5.42: Actividad a diseñar por el personal médico usando la herramienta de autor.

Una vez establecidos los pasos que tiene la actividad, se deben dividir en las actividades y tareas necesarias, además de obtener todos los contenidos que se vayan a mostrar en las tareas (imágenes, vídeos, audios y descripciones textuales).

1. Primer Paso: “Preparar café” o “Colacao”. Es una decisión que debe tomar el paciente, luego se modelará como una tarea en la que se dé a elegir entre dos opciones: “café” o “Colacao”. En cada una de ellas se introducirá la información que se quiera realizar:
    - Para la opción del café. Se creará una actividad en la que se explique cómo hacer café con la cafetera, una tarea en la se pida apagarla, otra para desenchufarla y finalmente una actividad en la que se enseñe a servirse una taza.
    - Para la segunda opción: “Colacao”. Se diseñará una actividad que consista en calentar la leche y una tarea, acto seguido, que muestre cómo se debe de añadir el cacao.
2. Segundo Paso: “Preparar un zumo de naranja”. Actividad en la que, tarea a tarea, se exponga los pasos para obtener un zumo de naranja natural.
3. Tercer Paso: ”Preparar tostadas”. Al igual que la actividad anterior, se explica las tareas necesarias para obtener unas tostadas con mantequilla y mermelada.
4. Cuarto Paso: “Servir el desayuno”. Tarea con texto descriptivo en la que se informa al paciente la manera de servir el desayuno que acaba de hacer.

A continuación, se ejecuta la aplicación:

Figura 5.43: Inicio de la aplicación.

Ahora se añade una nueva actividad arrastrándola hasta el lugar deseado:

Figura 5.44: Se crea la nueva actividad "Hacer el desayuno".

Dentro de ella, se cambia el nombre a la actividad y se crea la primera tarea de tipo imagen, que es la encargada de mostrar al usuario las dos opciones de bebida que puede elegir. Aquí se puede ver una diferencia con respecto al diseño de la interfaz; por facilidad, se pueden añadir cuatro tipos de tareas a la actividad, tantas como tipos de datos existen (imagen, vídeo, audio y descripción de la tarea mediante un texto).

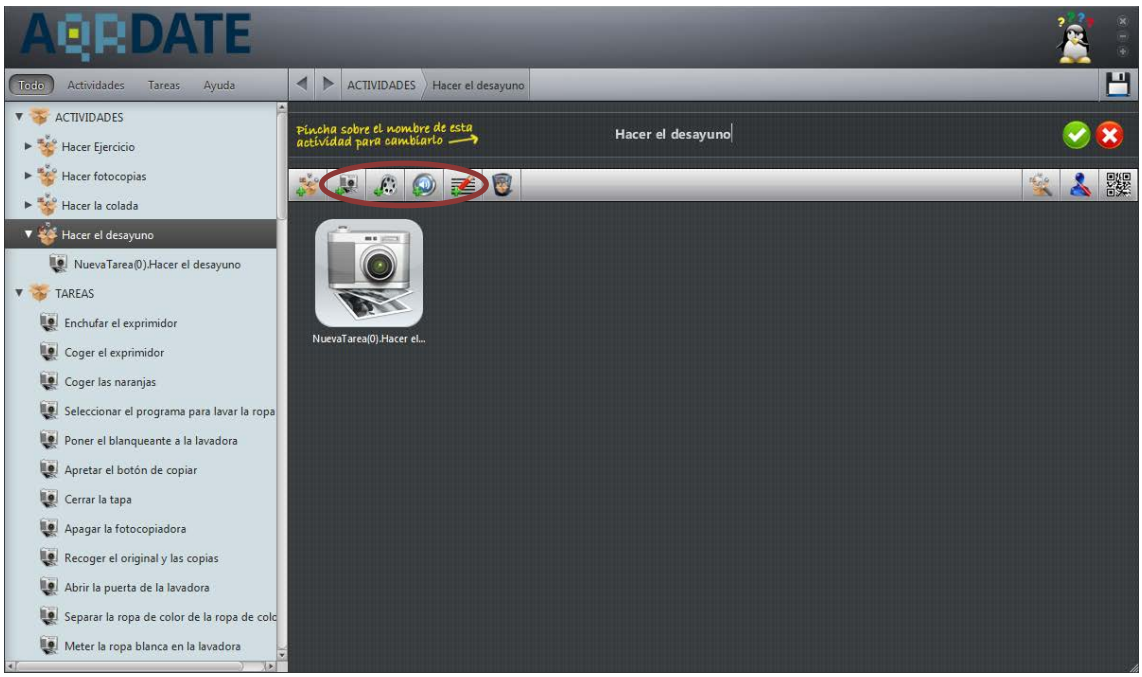


Figura 5.45: Se muestra cómo añadir una tarea y cómo cambiar el nombre a la actividad.

Ahora se edita la tarea. Se pone el título “¿Qué bebida quieres tomar?”, se introduce la imagen representativa y se crean las dos opciones: “café” y “Colacao”. Finalmente se guardan los cambios.

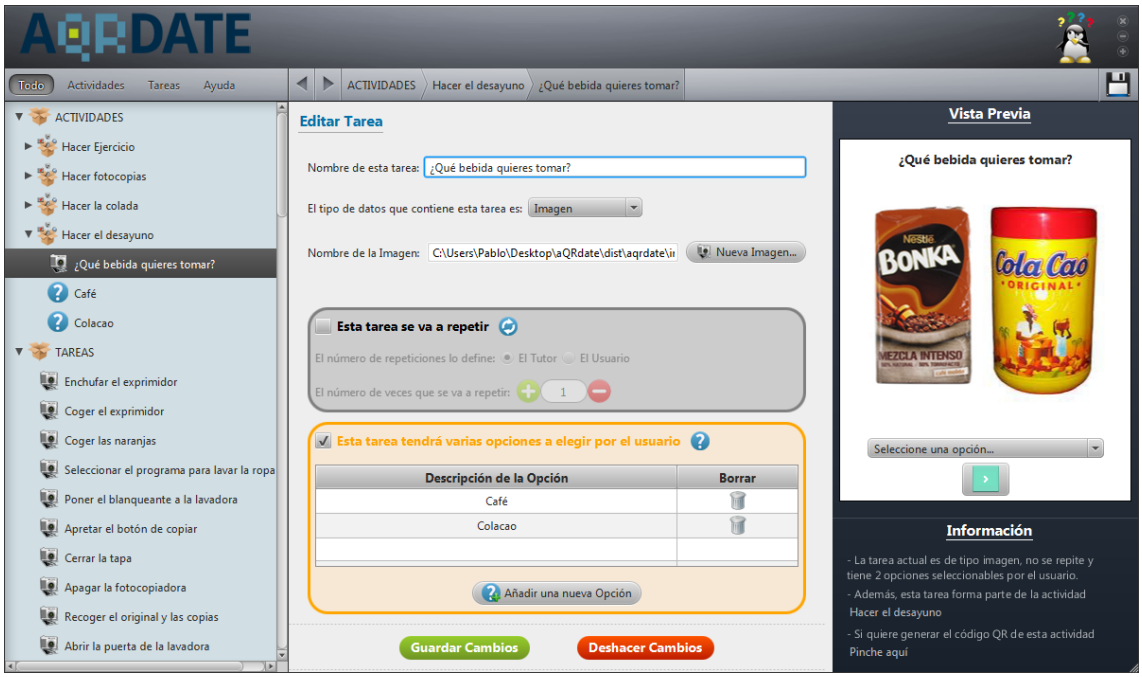


Figura 5.46: Se crean dos opciones en una tarea de selección múltiple.

Como resultado del paso anterior, en la actividad a la que pertenece aparecen dos iconos que representan a las opciones creadas y en la tarea aparece el símbolo de tarea de selección múltiple.

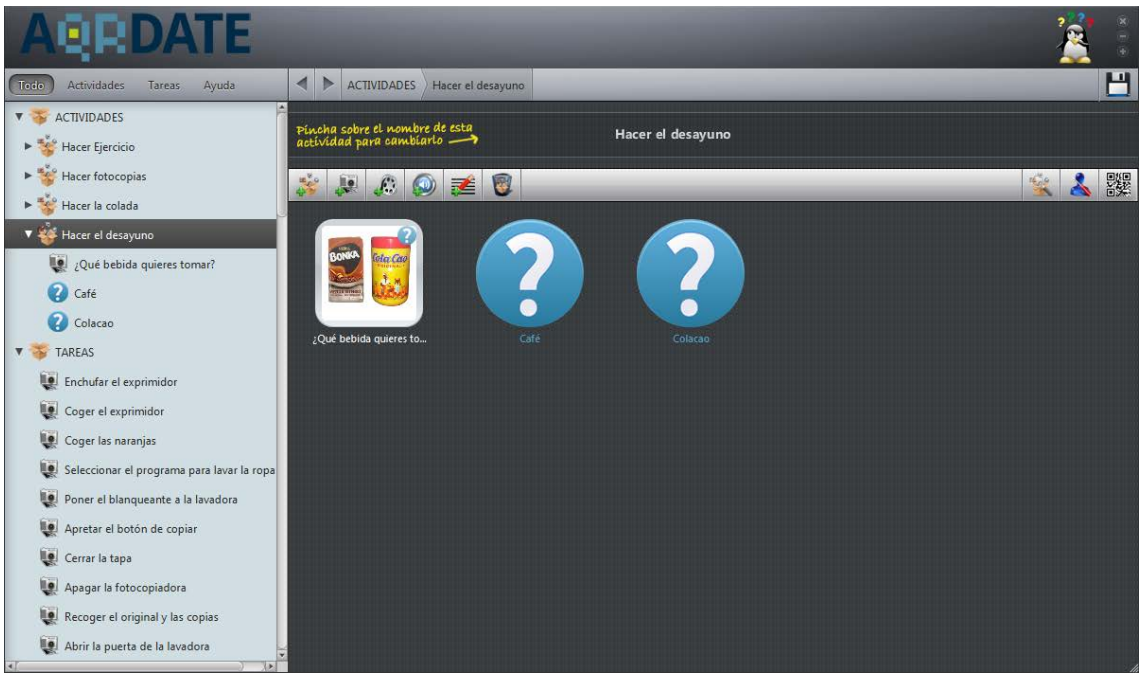


Figura 5.47: Actividad con dos opciones y una tarea de selección múltiple.

Se empieza completando la información para la primera opción; “café”. Aquí se crea una actividad llamada “Preparar café”. En ella se introduce cada tarea con contenidos multimedia tipo imagen. La edición de la primera de ellas se puede ver a continuación:

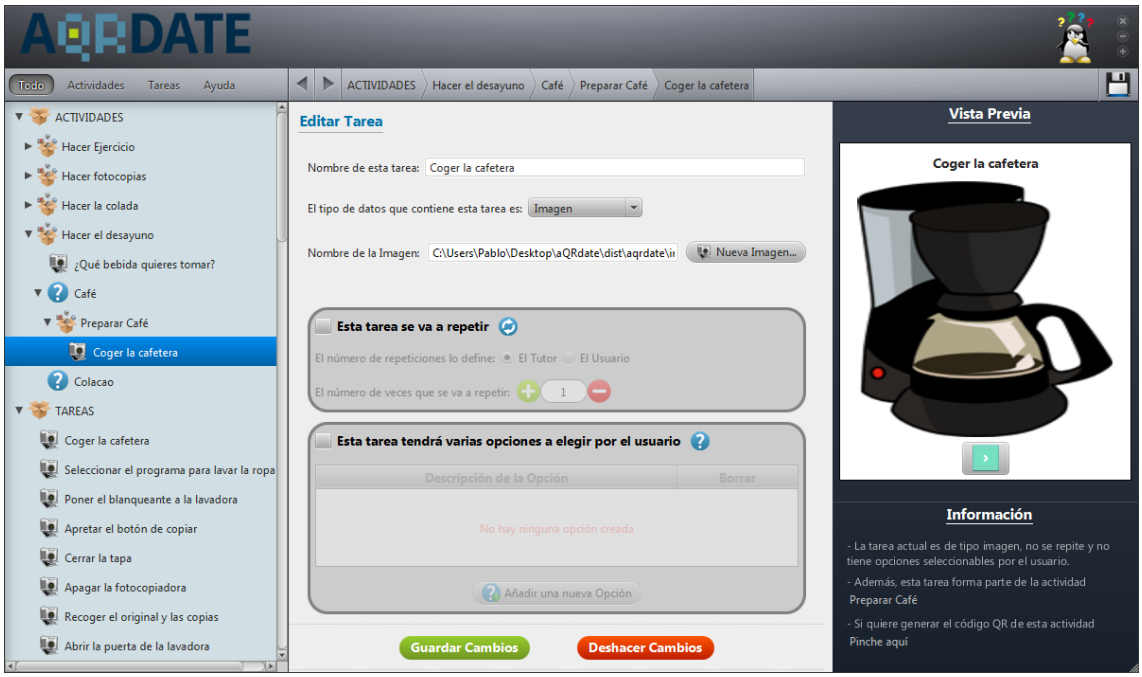


Figura 5.48: Tarea correspondiente a la actividad “Preparar café”.

El resultado de introducir todas las tareas necesarias para enseñar a preparar un café es:

Figura 5.49: Actividad “Preparar café” completada.

El siguiente paso es añadir dentro de la opción “café” dos tareas. Una para que apague la cafetera y otra para que la desconecte (ambas de nuevo de tipo imagen). Seguidamente se crea una nueva actividad que especifique los pasos para servir una taza de café.

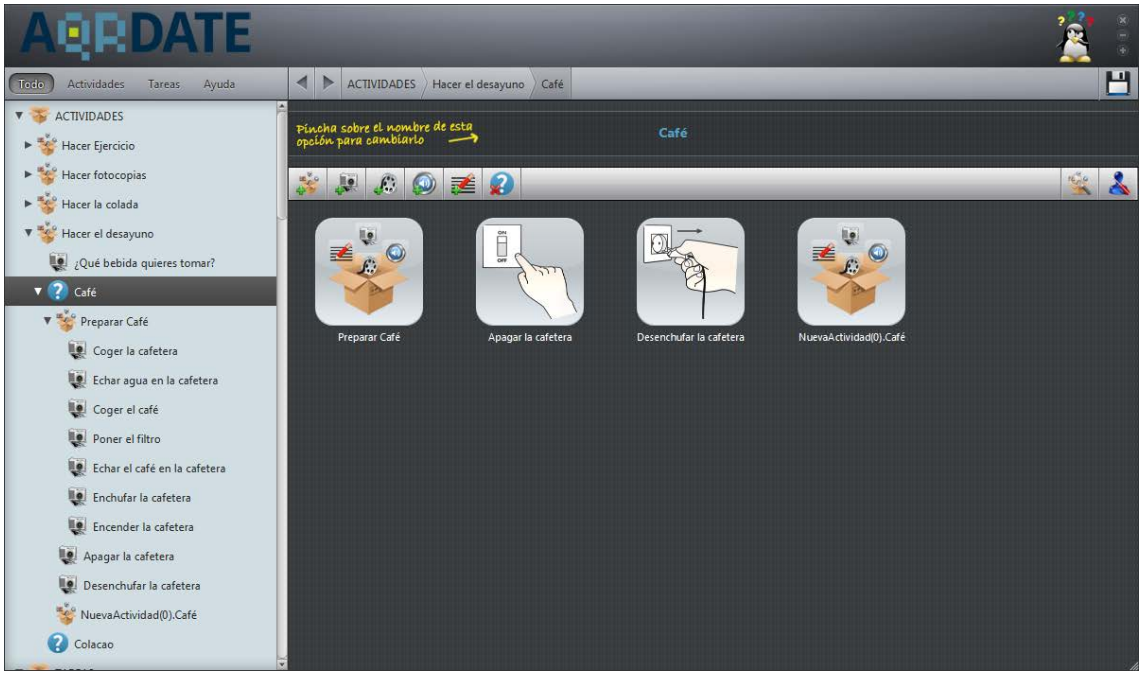


Figura 5.50: Opción “café” con dos nuevas tareas y una nueva actividad.

Se van introduciendo las tareas de la actividad “Servir una taza de café”.

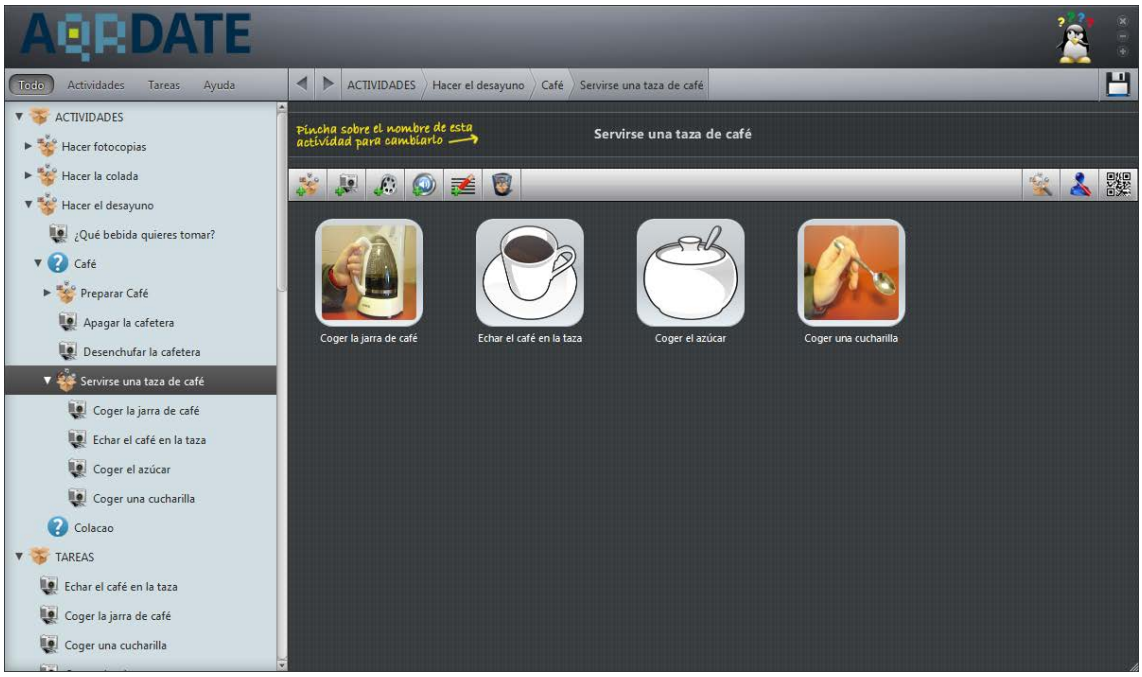


Figura 5.51: Primeras tareas de la actividad “Servir una taza de café”.

La última tarea de esta actividad (“Servir una taza de café”) es “echar azúcar en la taza”. Como a cada usuario le puede gustar el café más o menos dulce, esta tarea se diseña para que sea de tipo repetitiva en la que el usuario elija el número de cucharadas de azúcar que desea.

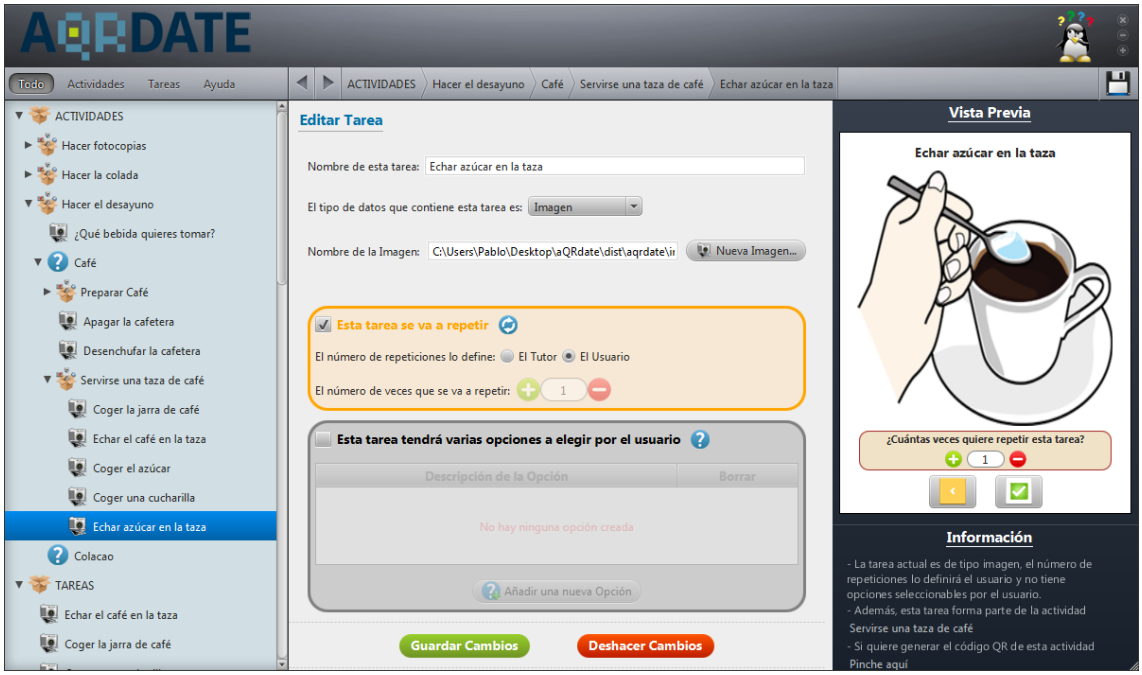


Figura 5.52: Tarea “Echar azúcar en la taza” que es de tipo repetitivo.

La opción “café” ha quedado concluida:

Figura 5.53: Opción “café” completada.

Se diseña ahora la opción “Colacao”. Lo primero que se tiene que hacer es la actividad “Calentar la leche”. Se introducen las tareas necesarias:

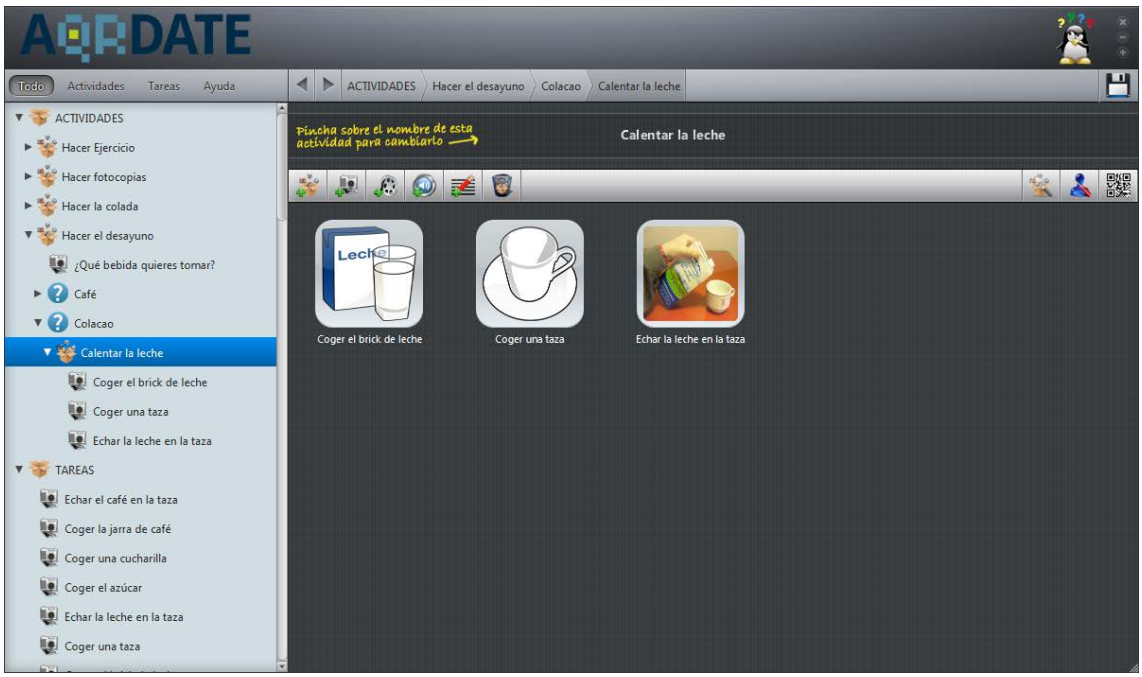


Figura 5.54: Primeras tareas de la actividad “Calentar la leche”.

La siguiente tarea será de tipo vídeo en la cual muestre una taza de leche metiéndose en el microondas y fijando el tiempo necesario para que se caliente:

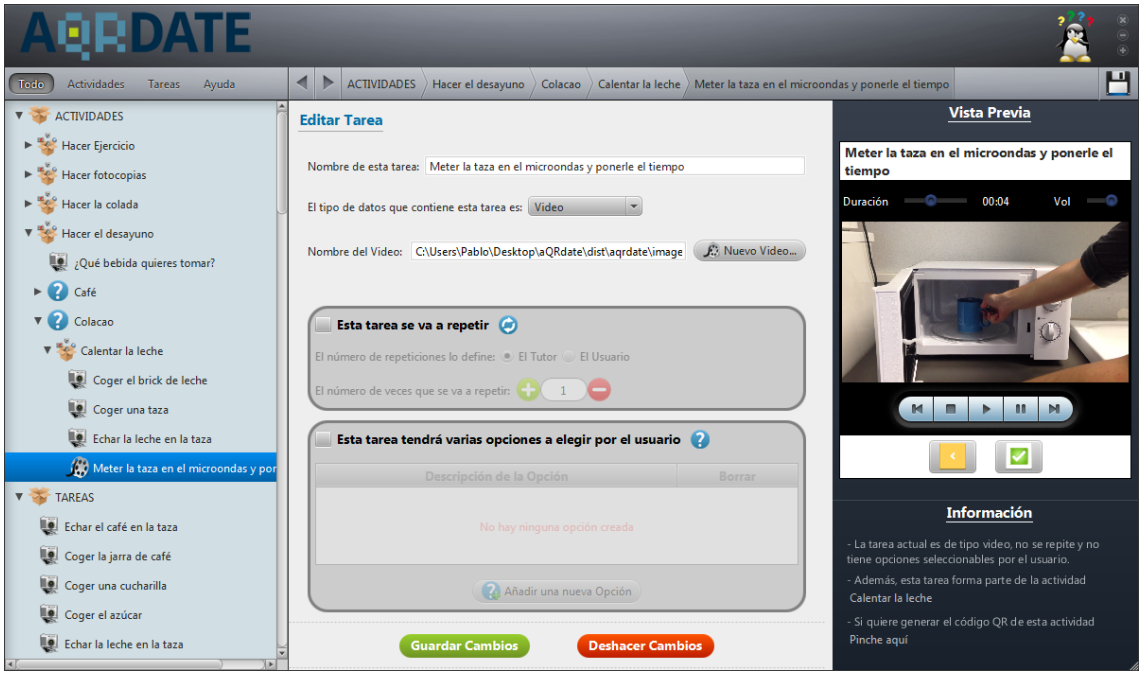


Figura 5.55: Tarea de tipo vídeo llamada “Meter la taza en el microondas y ponerle el tiempo”.

A continuación se incluye otra tarea en la que se avisa al usuario de que tiene que esperar hasta que el microondas terminé así como del sonido que éste hace cuando acaba (luego la tarea será de tipo audio).

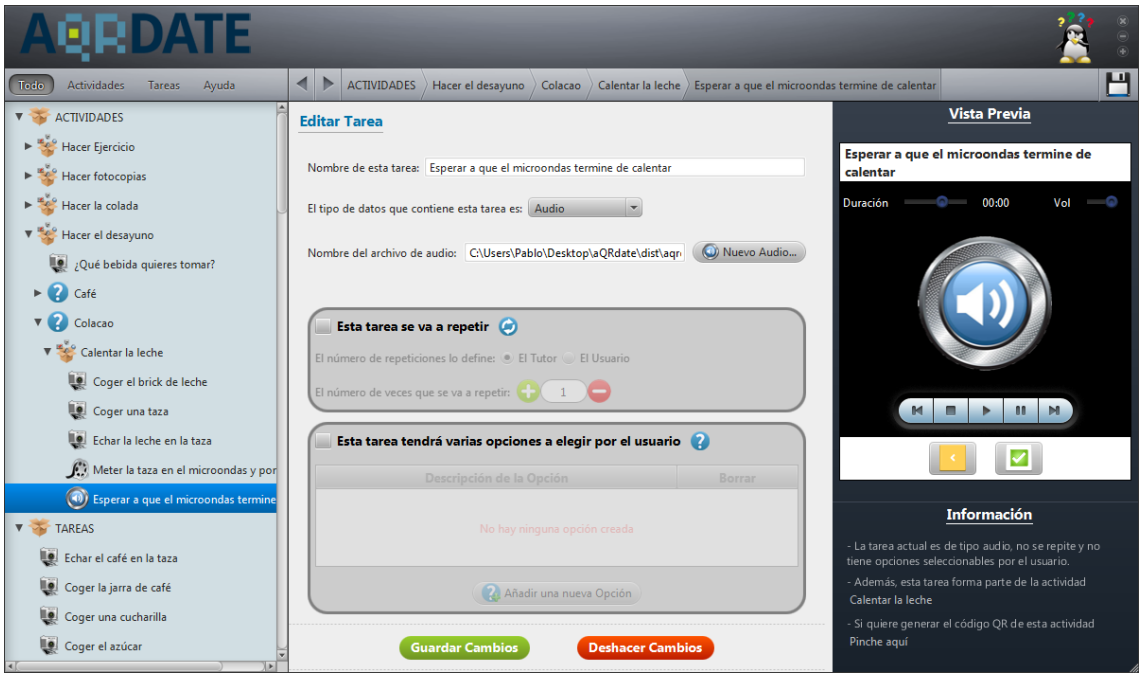


Figura 5.56: Tarea de tipo audio llamada “Esperar a que el microondas termine de calentar”.

Se retira la taza del microondas y termina la actividad. El aspecto que tiene como resultado es:

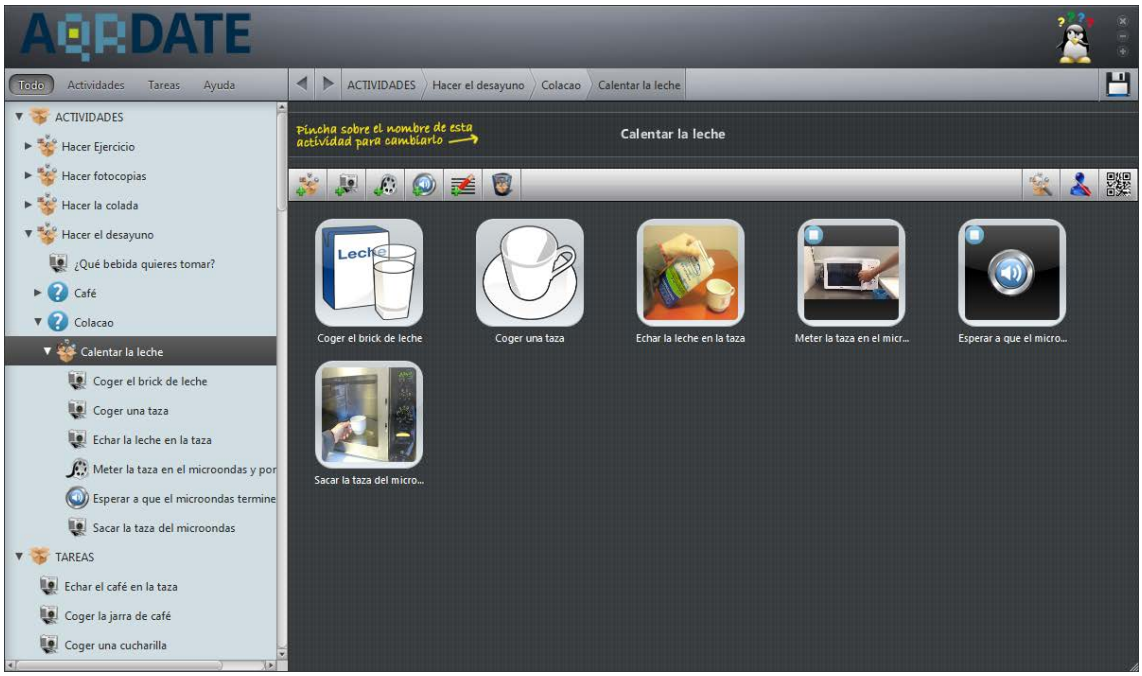


Figura 5.57: Actividad “Calentar la leche” terminada.

En la Figura 5.57 puede verse como, tanto en la tarea que tiene un vídeo como en la que tiene un archivo de audio aparece un símbolo de *stop.* Esto se debe a que si se mueve el ratón por encima de estas tareas, el contenido de las mismas se reproduce.

El último paso que queda pendiente dentro de la opción “Colacao” es crear una tarea en la que se muestre a una persona añadiendo el cacao a la leche.

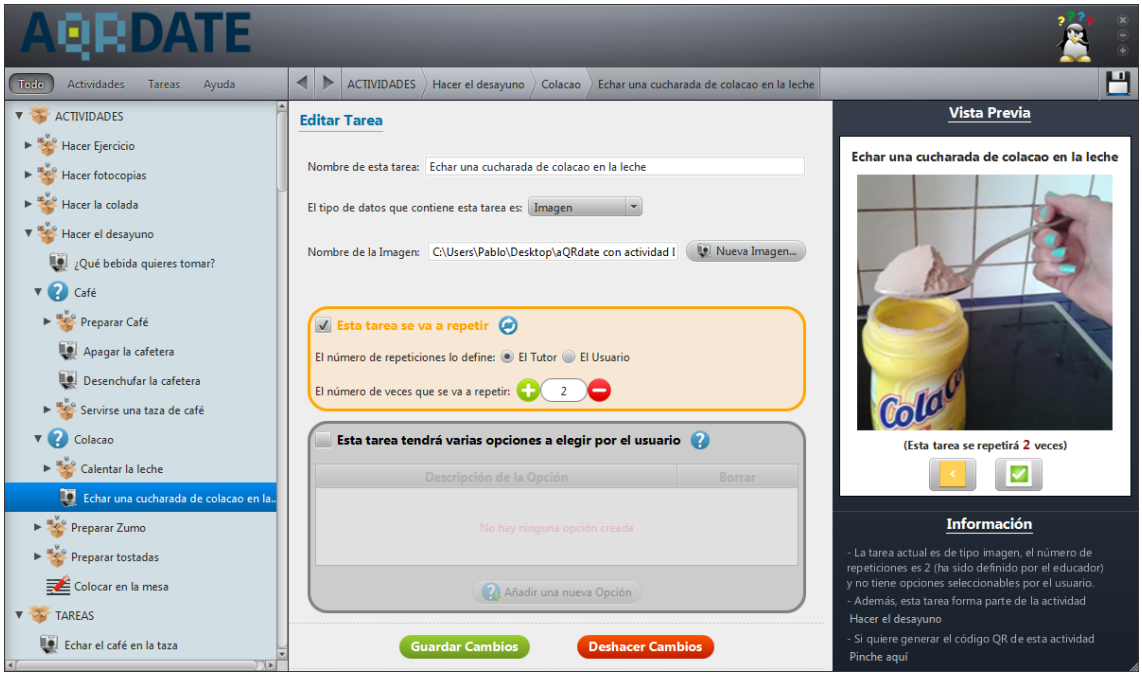


Figura 5.58: Tarea repetitiva llamada “Echar una cucharada de cacao en la leche”.

La opción “Colacao” una vez terminada es:

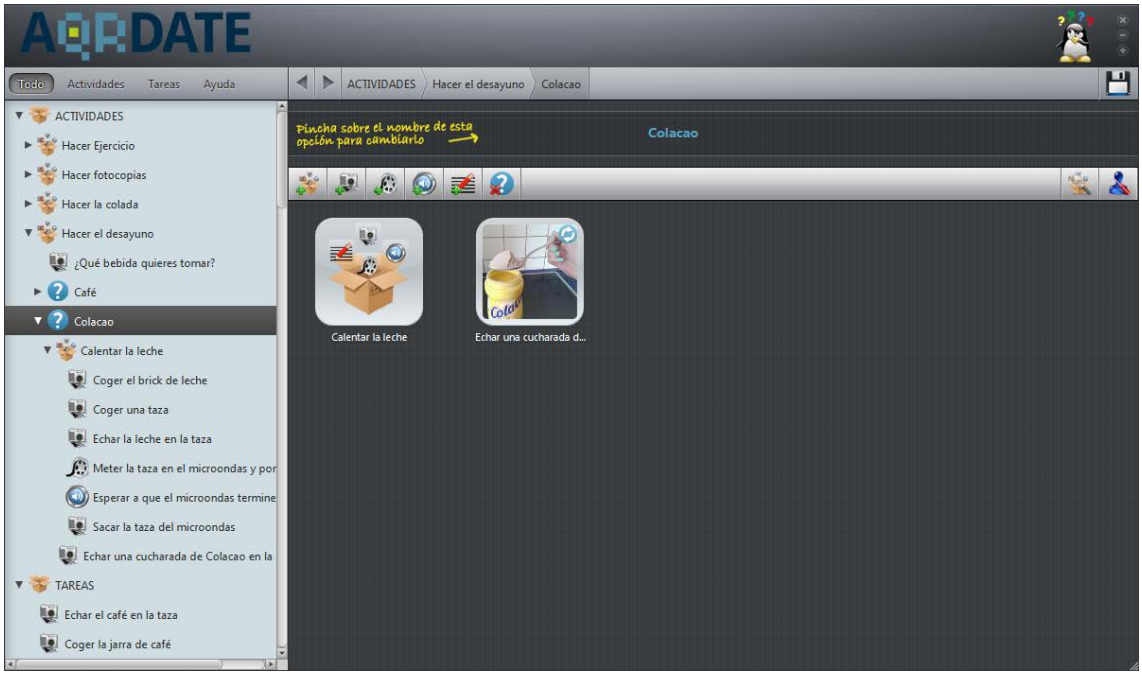


Figura 5.59: Imagen en la que se muestra el icono que representa a una tarea repetitiva.

En este caso el número de cucharadas (Figura 5.58) viene fijado por el personal médico que diseña esta tarea y por tanto el usuario no podrá modificarlo. En la Figura 5.59 se aprecia el símbolo que aparece en la tarea cuando es repetitiva (este mismo símbolo es el que tiene la tarea “Echar azúcar en la taza” que se describe en la Figura 5.52).

Una vez concluido el contenido de las dos opciones iniciales de la actividad, se añaden las dos actividades pendientes: “Preparar Zumo” y “Preparar tostadas”:

Figura 5.60: Imágenes de las actividades “Preparar Zumo” y “Preparar tostadas” terminadas.

Por último, se inserta una tarea de tipo descripción textual en la que se explica los pasos a seguir para colocar todo el desayuno en la mesa:

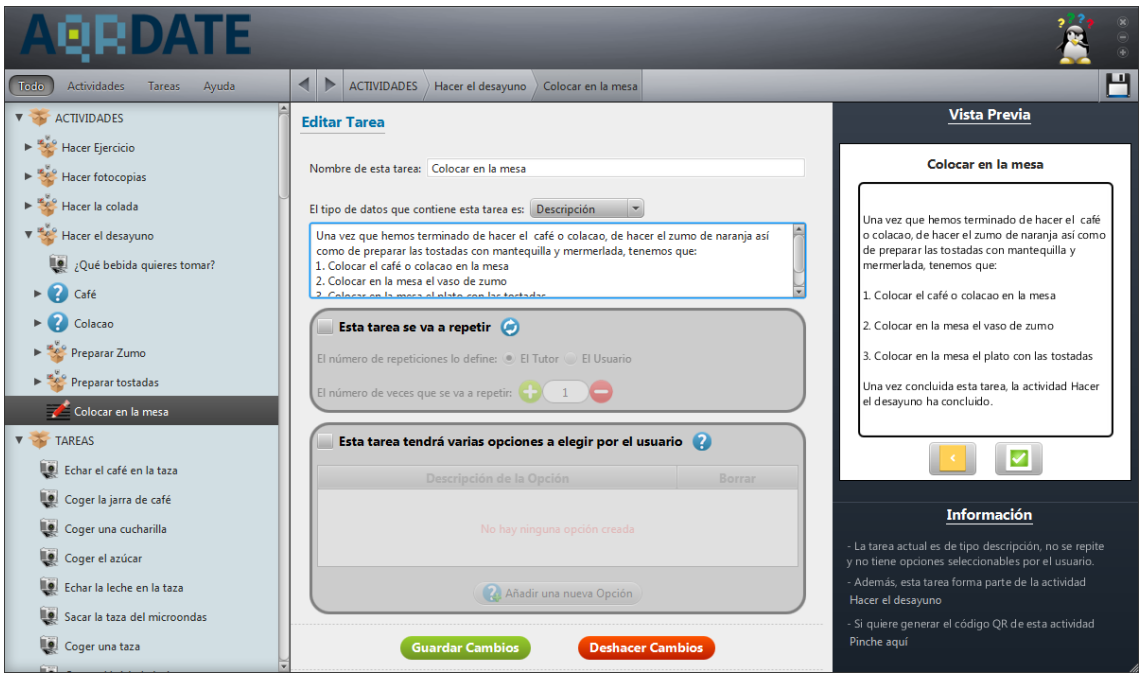


Figura 5.61: Tarea de tipo descripción textual llamada “Colocar en la mesa”.

El resultado final de la actividad desayuno se adjunta en la siguiente imagen.

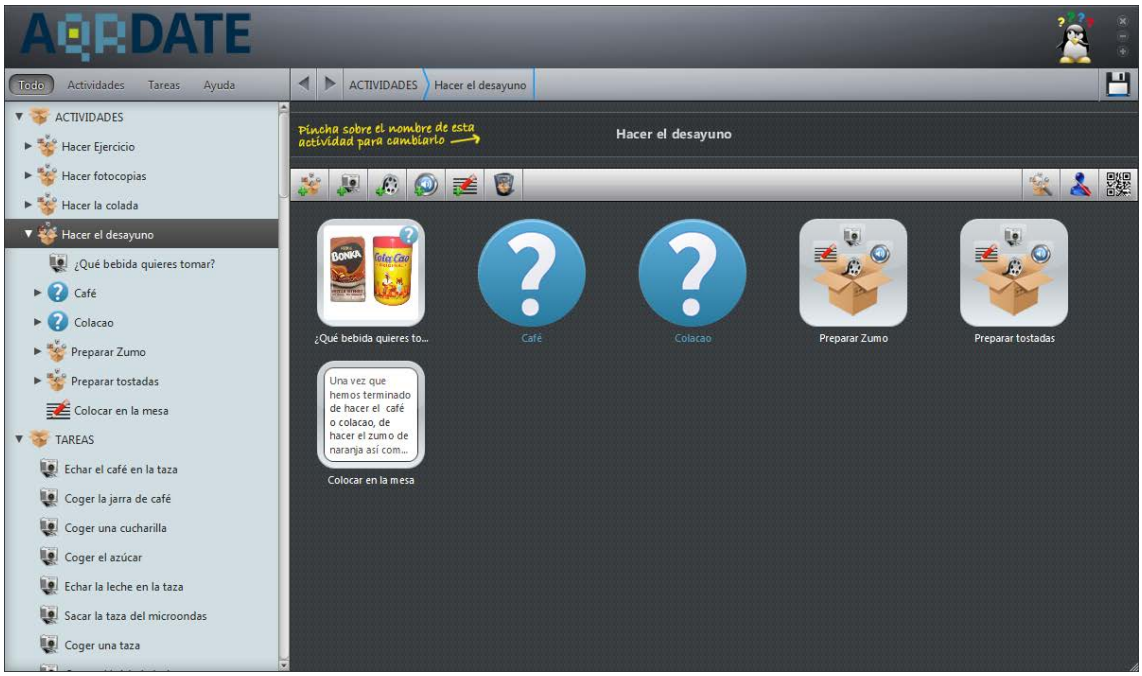


Figura 5.62: Actividad “Hacer el desayuno” terminada.

5.3.2. Segundo Ejemplo: Editar una actividad

Una actividad o una tarea que ya ha sido creada puede modificarse en cualquier momento. Para ilustrar este ejemplo se retoma la actividad diseñada en el primer ejemplo.

En la opción “café”, al servirse una taza de café no se añade leche. Por ello, se copia la actividad “Calentar la leche” (contenida en la opción “Colacao”) y se pega en el destino. Se puede ver que cuando se copia, se refleja en la tarea mediante un icono:

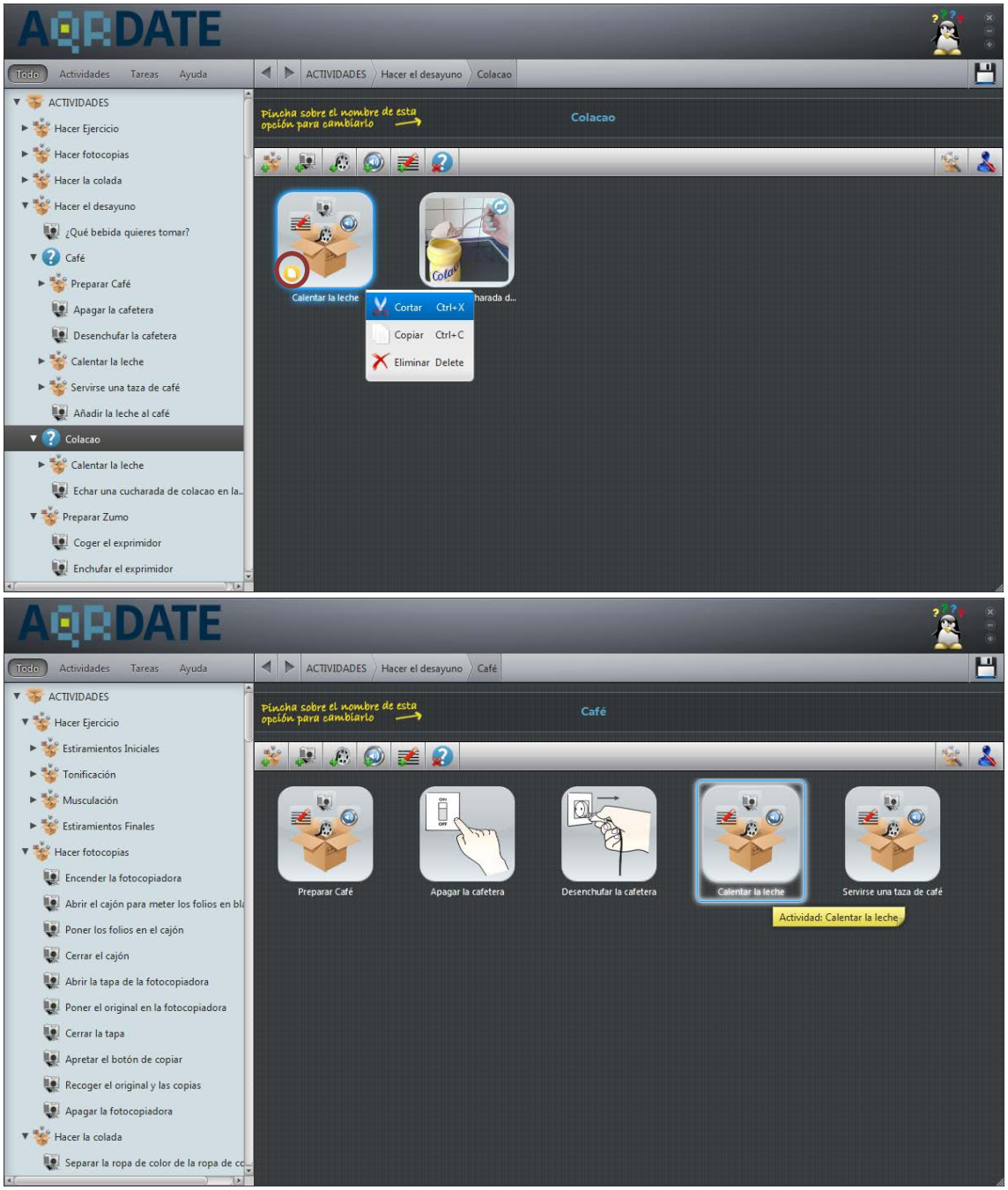


Figura 5.63: La primera imagen muestra el icono de tarea copiada y la segunda es el destino.

Para copiar, cortar, pegar y eliminar un contenido de la aplicación, se aprieta el botón secundario del ratón y se escoge la opción a realizar o, una vez seleccionada la actividad o tarea, se presionan las teclas Ctrl+C, Ctrl+X, Ctrl+V o Ctrl+Supr respectivamente.

Ahora se añade una tarea que indique al usuario que tiene que añadir la leche al café, pero ésta aparece situada en un lugar erróneo (debería ir después de servirse el café). Para ordenarla, se pincha en ella y sin soltar el ratón se arrastra a la posición deseada:

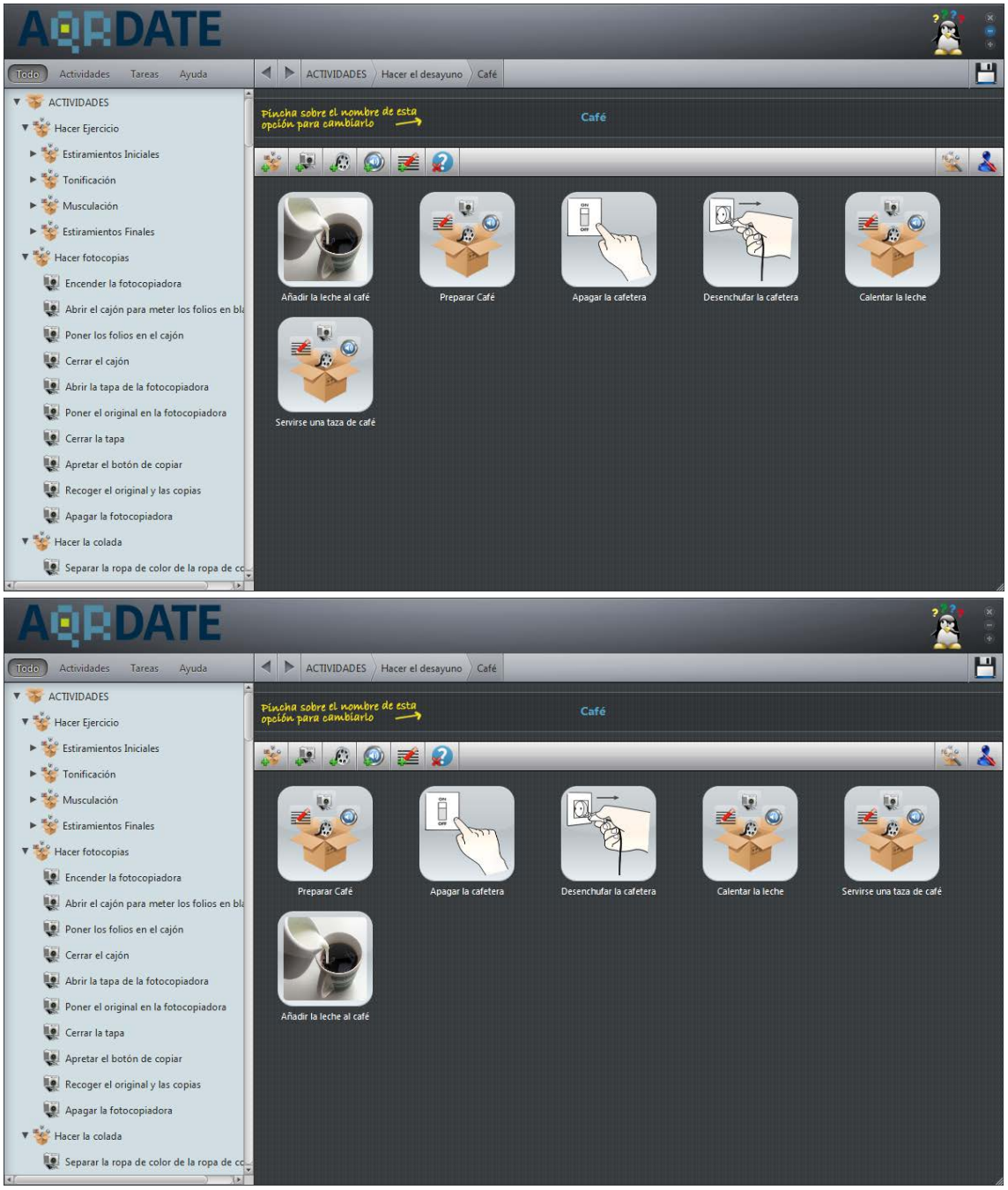


Figura 5.64: En la primera imagen se ve que se crea una tarea y en la segunda es colocada.

Antes de crear la actividad “Hacer el desayuno”, se diseñó la actividad “Preparar Fruta” para posteriormente introducirse dentro del desayuno. Como se quiere que esta actividad sólo forme parte del desayuno, se corta del panel de Actividades y se pega en la posición deseada:

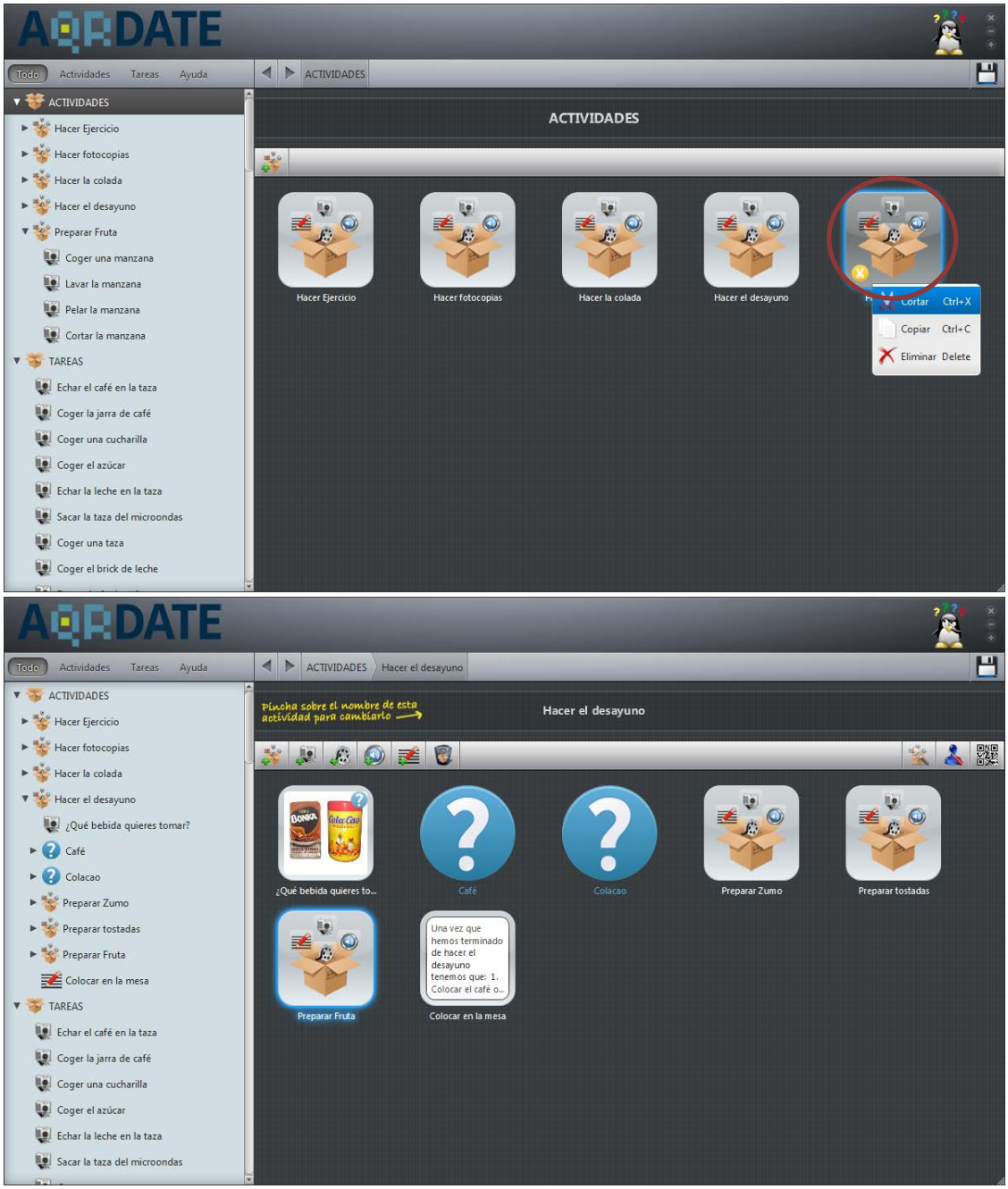


Figura 5.65: En la primera imagen se corta la actividad y en la segunda se muestra el destino.

Al igual que en el caso de copiar, cuando una tarea o actividad es cortada, se refleja en el objeto a cortar mediante un nuevo icono que representa este acto. Además cuando se corta, el fondo del objeto cambia de color para indicarlo (Figura 5.65).

Por otra parte, en la actividad “Preparar tostadas” hay una tarea que no se necesita, titulada “Poner el temporizador del tostador”, pues el tiempo de espera es automático. Por ello se elimina la tarea:

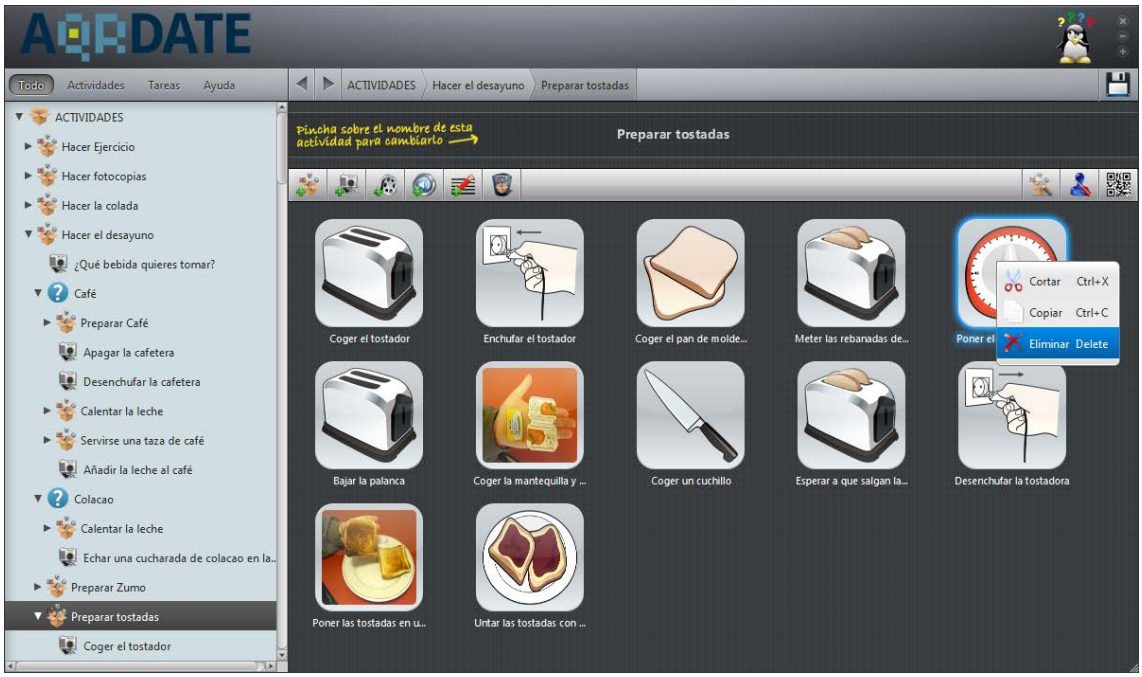


Figura 5.66: Eliminando una tarea.

Del mismo modo que se borran las tareas, se podrían borrar actividades:

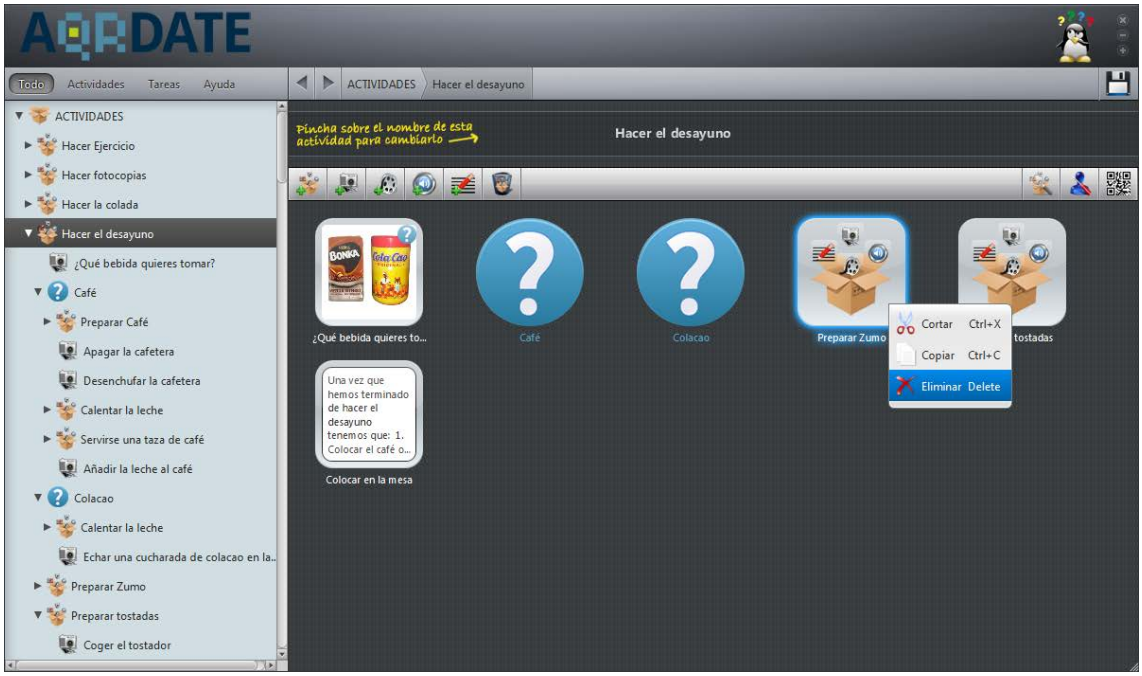


Figura 5.67: Eliminando una actividad.

5.3.3. Tercer Ejemplo: Personalizar una actividad

Se selecciona la actividad “Hacer el desayuno” para personalizarla. Una vez que se elige esa opción se abre un panel de usuarios. Por defecto, siempre aparece el usuario genérico que contiene todos los pasos que han sido diseñados en esta actividad.

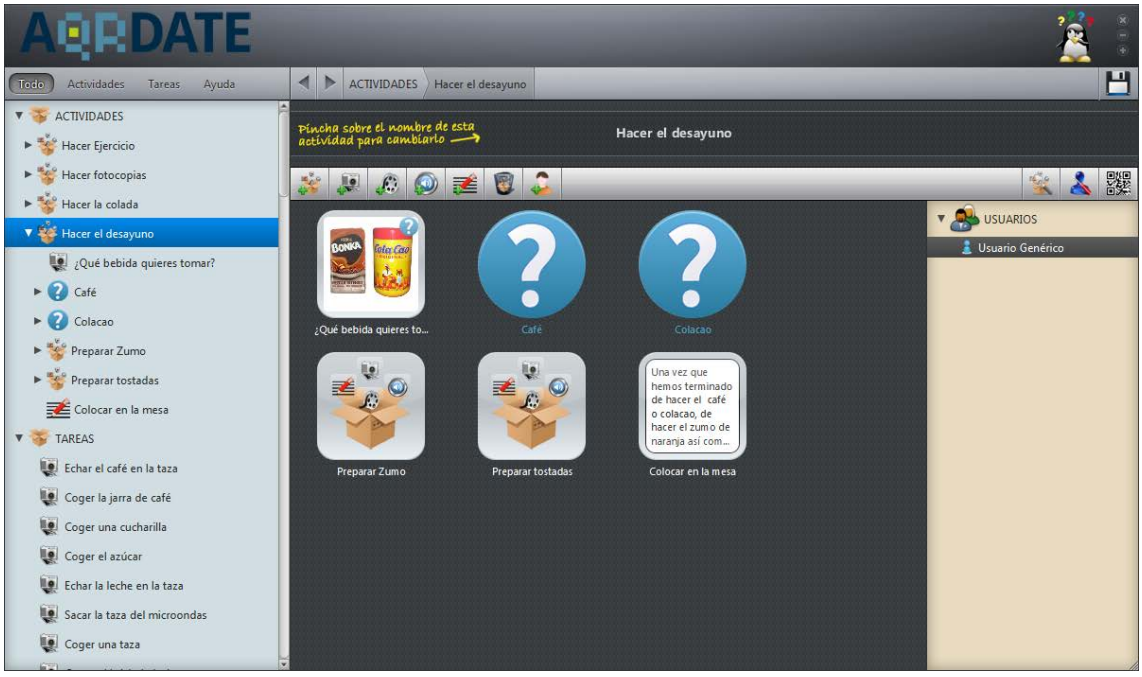


Figura 5.68: Impresión de pantalla de la aplicación con la opción de personalizar seleccionada.

Se introduce el nombre del usuario al cual se le quiere personalizar esta actividad.

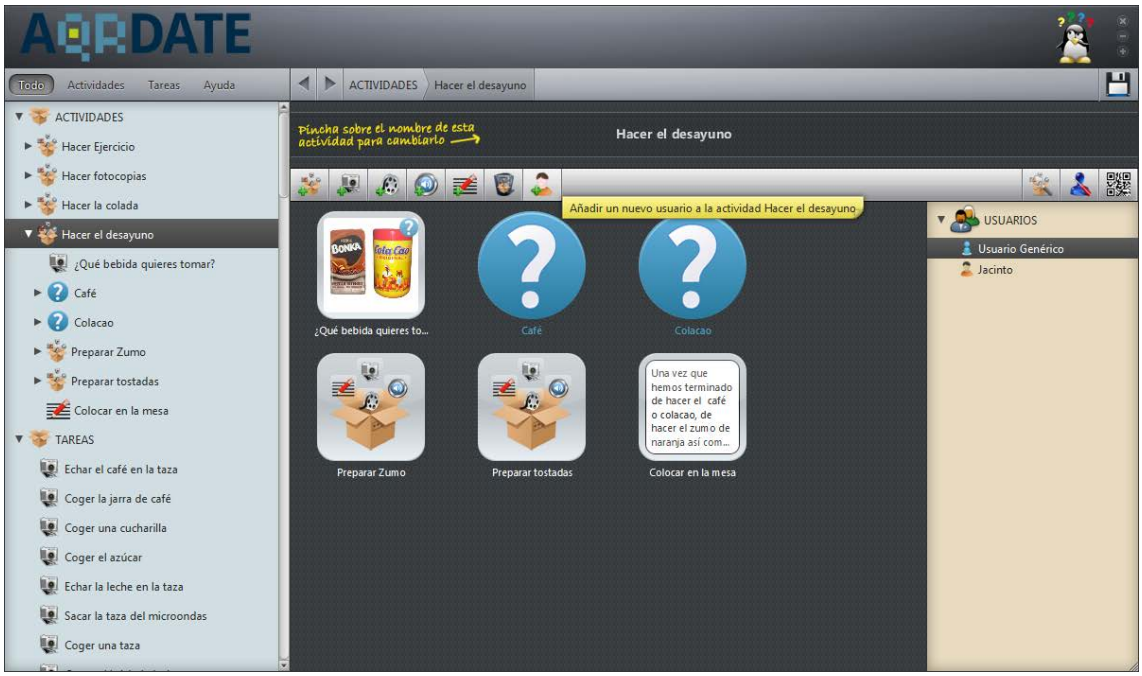


Figura 5.69: Imagen que muestra la inserción de un nuevo paciente.

Ahora es el momento de eliminar aquellas tareas o actividades que no se quieran mostrar a este usuario. Para el paciente Jacinto, se elimina la actividad “Preparar zumo”.

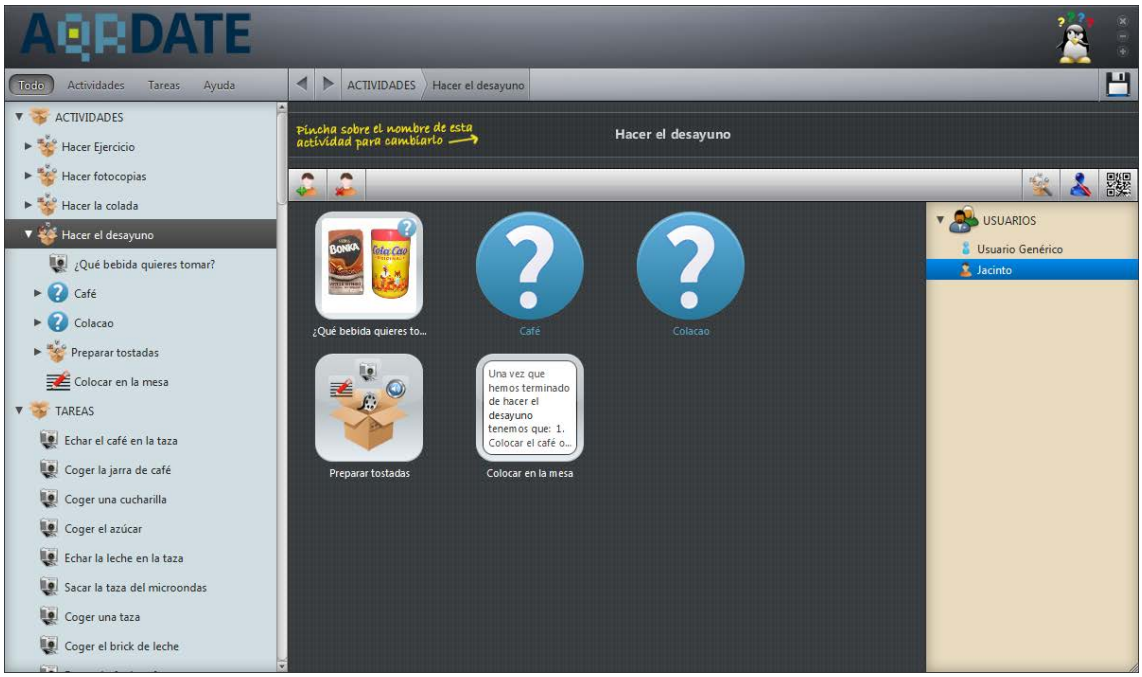


Figura 5.70: Actividad personalizada para el usuario “Jacinto”.

El proceso descrito en este ejemplo se repetirá tantas veces como personalizaciones a los usuarios se necesiten hacer.

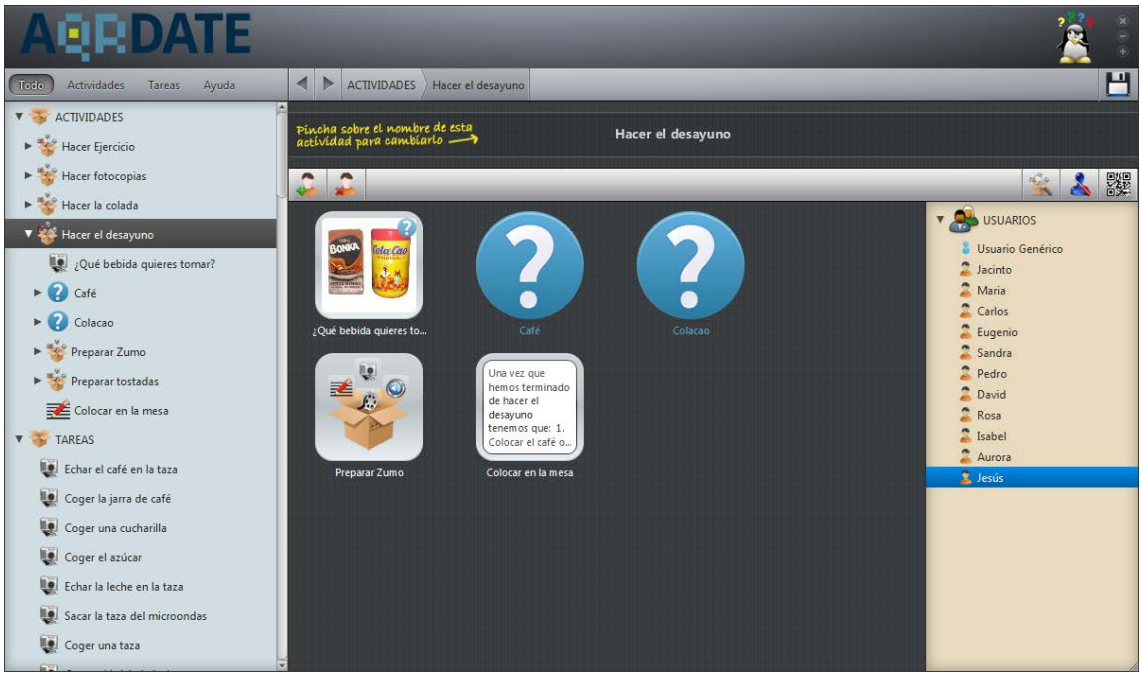


Figura 5.71: Actividad “Hacer el desayuno”con nuevos usuarios.

5.3.4. Cuarto Ejemplo: Vista previa

En esta ocasión se utiliza la opción “Vista Previa” para la actividad “Hacer el desayuno”. El contenido mostrado es el mismo que se muestra en el teléfono móvil del paciente. Para llevar a cabo esta opción, hay que elegir previamente el usuario, que ha sido creado en el sistema, sobre el que se desea ver su vista previa. Para este ejemplo se utilizara el “Usuario genérico”.

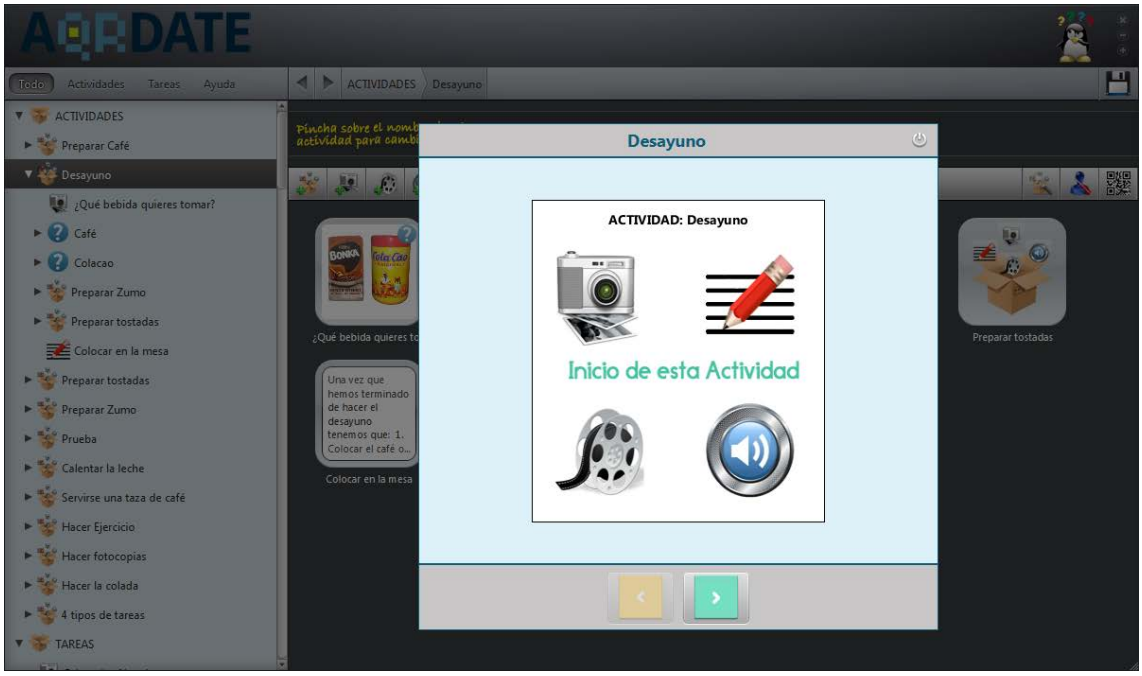


Figura 5.72: Inicio de la vista previa para el “Usuario Genérico”.

A continuación se muestra la vista previa de la actividad:

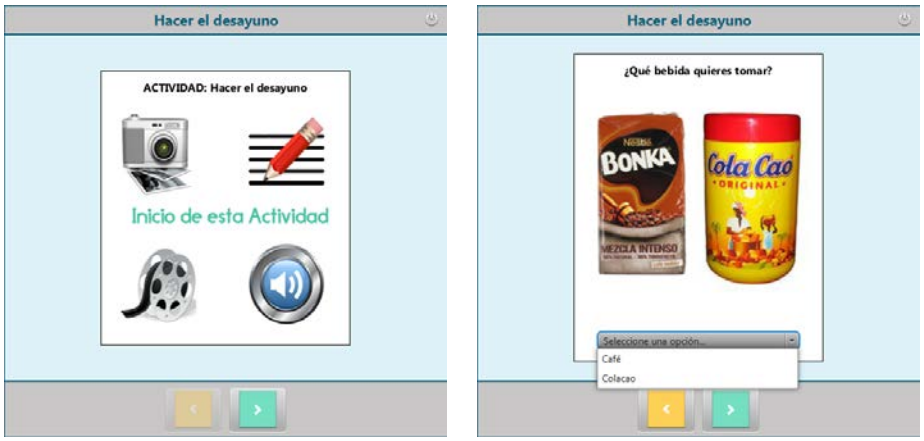
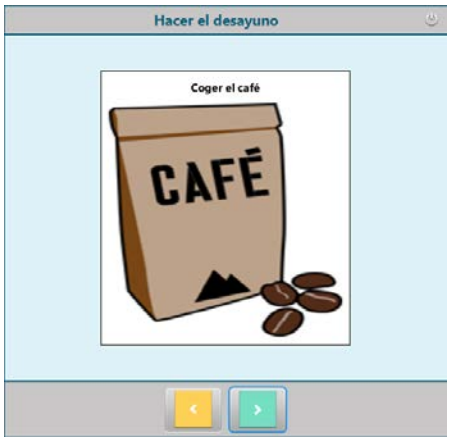
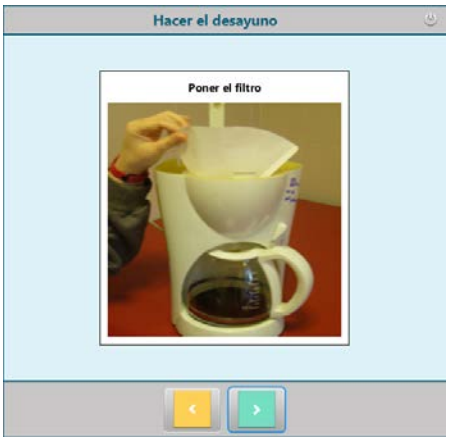
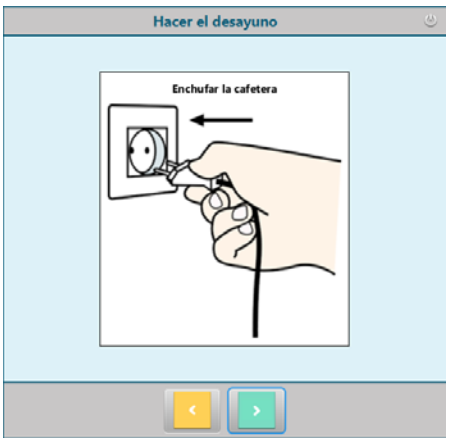


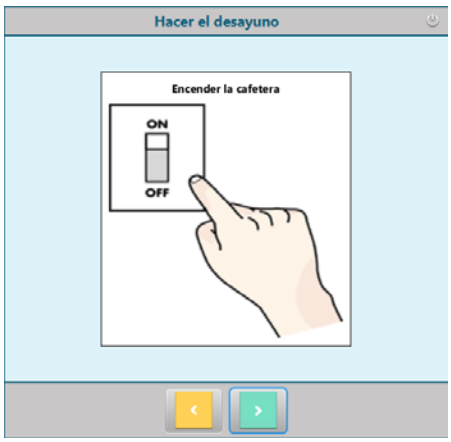
Figura 5.73: Primeras tareas de la actividad “Hacer el desayuno”.

Si se escoge la opción de “café”:









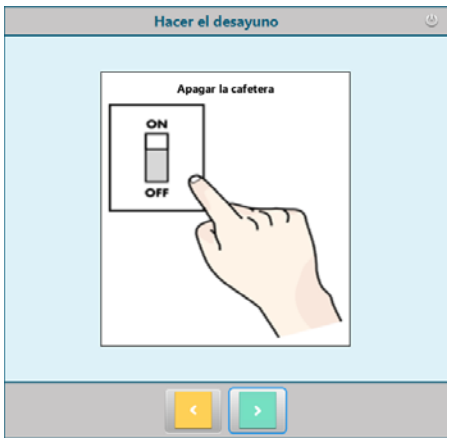


Figura 5.74: Tareas mostradas al usuario si escoge la opción “café”.

Si la opción seleccionada es: “Colacao”:

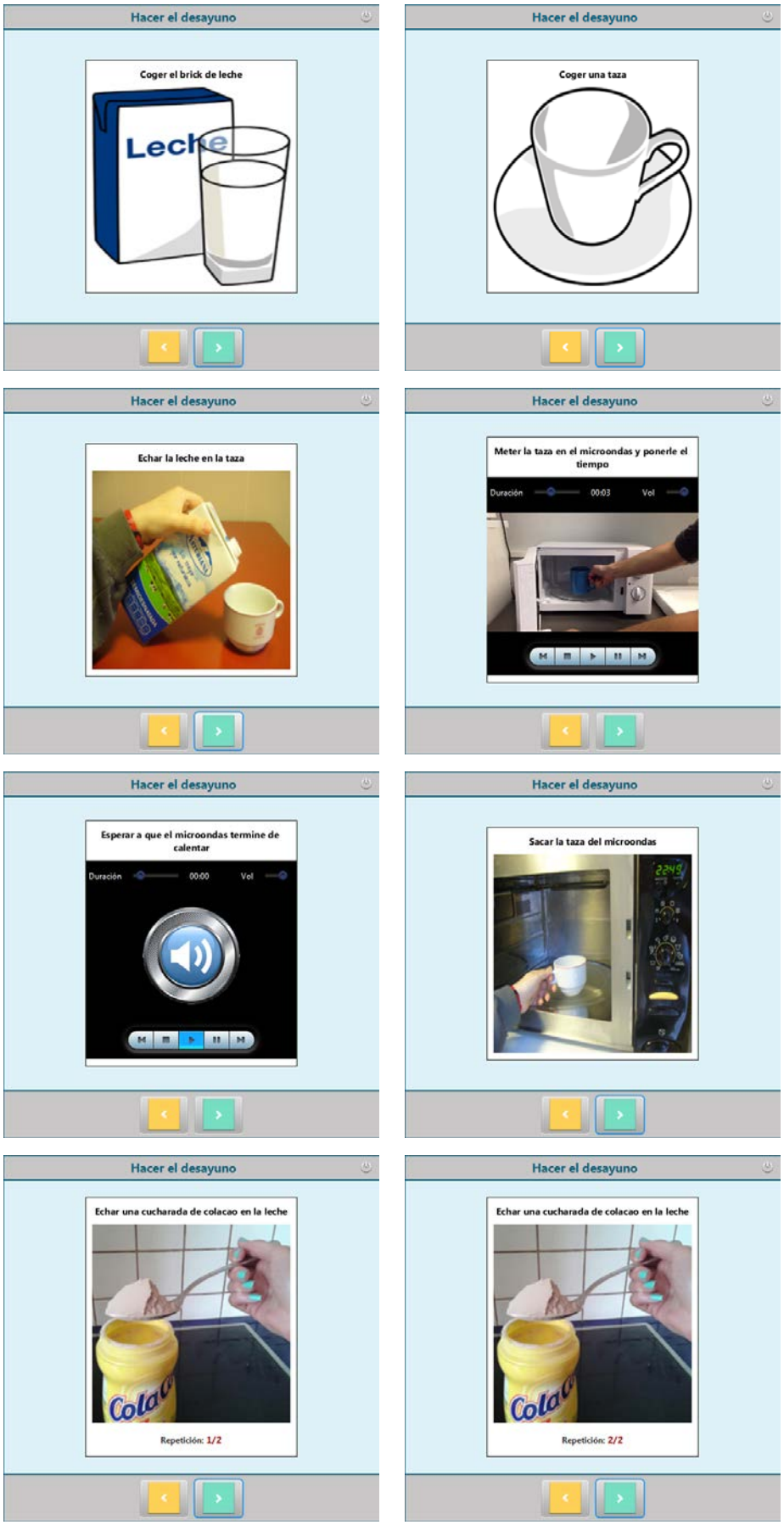


Figura 5.75: Tareas que son mostradas al usuario en caso de seleccionar la opción “Colacao”.

Una vez mostrada la información de cada una de las dos opciones, la vista previa continúa con la actividad “Preparar Zumo”:

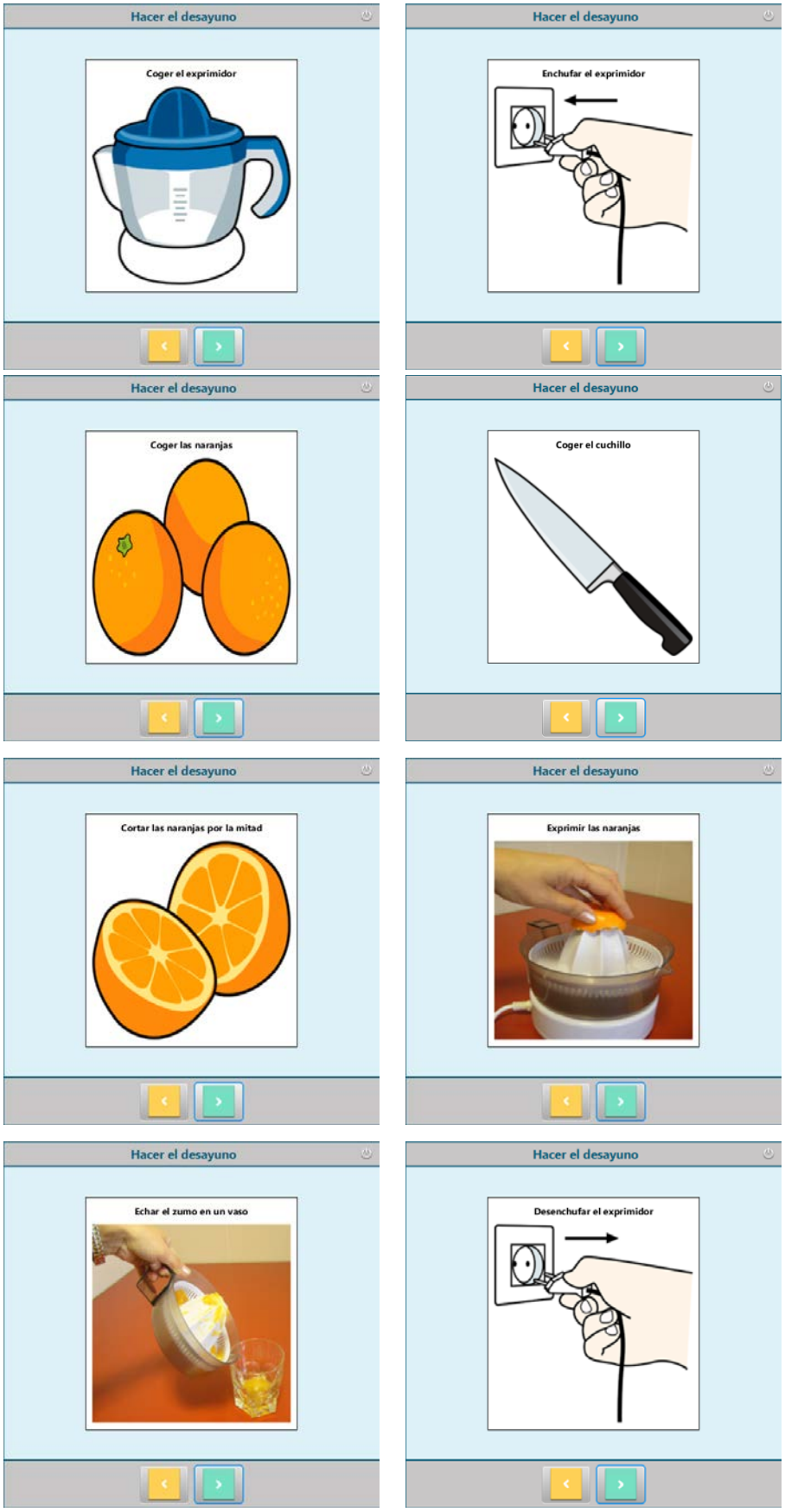
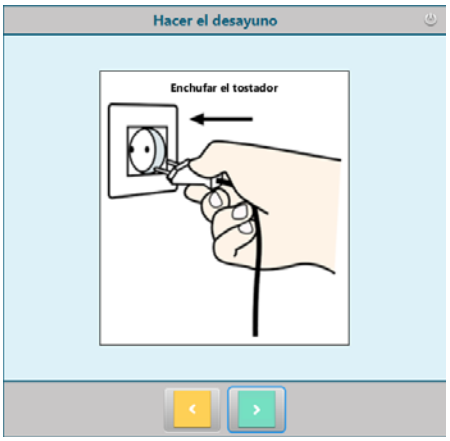
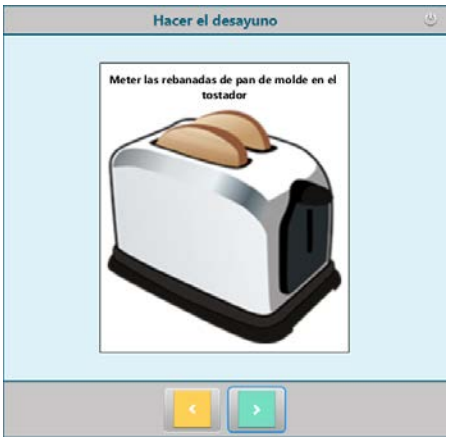
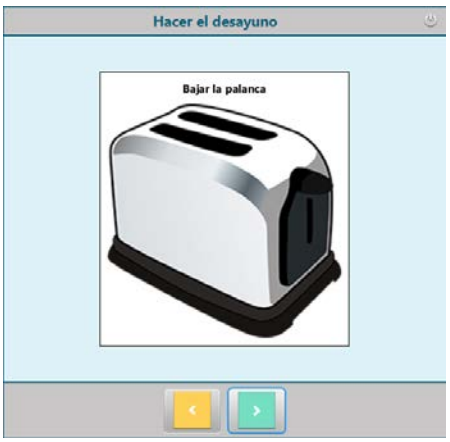


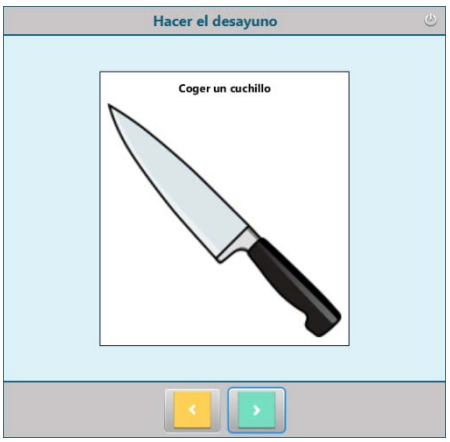
Figura 5.76: Vista Previa de la actividad “Preparar zumo”.

Se sigue con la última subactividad: “Preparar tostadas”:









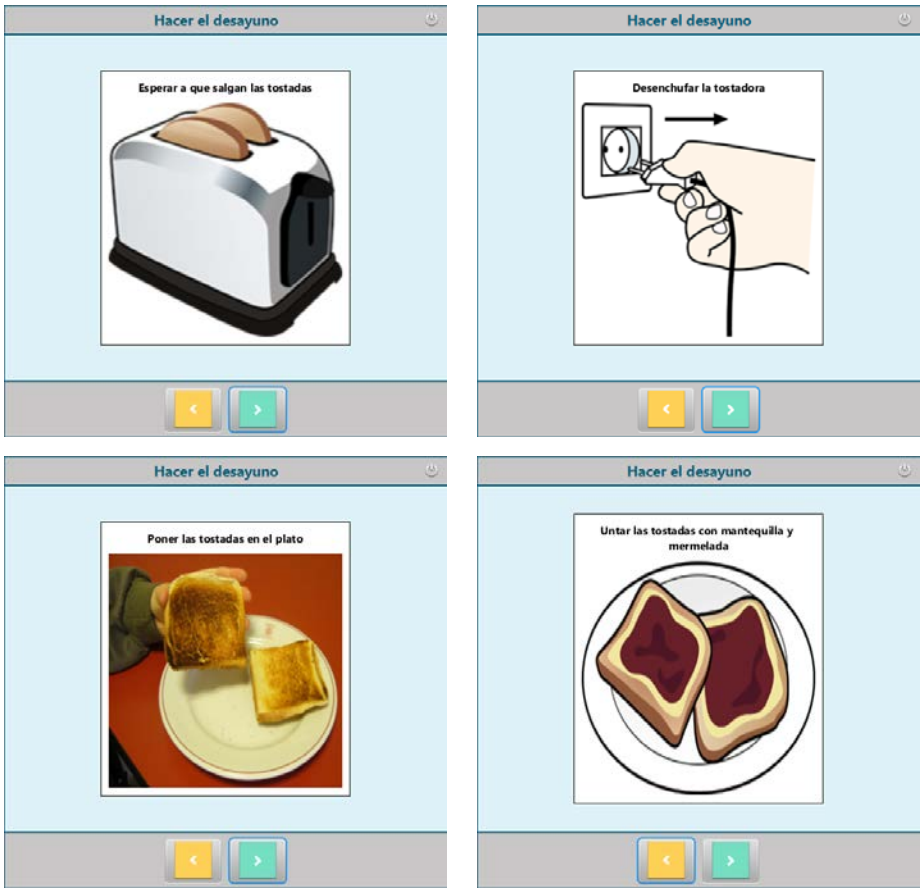


Figura 5.77: Vista previa de la actividad “Preparar tostadas”.

Para terminar la actividad “Hacer el desayuno”, se muestra la última tarea:

Figura 5.78: Última tarea de la actividad “Hacer el desayuno”.

En este caso la vista previa ha sido mostrada para el “Usuario Genérico” pero se podría haber escogido otro usuario como por ejemplo “Jacinto” (usuario introducido en el tercer ejemplo de este apartado).

5.3.5. Quinto Ejemplo: Generar el código QR

Una vez que la actividad ha sido creada, para que el paciente la pueda visualizar en el dispositivo móvil, tiene que escanear el código QR de la actividad que quiere realizar. Por ello, esta herramienta de autor dispone de una sencilla opción para generar este código:

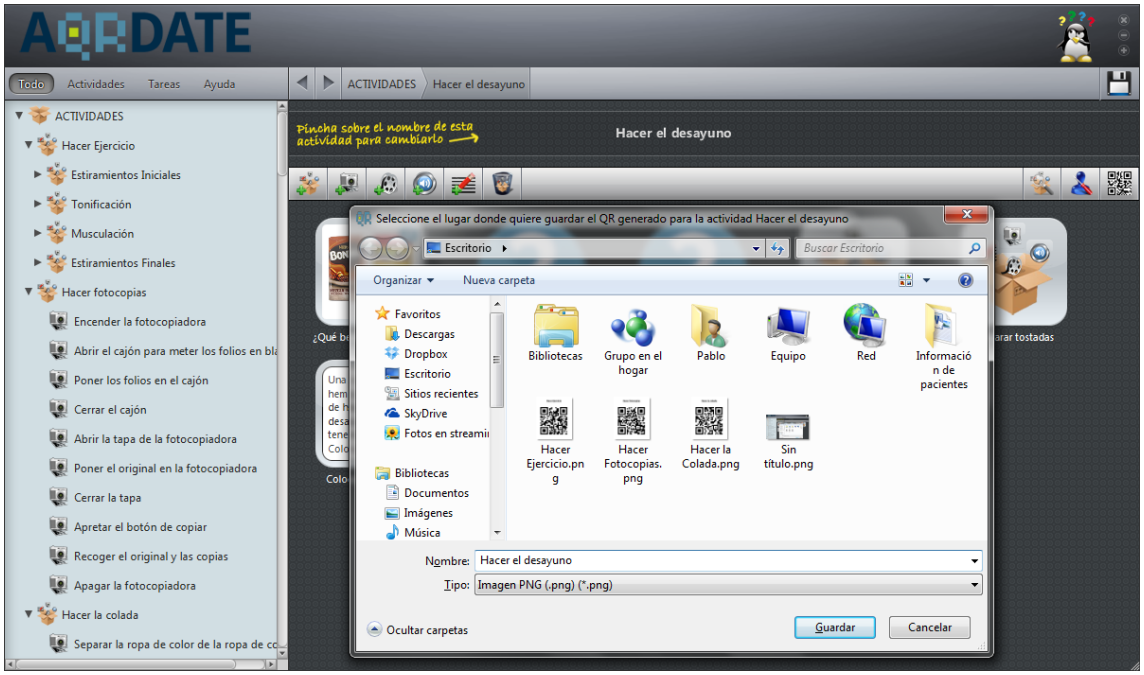


Figura 5.79: Imagen que muestra el proceso de la generación del código QR.

El resultado del código QR obtenido, que el personal médico deberá de imprimir y situar en el lugar donde la actividad se realice, es:

Figura 5.80: Código QR de la actividad “Hacer el desayuno”.

5.3.6. Sexto Ejemplo: Menú de ayuda

Uno de los objetivos que debía de cumplir este trabajo era disponer de un menú de ayuda en donde el usuario pudiese consultar todos los temas relacionados con la creación de actividades. Por ello se creó un menú de ayuda que es accesible desde dos lugares distintos de la interfaz:

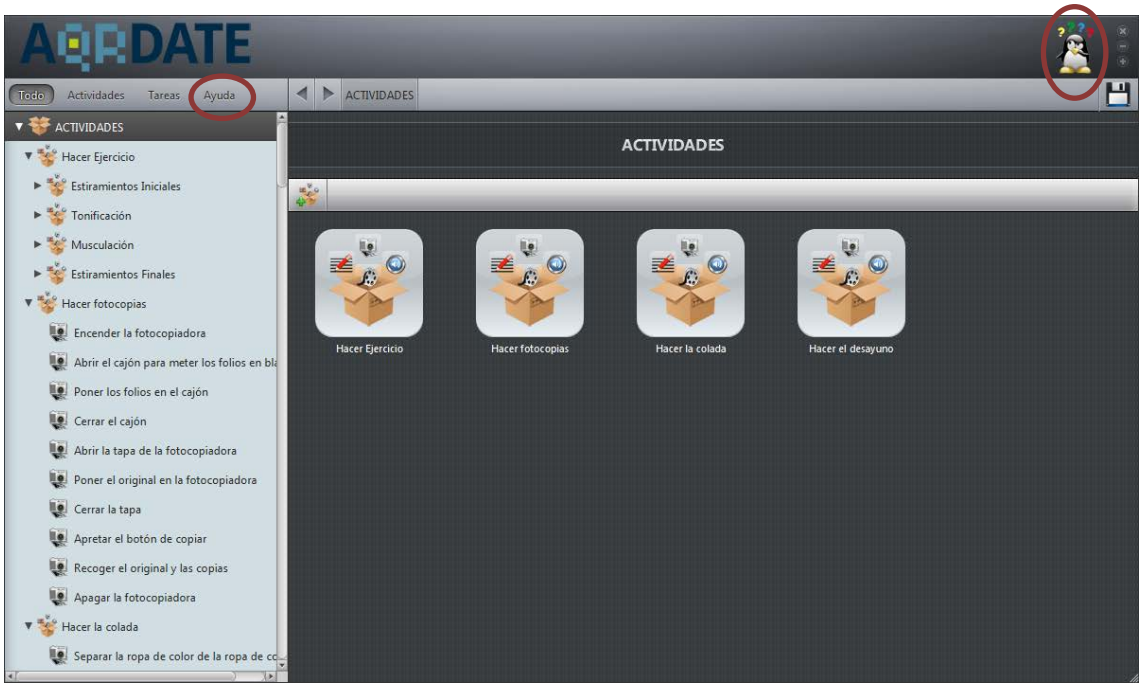


Figura 5.81: Imagen que muestra las dos posibles opciones para acceder al menú de ayuda.

Una vez escogido, el resultado es:

Figura 5.82: Imagen en donde se ve el menú de ayuda.

Como se aprecia en la imagen anterior, hay tres grandes grupos de ayuda a elegir por la persona que esté usando la aplicación. Ejemplos de escoger algún tema de estos grupos (“Ayuda sobre las actividades”, “Ayuda sobre las tareas” e “Información adicional”), se adjuntan a continuación:

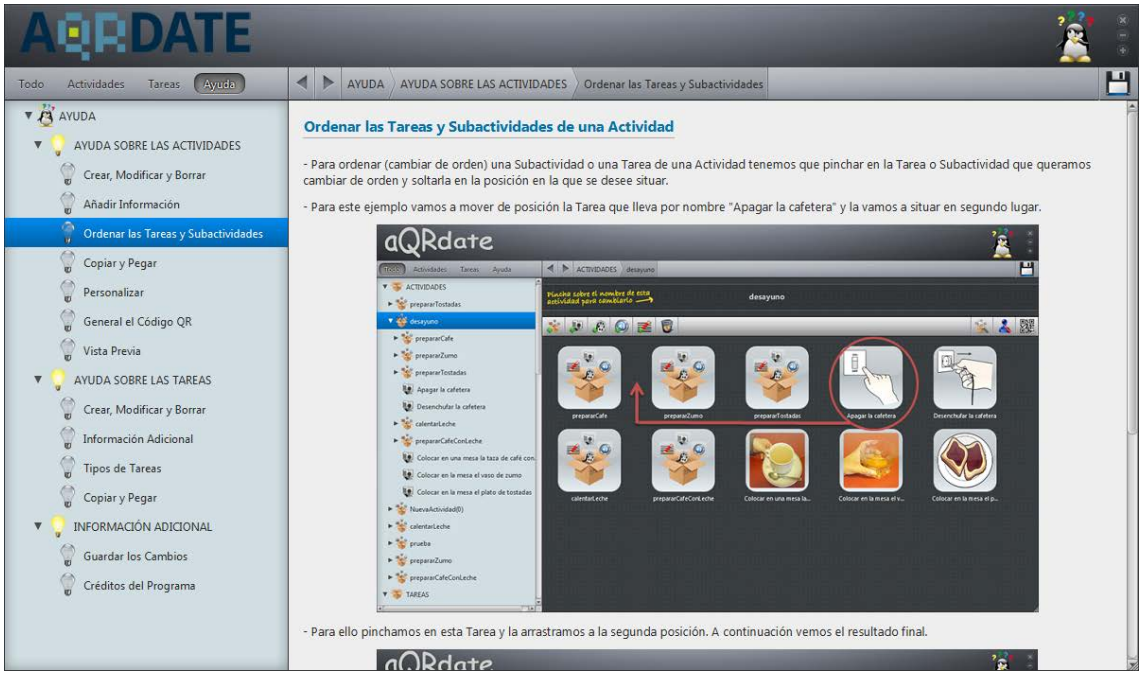


Figura 5.83: Tema escogido del grupo “Ayuda sobre las actividades”.

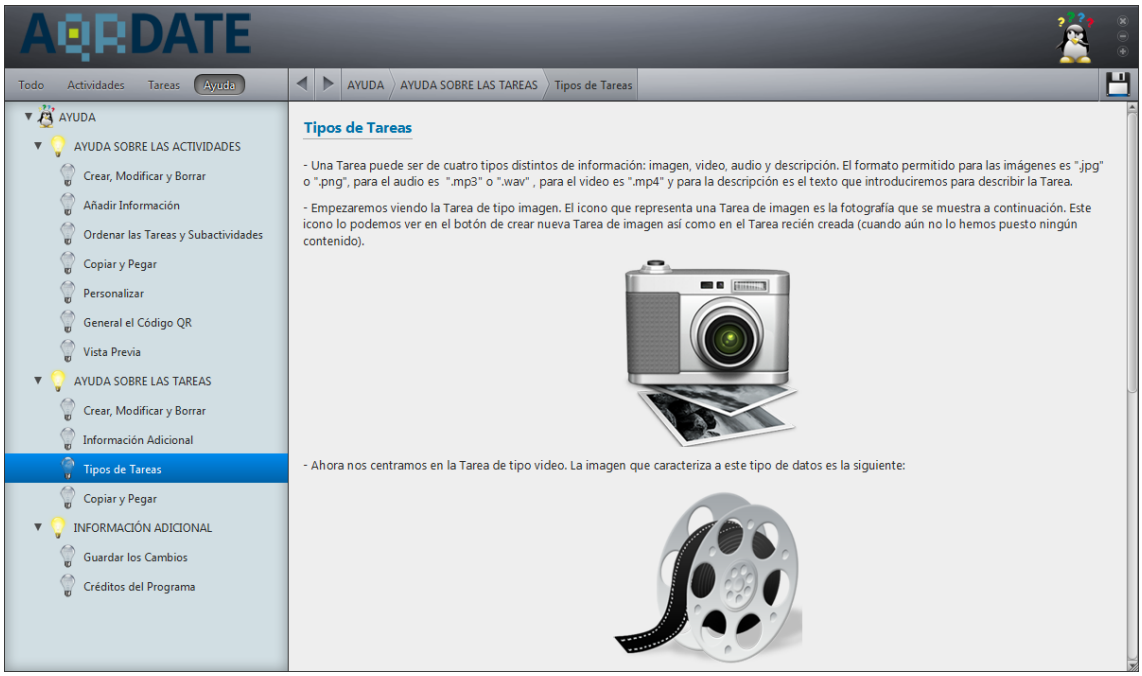


Figura 5.84: Información relativa al grupo “Ayuda sobre las tareas”.

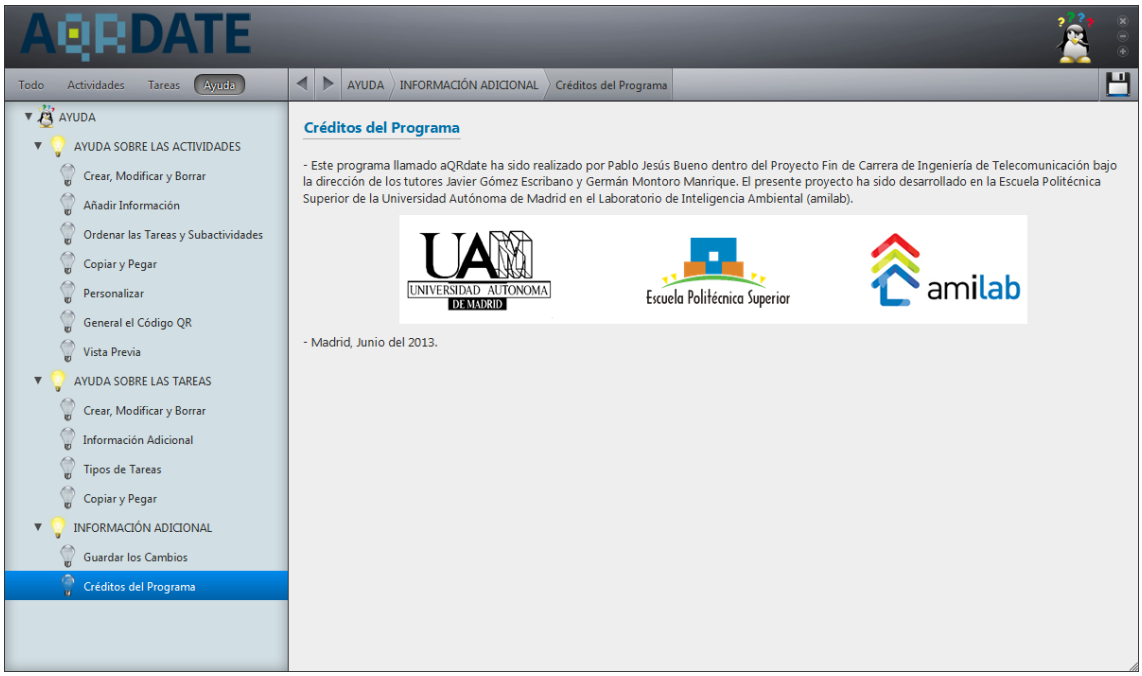


Figura 5.85: Información sobre el último grupo de ayuda “Información adicional”.

6

CONCLUSIONES Y TRABAJO FUTURO

6.1. CONCLUSIONES

Este Proyecto Fin de Carrera parte de un trabajo previo [6] llamado aQRdate, que está diseñado para personas con discapacidad intelectual cuya finalidad es la realización de tareas de la vida diaria. Este tipo de actividades requieren el uso de diferentes destrezas que estas personas pueden carecer, lo que supone que presenten dificultades en su ejecución. Por ello, en muchas ocasiones, es la ayuda de sus cuidadores la que permite realizarlas. Este sistema tiene como objetivo proporcionar un soporte a personas con discapacidad cognitiva para que adquieran las destrezas necesarias, sin la intervención directa y constante del educador.

En concreto, aQRdate pretende que gracias al uso de teléfonos móviles y códigos QR, ayudar a pacientes en la realización de estas tareas mediante el uso de manuales adaptados que les guíen paso a paso. El funcionamiento es el siguiente: el usuario apunta con el terminal móvil al código QR, situado en aquel lugar donde se desarrolle la actividad que se pretende realizar, y una vez escaneado, se muestra al usuario todos los pasos necesarios que tiene que ejecutar para completar la tarea.

Gracias a la arquitectura del sistema descrito anteriormente, la información de las tareas está disponible para los usuarios a través de internet. La descripción de las actividades permite varios tipos de contenido (imágenes, vídeos, audios, descripciones textuales) que contendrá cada tarea con el fin de resultar lo más sencillo posible de comprender. Además, esta aplicación móvil permite la posibilidad de adaptar el contenido a los diferentes usuarios, ofreciendo más o menos ayuda según se requiera.

Sin embargo, el sistema aQRdate sólo estaba formado por la aplicación móvil y el servidor donde se almacenaba toda la información relacionada con las actividades. Cuando los profesionales que atienden a este tipo de pacientes querían realizar modificaciones en el sistema, no podían pues no tienen los suficientes conocimientos informáticos como para poder actualizar la información en el servidor así como para generar el lenguaje descriptivo que define a cada actividad (explicado en el capítulo 3.4). Como solución al problema se plantea este PFC, en el cual se desarrolla una herramienta de autor para el diseño de actividades cotidianas, destinado al personal encargado de pacientes con daño cerebral adquirido, de una forma sencilla e intuitiva.

Este programa se ha diseñado para ordenadores convencionales y los únicos requisitos que se necesitan son: tener instalado el JRE (*Java Runtime Enviroment)* puesto que está diseñado en un lenguaje perteneciente a la familia de Java y conexión a internet.

Con esta herramienta es posible introducir nuevas actividades al sistema paso a paso, modificar las ya existentes así como borrarlas. En cada paso de la actividad, llamado tarea, se puede definir el título de la misma así como el tipo de contenido que se quiera mostrar: audio, vídeo, imagen o descripción textual. Además en cada uno de ellos se ofrece la posibilidad de que se repita (el número de veces puede ser escogido bien por el cuidador o bien por el usuario de la aplicación móvil) o que en él aparezca una lista de opciones de las cuales el usuario final tenga que elegir una (modificando por tanto el curso de la actividad). Otra de las opciones disponibles en este *software* es la posibilidad de reutilizar actividades para crear otras nuevas, de esta forma el usuario de este programa puede ahorrase el tiempo de volver a diseñarla de forma innecesaria.

Como resultado final de la actividad ya definida, el sistema cuenta con la opción de generar su correspondiente código QR. Una vez creado, el personal médico deberá imprimirlo y situarlo en el lugar donde se realice la actividad para que los usuarios puedan escanearlo con su teléfono móvil.

Además se dispone de una vista previa de cada actividad diseñada con el fin de que los cuidadores vean en todo momento el resultado de la misma forma que el paciente lo vería en el teléfono móvil.

Una de las posibilidades más interesantes que ofrece esta herramienta es poder personalizar cada actividad a usuarios concretos. De esta forma se puede eliminar aquella información que, si bien se encuentra definida para un usuario genérico, para otro concreto no es necesario que aparezca.

Por último, se encuentra disponible un manual de ayuda al usuario para poder consultar cualquier duda que pueda surgir durante el uso de esta aplicación.

A la vista del resultado final, la herramienta de autor que se ha implementado cumple con los objetivos fijados, y por tanto puede utilizarse para que el personal médico se encargue de diseñar en ella las actividades del ámbito diario que posteriormente realizarán los pacientes con daño cerebral adquirido usando el terminal móvil.

6.2. TRABAJO FUTURO

Para mejorar el sistema presentado, se proponen una serie de aspectos que sirvan como líneas de trabajo futuro:

- **Evaluación con los usuarios**: con el fin de probar la herramienta de autor realizada, se deberían realizar diferentes pruebas con los usuarios a los que va dirigido este programa. En ellas se podrían valorar múltiples aspectos tales como la usabilidad y la sencillez de la aplicación (aspectos fundamentales que ha de tener este *software*), así como el tiempo medio que se tarda en crear actividades o sugerencias aportadas por los usuarios.
- **Multimodalidad de los contenidos de las tareas**: hasta ahora cada tarea que forma parte de una actividad podía tener un solo tipo de contenido (una imagen, un audio, un vídeo o una descripción textual). Sin embargo, en ocasiones ayudaría la posibilidad de dejar escoger al usuario el tipo de contenido que desea obtener ya que por ejemplo una persona sorda no podría comprender el audio pero sí un descripción textual, del mismo modo una persona ciega no podría entender la información de un vídeo pero sí de una locución.
- **Ubiquidad de la herramienta de autor**: este programa, por el momento, sólo es posible ejecutarse en un ordenador. Para mejorar el acceso ubicuo al mismo

del personal médico desde cualquier lugar, sería necesario adaptar esta aplicación a diferentes plataformas como pueden ser las *tablets* o los teléfonos móviles. Para el caso de las *tablets* bastaría con introducir el programa en una página web. Para el caso de los teléfonos móviles, habría que adaptar toda la interfaz a las reducidas dimensiones que éste tiene.

- **Creación de una base de datos de pacientes**: por el momento, para cada actividad se añaden los usuarios a los que se les quiere personalizar esta actividad. Sin embargo no existe un registro único de cada paciente al sistema. Por ello se propone crear una base de datos en el programa y a partir de este registro personalizar las actividades con los usuarios con los que cuenta el sistema.
- **Obtención de los resultados por paciente**: gracias a la base de datos de pacientes que se acaba de describir sería posible obtener unas estadísticas de las ejecuciones de las actividades por usuario. De esta forma se podrían obtener los tiempos en los que se realizan las tareas, momentos de bloqueo de los usuarios, número de veces en los que el usuario tiene que retroceder en la aplicación, etc…
- **Adaptación inteligente**: una posible mejora sería la inclusión de un motor de adaptación que estudiase los registros de ejecución de cada actividad y de cada usuario y que con esa información propusiese la posible adaptación. Esta propuesta debería ser evaluada por el equipo clínico para que éstos decidieran la inclusión o no de la adaptación.
- **Inclusión de este proyecto en otros relacionados:** en el laboratorio se están llevando a cabo otros proyectos destinados al mismo tipo de personas (discapacidad intelectual) dedicado al guiado en interiores basado también en códigos QR así como otro de navegación en exteriores basado en navegación GPS. Un trabajo muy interesante sería englobar los tres trabajos en un sistema integral de la vida diaria de usuarios con estas características.

REFERENCIAS

[1] Centro de referencia estatal de atención al daño cerebral (CEADAC). Visto online en: http://www.ceadac.es/ceadac_01/index.htm.

[2] Defensor del Pueblo, «Informe sobre daño cerebral sobrevenido en España: un acercamiento epidemiológico y sociosanitario» 2006.

[3] S. Carmien, «Task support for people with cognitive impairments and their caregivers» *American Journal of Occupational Therapy,* vol. 14, pp. 1-4, 2004.

[4] J. Gómez, G. Montoro, P. A. Haya, X. Alamán, S. Alves, M. Martínez, E. Pascual, O. Robles y C. González, «aQRdate: assessing how ubiquitous computing can help people with acquired brain injury in their rehabilitation process» de *5th international symposium on ubiquitous computing and ambient intelligence (UCAmI)*, Riviera Maya, México, 2011.

[5] Iso9999, «Assistive products for persons with disability-classification and terminology» 2007.

[6] Javier Gómez Escribano, aQRdate: Sistema para el recordatorio de actividades de la vida diaria a personas con daño cerebral adquirido, Escuela Politécnica Superior de la Universidad Autónoma de Madrid. Madrid, España, 2011.

[7] I. 18004, Automatic identification and data capture thecniques - qr code 2005 bar code symbology specification, 2005.

[8] S. Carmien, «Socio-Technical Environments and Assistive Technology Abandonment» *Taylor & Francis LLC, CRC Press,* 2010.

[9] Carl Dea, JavaFX 2.0: Introduction by Example, Apress, 2011.

[10] J. Weaver, W. Gao, S. Chin, D. Iverson y J. Vos, Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology, Apress, 2012.

[11] J. L. M. &. T. E. H. O'Farrill, «Las herramientas de autor en el proceso de producción de cursos en formato digital» *Pixel-Bit.Revista de Medios y Educación,* nº 33, pp. 59-72, 2008.

[12] C. Catalina, GuiónEditor y HamWeb, Centro de Tecnología e Innovación (Teledomedia), 2002.

[13] N. Dabbagh, «Authoring tools and Learning Systems: A Historical Perspective» de *National Convention of the Association for Educational Communications and Technology*, Atlanta, 2001.

[14] J. Daccach T., «No aprenda tecnología» *Delta,* 2006.

[15] B. de Benito Crosetti, «Herramientas para la creación, distribución y gestión de cursos» *Edutec. Revista Electrónica de Tecnología Educativa,* nº 12, 2000.

[16] B. y. S. J. De Benito, «Webtools: Aplicaciones para sistemas virtuales de formación» de *Educar en Red. Internet como recurso para la Educación*, Málaga, 2002.

[17] T. Murray, S. Blessing y S. Ainsworth, Authoring tools for advanced technology learning environments. Toward cost-effective adaptive, interactive and intelligent educational software, Dordrecht,The Netherlands: Kluwer Academic Publishers, 2003.

[18] L. Perurena Cancio, «Sistema de herramientas para la construcción y administración de cursos multimedia» *Pixel-Bit. Universidad de la Habana,Cuba,* nº 21, Julio 2003.

[19] J. Sauer, «Integration Now, Integration Forever?» *E Media Magazine,* nº 17, pp. 16-23, 2004.

[20] D. Sussman, «Time Technology» *T+ D,* vol. 59, nº 8, pp. 53-56, 2005.

[21] T. Murray, «Authoring intelligent tutoring systems: An analysis of the state of the art» *International Journal of Artificial Intelligence in Education (IJAIED),* vol. 10, pp. 98-129, 1999.

[22] J. Rodríguez, Un sistema abierto para la creación de contenidos educativos digitales.Universidad de Alcalá, Guadalajara, 2004.

[23] IEEE Learning Technology Standards Comitee, «Learning Object Metadata. IEEE 1484.12.1.» 2002.

[24] Q. Changtao y N. Wolfgang, «Towards interoperability and reusability of learning resources: A SCORM-conformant courseware for computer science education» de *Proceedings of the 2nd IEEE International Conference on Advanced Learning Technologies (IEEE ICALT 2002)*, Kazan, Tatarstan, Russia, 2002.

[25] Centro de humanidades y computación de la Universidad de Victoria, Canadá, «Hot Potatoes» 2013. Visto online en: http://hotpot.uvic.ca.

[26] Departamento de Educación de la Generalitat de Cataluña, «JClic,» 2013. Visto online en: http://clic.xtec.cat/es/jclic/.

[27] Consejería de Educación y Ciencia de Castilla la Mancha, «Cuadernia» 2013. Visto online en: http://cuadernia.educa.jccm.es/.

[28] «Ardora» 2013. Visto online en: http://webardora.net/index_cas.htm.

[29] Universidad de Auckland, Universidad de Tecnología de Auckland y Politécnica de Tairawhiti, «Exe Learning» 2013. Visto online en: http://exelearning.net/.

[30] Laboratorio de Inteligencia Ambiental (Amilab-UAM) y el Laboratorio de Tecnologías de la Información en la Educación (LITE-URJC), «DEDOS» 2013. Visto online en: http://hada.ii.uam.es/dedos/.

[31] J. Roca Dorda, J. Roca González y M. E. Del Campo Adrián, «De las ayudas técnicas a la tecnología asistiva» de *Foro de Tecnología Educativa y Atención la Diversidad. Universidad Politécnica de Cartagena. Departamento Tecnología Eletrónica*, Murcia, España, 2004.

[32] E.F. LoPresti, A. Mihailidis y N. Kirsch, «Assistive technology for cognitive rehabilitation: State of the art» *Neuropsychological Rehabilitation,* vol. 14, pp. 5-39, 2004.

[33] D. Braddock, M.C. Rizzolo, M. Thompson y R. Bell, «Emerging technologies and cognitive disability» *Journal of Special Education Technology,* vol. 19, pp. 49-56, 2004.

[34] S. Carmien, «End user programming and context responsiveness in handheld prompting systems for persons with cognitive disabilities and caregivers» de *CHI'05 extended abstracts on Human factors in computing systems*, Portland, Oregon, USA, 2005.

[35] S. Carmien, «Assistive Technologies for persons with cognitive disabilities - Artifacts of distributed cognition» de *CHI 2006 workshop: Designing Technology for People with Cognitive Impairments*, Montreal, Canadá., 2006.

[36] A. Ferreras, J.-M. Belda, R. Barberà, R. Poveda, M. Urra, N. García, M. Tito y M. Valero, «PDA software aimed at improving workplace adaptation for people with cognitive disabilities» de *Computers Helping People with Special Needs*, Springer Berlin Heidelberg, 2010, pp. 13-20.

[37] D. Tost, S. Grau, M. Ferré, P. García, J. M. Tormos, A. Garcia y T. Roig, «PREVIRNEC: A cognitive telerehabilitation system based on Virtual Environments» de *Virtual Rehabilitation International Conference, 2009*, IEEE, 2009, pp. 87-93.

[38] L. S. Shafti, P. A. Haya, M. García-Herranz y X. Alamán, «Personal ambient intelligent reminder for people with cognitive disabilities» de *Ambient Assisted Living and Home Care*, Springer Berlin Heidelberg, 2012, pp. 383-390.

[39] J. Gómez, G. Montoro, P. Haya, M. García-Herranz y X. Alamán, «Distributed Schema-Based Middleware for Ambient Intelligence Enviroments» de *Ubiquitous Developments in Ambient Computing and Intelligence: Human-Centered Applications*, IGI Global, 2011, pp. 205-218.

[40] R. Engelmore y T. Morgan, Blackboard systems, Addison-Wesley Reading, 1988.

[41] D.E. Knuth., Art of Computer Programming, Volume 1: Fundamental Algorithms, Addison-Wesley, 1973.

[42] J. C. T. Vidal, Sistema adaptativo para la asistencia de personas con necesidades especiales en sus tareas de la vida diaria, Escuela Politécnica Superior, Universidad Autónoma de Madrid, 2013.

[43] I. Sommerville, Software engineering, Novena ed., Pearson (Addison-Wesley), 2011.

[44] «Librería Swing de Java» 2013. Visto online en: http://docs.oracle.com/javase/tutorial/uiswing/index.html.

[45] L. R. Baena, «Fundamentos de Interacción Persona-Ordenador. Diseño y prototipado» Universidad Pontificia de Salamanca. Facultad de Informática, Madrid, 2013.

[46] P. B. Alcázar, Aplicación de video-vigilancia multi-plataforma basada en tecnología JavaFX, Escuela Técnica Superior de Ingeniería de Telecomunicación. Universidad Politécnica de Cartagena, 2010.

[47] Oracle, «Características de JavaFX» 2013. Visto online en: http://www.oracle.com/us/technologies/java/fx/overview/index.html.

[48] L. PREMKUMAR y P. MOHAN, Beginning JavaFX, Apress, 2010.

[49] C. DEA, JavaFX 2.0: introduction by example, Apress, 2011.

[50] J. Weaver, W. Gao, S. Chin, D. Iverson y J. Vos, Pro JavaFX 2: A Definitive Guide to Rich Clients with Java Technology, Apress, 2012.

[51] Oracle JavaFX , Visto online en: http://docs.oracle.com/javafx/.

ANEXO A: MODELADO DE LAS ACTIVIDADES DEL ESQUEMA

```
# Action
class Action extends RootClass{
      capability hasToBeRepeated;
      capability isMultipleChoiceAction;
}

# Routine
class Routine extends Action;

# Task
class Task extends Action;

# Fragment
class Fragment extends RootClass{
      must properties{
            fragmentType;
            fragmentContent;
      }
}

capability hasToBeRepeated extends RootCap{
      must properties{
            nTimes;
      }
}

capability isMultipleChoiceAction extends RootCap{
      must properties{
            options;
      }
}

enum FragmentType = ("title", "description" , "video" , "audio"
, "image");
property fragmentType String = "title";
property fragmentContent String;
property userName String = "default";
property nTimes Numeric;
property options String = "default";
property forOption String;

relation hasAction extends RootRel{
      range Routine;
      domain Action;
```

```
	card 1:*;
	#inverse_of composesRoutine;
}

relation composesRoutine extends RootRel{
	range Action;
	domain Routine;
	card 1:*;
	# inverse_of hasTask;
}

relation hasFragment extends RootRel{
	range Task;
	domain Fragment;
	card 1:*;
	#inverse_of composesTask;
}

relation composesTask extends RootRel{
	range Fragment;
	domain Task;
	card 1:*;
	#inverse_of hasFragment;
}

relation nextAction{
	range Action;
	domain Action;
	card 1:*;
	must properties{
		userName = "default";
	}
	may properties{
		forOption;
	}
}

relation prevAction{
	range Action;
	domain Action;
	card 1:*;
	must properties{
		userName = "default";
	}
}

relation nextFragment{
	range Fragment;
	domain Fragment;
```

```
      card 1:*;
      must properties{
            userName = "default";
      }
}

relation prevFragment{
      range Fragment;
      domain Fragment;
      card 1:*;
      must properties{
            userName = "default";
      }
}
```

ANEXO B: EJEMPLOS DE CÓDIGOS DE FIGURAS

B.1. DESCRIPCIÓN DEL EJEMPLO DEL ÁRBOL DE LA FIGURA 3.2

```
## Actividad, Tarea y Fragmento nulos ##
entity Routine:RoutineNull@amilab;
entity Task:TaskNull@amilab;
entity Fragment:FragmentNull@amilab;

## Routine 1 ##
entity Routine:routine1@amilab{
      properties{
            descriptedName = "Actividad principal (routine1)";
      }
      relations{
            composesRoutine = Routine:RoutineNull@amilab;
            hasAction = Routine:routine2@amilab;
            hasAction = Routine:routine3@amilab;
            hasAction = Routine:routine4@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:TaskNull@amilab;
            }
}

## Routine 2 ##
entity Routine:routine2@amilab{
      properties{
            descriptedName = "Actividad 2 (routine2)";
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasAction = Task:task1@amilab;
            hasAction = Task:task2@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Routine:routine3@amilab;
      }
}

## Routine 3 ##
entity Routine:routine3@amilab{
      properties{
            descriptedName = "Actividad 3 (routine3)";
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasAction = Task:task3@amilab;
            hasAction = Task:task4@amilab;
```

```
            prevAction = Routine:routine2@amilab;
            nextAction = Routine:routine4@amilab;
      }
}

## Routine 4 ##
entity Routine:routine4@amilab{
      properties{
            descriptedName = "Actividad 4 (routine4)";
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasAction = Task:task5@amilab;
            hasAction = Task:task6@amilab;
            prevAction = Routine:routine3@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

## Task1 ##
entity Task:task1@amilab{
      relations{
            composesRoutine = Routine:routine2@amilab;
            hasFragment = Fragment:fragment1@amilab;
            hasFragment = Fragment:fragment2@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:task2@amilab;
      }
}

entity Fragment:fragment1@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 1";
      }
      relations{
            composesTask = Task:task1@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment2@amilab;
      }
}

entity Fragment:fragment2@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T1.jpg";
      }
      relations{
            composesTask = Task:task1@amilab;
```

```
            prevFragment = Fragment:fragment1@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Task2 ##
entity Task:task2@amilab{
      relations{
            composesRoutine = Routine:routine2@amilab;
            hasFragment = Fragment:fragment3@amilab;
            hasFragment = Fragment:fragment4@amilab;
            prevAction = Task:task1@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

entity Fragment:fragment3@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 2";
      }
      relations{
            composesTask = Task:task2@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment4@amilab;
      }
}

entity Fragment:fragment4@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T2.jpg";
      }
      relations{
            composesTask = Task:task2@amilab;
            prevFragment = Fragment:fragment3@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Task3 ##
entity Task:task3@amilab{
      relations{
      composesRoutine = Routine:routine3@amilab;
      hasFragment = Fragment:fragment5@amilab;
      hasFragment = Fragment:fragment6@amilab;
      prevAction = Task:TaskNull@amilab;
      nextAction = Task:task4@amilab;
      }
```

```
}

entity Fragment:fragment5@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 3";
      }
relations{
      composesTask = Task:task3@amilab;
      prevFragment = Fragment:FragmentNull@amilab;
      nextFragment = Fragment:fragment6@amilab;
      }
}

entity Fragment:fragment6@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T3.jpg";
      }
      relations{
            composesTask = Task:task3@amilab;
            prevFragment = Fragment:fragment5@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}
## Task4 ##
entity Task:task4@amilab{
      relations{
            composesRoutine = Routine:routine3@amilab;
            hasFragment = Fragment:fragment7@amilab;
            hasFragment = Fragment:fragment8@amilab;
            prevAction = Task:task3@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

entity Fragment:fragment7@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 4";
      }
      relations{
            composesTask = Task:task4@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment8@amilab;
      }
}
```

```
entity Fragment:fragment8@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T4.jpg";
      }

      relations{
      composesTask = Task:task4@amilab;
      prevFragment = Fragment:fragment7@amilab;
      nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Task5 ##
entity Task:task5@amilab{
      relations{
            composesRoutine = Routine:routine4@amilab;
            hasFragment = Fragment:fragment9@amilab;
            hasFragment = Fragment:fragment10@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:task6@amilab;
      }
}

entity Fragment:fragment9@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 5";
      }
      relations{
            composesTask = Task:task5@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment10@amilab;
      }
}

entity Fragment:fragment10@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T5.jpg";
      }
      relations{
            composesTask = Task:task5@amilab;
            prevFragment = Fragment:fragment9@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}
```

```
## Task6 ##
entity Task:task6@amilab{
      relations{
            composesRoutine = Routine:routine4@amilab;
            hasFragment = Fragment:fragment11@amilab;
            hasFragment = Fragment:fragment12@amilab;
            prevAction = Task:task5@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}
entity Fragment:fragment11@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 6";
      }
      relations{
            composesTask = Task:task6@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment12@amilab;
      }
}

entity Fragment:fragment12@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T6.jpg";
      }
      relations{
            composesTask = Task:task6@amilab;
            prevFragment = Fragment:fragment11@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}
```

B.2. DESCRIPCIÓN DEL EJEMPLO DEL ÁRBOL DE LA FIGURA 3.3

```
## Actividad, Tarea y Fragmento nulos ##
entity Routine:RoutineNull@amilab;
entity Task:TaskNull@amilab;
entity Fragment:FragmentNull@amilab;

## Routine 1 ##
entity Routine:routine1@amilab{
      properties{
            descriptedName = "Actividad principal (routine1)";
      }
      relations{
            composesRoutine = Routine:RoutineNull@amilab;
            hasAction = Routine:routine2@amilab;
            hasAction = Routine:routine3@amilab;
            hasAction = Routine:routine4@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:TaskNull@amilab;
            }
}

## Routine 2 ##
entity Routine:routine2@amilab{
      properties{
            descriptedName = "Actividad 2 (routine2)";
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasAction = Task:task1@amilab;
            hasAction = Task:task2@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Routine:routine3@amilab;
            nextAction = Routine:routine4@amilab{userName="UA";};
      }
}

## Routine 3 ##
entity Routine:routine3@amilab{
      properties{
            descriptedName = "Actividad 3 (routine3)";
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasAction = Task:task3@amilab;
            hasAction = Task:task4@amilab;
            prevAction = Routine:routine2@amilab;
            nextAction = Routine:routine4@amilab;
      }
```

```
}

## Routine 4 ##
entity Routine:routine4@amilab{
      properties{
            descriptedName = "Actividad 4 (routine4)";
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasAction = Task:task5@amilab;
            hasAction = Task:task6@amilab;
            prevAction = Routine:routine3@amilab;
            prevAction = Routine:routine2@amilab{userName="UA";};
            nextAction = Task:TaskNull@amilab;
      }
}

## Task1 ##
entity Task:task1@amilab{
      relations{
            composesRoutine = Routine:routine2@amilab;
            hasFragment = Fragment:fragment1@amilab;
            hasFragment = Fragment:fragment2@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:task2@amilab;
      }
}

entity Fragment:fragment1@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 1";
      }
      relations{
            composesTask = Task:task1@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment2@amilab;
      }
}

entity Fragment:fragment2@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T1.jpg";
      }
      relations{
            composesTask = Task:task1@amilab;
            prevFragment = Fragment:fragment1@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
```

```
	}
}

## Task2 ##
entity Task:task2@amilab{
	relations{
		composesRoutine = Routine:routine2@amilab;
		hasFragment = Fragment:fragment3@amilab;
		hasFragment = Fragment:fragment4@amilab;
		prevAction = Task:task1@amilab;
		nextAction = Task:TaskNull@amilab;
	}
}

entity Fragment:fragment3@amilab{
	properties{
		fragmentType = "text";
		fragmentContent = "Tarea número 2";
	}
	relations{
		composesTask = Task:task2@amilab;
		prevFragment = Fragment:FragmentNull@amilab;
		nextFragment = Fragment:fragment4@amilab;
	}
}

entity Fragment:fragment4@amilab{
	properties{
		fragmentType = "image";
		fragmentContent = "http://ruta.imagen.T2.jpg";
	}
	relations{
		composesTask = Task:task2@amilab;
		prevFragment = Fragment:fragment3@amilab;
		nextFragment = Fragment:FragmentNull@amilab;
	}
}

## Task3 ##
entity Task:task3@amilab{
	relations{
	composesRoutine = Routine:routine3@amilab;
	hasFragment = Fragment:fragment5@amilab;
	hasFragment = Fragment:fragment6@amilab;
	prevAction = Task:TaskNull@amilab;
	nextAction = Task:task4@amilab;
	}
}
```

```
entity Fragment:fragment5@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 3";
      }
relations{
      composesTask = Task:task3@amilab;
      prevFragment = Fragment:FragmentNull@amilab;
      nextFragment = Fragment:fragment6@amilab;
      }
}

entity Fragment:fragment6@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T3.jpg";
      }
      relations{
            composesTask = Task:task3@amilab;
            prevFragment = Fragment:fragment5@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Task4 ##
entity Task:task4@amilab{
      relations{
            composesRoutine = Routine:routine3@amilab;
            hasFragment = Fragment:fragment7@amilab;
            hasFragment = Fragment:fragment8@amilab;
            prevAction = Task:task3@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

entity Fragment:fragment7@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 4";
      }
      relations{
            composesTask = Task:task4@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment8@amilab;
      }
}
```

```
entity Fragment:fragment8@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T4.jpg";
      }

      relations{
      composesTask = Task:task4@amilab;
      prevFragment = Fragment:fragment7@amilab;
      nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Task5 ##
entity Task:task5@amilab{
      relations{
            composesRoutine = Routine:routine4@amilab;
            hasFragment = Fragment:fragment9@amilab;
            hasFragment = Fragment:fragment10@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:task6@amilab;
            nextAction = Task:TaskNull@amilab{userName="UA";}
      }
}

entity Fragment:fragment9@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 5";
      }
      relations{
            composesTask = Task:task5@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment10@amilab;
      }
}

entity Fragment:fragment10@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T5.jpg";
      }
      relations{
            composesTask = Task:task5@amilab;
            prevFragment = Fragment:fragment9@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}
```

```
## Task6 ##
entity Task:task6@amilab{
      relations{
            composesRoutine = Routine:routine4@amilab;
            hasFragment = Fragment:fragment11@amilab;
            hasFragment = Fragment:fragment12@amilab;
            prevAction = Task:task5@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

entity Fragment:fragment11@amilab{
      properties{
            fragmentType = "text";
            fragmentContent = "Tarea número 6";
      }
      relations{
            composesTask = Task:task6@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment12@amilab;
      }
}

entity Fragment:fragment12@amilab{
      properties{
            fragmentType = "image";
            fragmentContent = "http://ruta.imagen.T6.jpg";
      }
      relations{
            composesTask = Task:task6@amilab;
            prevFragment = Fragment:fragment11@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}
```

B.3. DESCRIPCIÓN DEL EJEMPLO DE LA SECCIÓN 3.8

```
## Actividad "Hacer Fotocopias" ##
entity Routine:routine0@amilab{
      properties{
            descriptedName = "Hacer Fotocopias";
      }
      relations{
            composesRoutine = Routine:RoutineNull@amilab;
            hasAction = Task:task0@amilab;
            hasAction = Task:task1@amilab;
            hasAction = Task:task2@amilab;
            hasAction = Routine:routine1@amilab;
            hasAction = Routine:routine2@amilab;
            hasAction = Task:task4@amilab;
            hasAction = Task:task5@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

## Primera opción: "A una cara" ##
entity Routine:routine1@amilab{
      properties{
            descriptedName = "A una cara";
      }
      relations{
            composesRoutine = Routine:routine0@amilab;
            hasAction = Task:task3@amilab;
            prevAction = Task:task2@amilab;
            nextAction = Task:task4@amilab;
      }
}

## Segunda opción: "A dos caras" ##
entity Routine:routine2@amilab{
      properties{
            descriptedName = "A dos caras";
      }
      relations{
            composesRoutine = Routine:routine0@amilab;
            prevAction = Task:task2@amilab;
            nextAction = Task:task4@amilab;
      }
}

## Tarea para encender la fotocopiadora ##
entity Task:task0@amilab{
      relations{
```

```
            composesRoutine = Routine:routine0@amilab;
            hasFragment = Fragment:fragment0@amilab;
            hasFragment = Fragment:fragment1@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:task1@amilab;
      }
}

entity Fragment:fragment0@amilab{
      properties{
            fragmentType = "text";
            fragmentContent="<H1>Pulsar    el    bot&oacute;n    de
            encendido</H1>";
      }
      relations{
            composesTask = Task:task0@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment1@amilab;
      }
}

entity Fragment:fragment1@amilab{
      properties{
            fragmentType = "image";
            fragmentContent="encender.png";
      }
      relations{
            composesTask = Task:task0@amilab;
            prevFragment = Fragment:fragment0@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Poner las hojas en la fotocopiadora ##
entity Task:task1@amilab{
      relations{
            composesRoutine = Routine:routine0@amilab;
            hasFragment = Fragment:fragment2@amilab;
            hasFragment = Fragment:fragment3@amilab;
            prevAction = Task:task0@amilab;
            nextAction = Task:task2@amilab;
      }
}

entity Fragment:fragment2@amilab{
      properties{
            fragmentType = "text";
            fragmentContent="<H1>Poner    las    hojas    en    la
            fotocopiadora</H1>";
```

```
	}
	relations{
		composesTask = Task:task1@amilab;
		prevFragment = Fragment:FragmentNull@amilab;
		nextFragment = Fragment:fragment3@amilab;
	}
}

entity Fragment:fragment3@amilab{
	properties{
		fragmentType = "image";
		fragmentContent="poneroriginales.png";
	}
	relations{
		composesTask = Task:task1@amilab;
		prevFragment = Fragment:fragment2@amilab;
		nextFragment = Fragment:FragmentNull@amilab;
	}
}

## Seleccionar el modo de copia ##
entity Task:task2@amilab{
	capabilities{
		isMultipleChoiceAction{
			options = ("A una cara","A dos caras");
		}
	}
	relations{
		composesRoutine = Routine:routine0@amilab;
		hasFragment = Fragment:fragment4@amilab;
		hasFragment = Fragment:fragment5@amilab;
		prevAction = Task:task1@amilab;
		nextAction = Task:task4@amilab;
		nextAction = Routine: routine1@amilab {forOption = "A
		una cara";}
		nextAction = Routine: routine2@amilab {forOption = "A
		dos caras";}
	}
}

entity Fragment:fragment4@amilab{
	properties{
		fragmentType = "text";
		fragmentContent="<H1>Seleccionar el modo de
		copia</H1>";
	}
	relations{
		composesTask = Task:task2@amilab;
		prevFragment = Fragment:FragmentNull@amilab;
```

```
            nextFragment = Fragment:fragment5@amilab;
      }
}

entity Fragment:fragment5@amilab{
      properties{
            fragmentType = "image";
            fragmentContent="escogermodo.png";
      }
      relations{
            composesTask = Task:task2@amilab;
            prevFragment = Fragment:fragment4@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}

## Realizar las copias. Número a escoger por el usuario ##
entity Task:task3@amilab{
      capabilities{
            hasToBeRepeated{
                  nTimes=-1;
            }
      }
      relations{
            composesRoutine = Routine:routine1@amilab;
            hasFragment = Fragment:fragment6@amilab;
            hasFragment = Fragment:fragment7@amilab;
            prevAction = Task:TaskNull@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

entity Fragment:fragment6@amilab{
      properties{
            fragmentType = "text";
            fragmentContent="<H1>Pulsar    el    bot&oacute;n    de
            copia</H1>";
      }
      relations{
            composesTask = Task:task3@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment7@amilab;
      }
}

entity Fragment:fragment7@amilab{
      properties{
            fragmentType = "image";
            fragmentContent="hacercopia.png";
```

```
	}
	relations{
		composesTask = Task:task3@amilab;
		prevFragment = Fragment:fragment6@amilab;
		nextFragment = Fragment:FragmentNull@amilab;
	}
}

## Recoger el original y las copias de la fotocopiadora ##
entity Task:task4@amilab{
	relations{
		composesRoutine = Routine:routine0@amilab;
		hasFragment = Fragment:fragment8@amilab;
		hasFragment = Fragment:fragment9@amilab;
		prevAction = Task:task2@amilab;
		nextAction = Task:task5@amilab;
	}
}

entity Fragment:fragment8@amilab{
	properties{
		fragmentType = "text";
		fragmentContent="<H1>Recoger las copias y las
		hojas</H1>";
	}
	relations{
		composesTask = Task:task4@amilab;
		prevFragment = Fragment:FragmentNull@amilab;
		nextFragment = Fragment:fragment9@amilab;
	}
}

entity Fragment:fragment9@amilab{
	properties{
		fragmentType = "image";
		fragmentContent="recoger.png";
	}
	relations{
		composesTask = Task:task4@amilab;
		prevFragment = Fragment:fragment8@amilab;
		nextFragment = Fragment:FragmentNull@amilab;
	}
}

## Apagar la fotocopiadora ##
entity Task:task5@amilab{
	relations{
		composesRoutine = Routine:routine0@amilab;
		hasFragment = Fragment:fragment10@amilab;
```

```
            hasFragment = Fragment:fragment11@amilab;
            prevAction = Task:task4@amilab;
            nextAction = Task:TaskNull@amilab;
      }
}

entity Fragment:fragment10@amilab{
      properties{
            fragmentType = "text";
            fragmentContent="<H1>Pulsar    el    bot&oacute;n    de
            apagar</H1>";
      }
      relations{
            composesTask = Task:task5@amilab;
            prevFragment = Fragment:FragmentNull@amilab;
            nextFragment = Fragment:fragment11@amilab;
      }
}

entity Fragment:fragment11@amilab{
      properties{
            fragmentType = "image";
            fragmentContent="apagar.png";
      }
      relations{
            composesTask = Task:task5@amilab;
            prevFragment = Fragment:fragment10@amilab;
            nextFragment = Fragment:FragmentNull@amilab;
      }
}
```

ANEXO C: CREACIÓN DE UNA INTERFAZ GRÁFICA CON JAVAFX

C.1. INTRODUCCIÓN

En la manipulación de objetos gráficos es donde se encuentra el punto fuerte de JavaFX. Como se describe en la figura incluida a continuación, en el nivel más alto para el manejo de los objetos gráficos se encuentra el *Stage,* el *Scene* contendrá los objetos mostrados en el escenario y la clase *Node* nos aportará toda una serie de herramientas que permitirán manejar los objetos gráficos de forma consistente [46] [47].

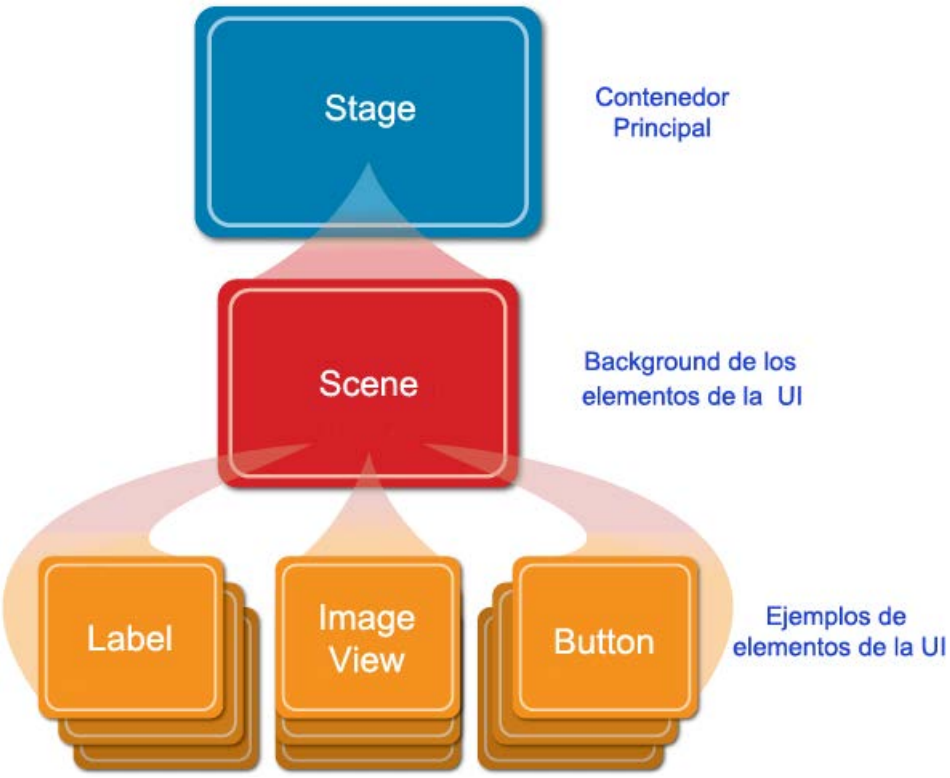


Figura C.1: Estructura de los elementos de JavaFX.

Stage

Todo programa JavaFX que pretenda manejar material gráfico debe incluir un *Stage* en su nivel más alto, el *Stage* contendrá un objeto *Scene*, que a su vez contendrá una secuencia de *Nodes*. Por defecto el *Stage* tiene estilo StageStyle.DECORATED, que será visto de distinta forma según el ambiente (Windows 7, Windows Vista, Windows XP, Mac OS). StageStyle.UNDECORATED eliminará la decoración de la ventana.

Scene

La clase *Scene* es el *root* (la raíz) de todo el contenido en un escenario gráfico. El color de fondo de un *Scene* está especificado por la propiedad *fill* (por defecto Color.WHITE). La secuencia de nodos contenida por el *Scene* será mostrada en el escenario gráfico.

Node

La clase *Node* es la clase base para todos los objetos en el escenario gráfico. Se pueden añadir objetos *Node* (y subclases de *Node*) al escenario, especificar sus propiedades y aplicarles transformaciones. Esta clase tiene muchas propiedades que permiten personalizar su aspecto y comportamiento, como la visibilidad, los manejadores de eventos, opacidad, rotación, visibilidad y otras muchas propiedades. Además existen varias subclases de tipo *Node* (existen tres tipos: subclases predefinidas como *Shape* o *ImageView* con una funcionalidad predeterminada, la subclase *Group* aportará contenedores para más elementos tipo *Node*, y por último *CustomNode* nos permitirá personalizar nuestro propio tipo de *Node* según nuestras necesidades) para poder situar mejor la información en la interfaz. En el ejemplo de la figura anterior, los elementos de la UI que se muestran (elementos de color naranja) pertenecen a esta clase.

Es curioso ver como algunos de los objetos que se manejan en este lenguaje se llaman *Stage* (escenario) o *Scene* (escena). Esto es así porque los creadores de la API lo modelaron de manera similar a una obra de teatro en la que los actores (*Nodes*) llevan a cabo la función delante de la audiencia. Con esta misma analogía, se entiende que aunque haya muchas escenas en las que trabajan los actores, todas ellas se realizan sobre un mismo escenario (relación de uno a muchos). Para Java Swing el símil de un *Stage* sería un *JFrame* o un *JDialog*. En estos contenedores había en su interior varios paneles en los cuales se insertaba a su vez los objetos que se quisieran mostrar en la interfaz. Estos paneles, en el caso de JavaFX corresponderían con las *Scenes* las cuales son capaces de albergar a varios *Nodes* (nodos) como se ha comentado previamente (ejemplos de nodos son: *button*, *label*, *imageView*, etc) [49].

C.2. EJEMPLO DE CÓDIGO Y RESULTADO

El código generado para este ejemplo es:

```
package ejemplojavafx;

import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.control.Label;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.VBox;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import javafx.stage.Stage;

/*****************************************
 * Programa de ejemplo para conocer JavaFX *
 * Realizado por: Pablo Jesús Bueno Soto   *
 * Fecha: 15/07/2013                       *
 *****************************************/

public class EjemploJavaFX extends Application {

 @Override
 public void start(Stage primaryStage) {

  Button btn = new Button();
  btn.setText("Pulse este botón para cerrar la ventana");
  btn.setOnAction(new EventHandler<ActionEvent>() {

  @Override
   public void handle(ActionEvent event) {
     System.out.println("Programa terminado");
     System.exit(0);
   }
  });

  Label label = new Label("¡¡¡Bienvenido a JavaFX!!!");
  label.setFont(Font.font("Arial", FontWeight.BOLD, 32));

  Image image=new Image(getClass().getResourceAsStream("imagenFX.jpg"));
  ImageView imageView= new ImageView(image);
```

```
 VBox root = new VBox(10);
 root.setAlignment(Pos.CENTER);
 root.getChildren().addAll(label,imageView,btn);

 Scene scene = new Scene(root, 700, 500, Color.LIGHTCYAN);

 primaryStage.setTitle("Interfaz de ejemplo");
 primaryStage.setScene(scene);
 primaryStage.show();
}

public static void main(String[] args) {
 launch(args);
}
}
```

Figura C.2: Ejemplo de código en JavaFX.

En primer lugar, en el código aparece el paquete al que pertenece esta clase y también todas las librerías que son necesarias para trabajar con los diferentes objetos.

Para JavaFX, la clase principal tiene que descender de *javafx.application.Application,* la cual otorga funciones necesarias a las aplicaciones como son el lanzamiento y la parada en tiempo de ejecución.

El método *main()* que aparece en este código no es obligatorio que figure, siempre y cuando los archivos JAR generados hayan sido creados usando la herramienta *JavaFX Packager*, la cual contiene *JavaFX Launcher*. En ese caso el método *main()* es ignorado por el programa pues es este mismo *launcher* el que lanza las aplicaciones JavaFX. Sin embargo, se ha preferido mantener para que, en el caso de que no se disponga de las herramientas que se acaban de describir, el método *main()* de Java lance la aplicación de JavaFX pasando los argumentos por línea de comandos al método *Application.launch()*.

El método *start()* es el principal para todas las aplicaciones en este lenguaje (similar al método *main()* en Java). Éste es lanzado cuando la aplicación se encuentra preparada. Es en ese momento cuando se dispone de una nueva *javafx.stage.Stage* (escenario) para que los programadores desarrollen sobre ella la información que quieran mostrar (escenas). En este caso, se cambia el título que aparece en la ventana de este *Stage* y se le añade una escena. Finalmente se indica que el escenario sea visible al usuario.

Sobre la escena creada, se le añade un nodo llamado root, se define el tamaño de ésta y se cambia el color del fondo.

El nodo que se incorpora a la escena es de la clase *javafx.scene.layout.VBox*. Este *layout* hace que los objetos introducidos en él se coloquen de forma vertical con una separación fija (10) y con una alineación justificada. Los objetos que contiene son:

- *Label*: una etiqueta la cual sirve para mostrar un texto.
- *ImageView*: un objeto el cual permite insertar en la UI una imagen.
- *Button*: un botón, el cual cuando es pulsado, la aplicación finaliza. Para realizar esta acción se añade un oyente a este objeto.

El resultado obtenido de la ejecución del ejemplo es:

Figura C.3: Interfaz gráfica realizada en JavaFX con el código de la Figura C.2.

ANEXO D: PRESUPUESTO

1) Ejecución Material

• Compra de ordenador personal (*software* incluido)	2000 €
• Alquiler de una impresora láser	250 €
• Material de oficina	150 €
• Total de ejecución material	2400 €

2) Gastos generales

• 16% sobre Ejecución Material	384 €

3) Beneficio Industrial

• 6% sobre Ejecución Material	144 €

4) Honorarios Proyecto

• 800 horas a 15 €/hora	12000 €

5) Material fungible

• Gastos de impresión	250 €
• Encuadernación	100 €

6) Subtotal del presupuesto

• Subtotal Presupuesto	15278 €

7) I.V.A. aplicable

• 21% Subtotal Presupuesto	3208 €

8) Total presupuesto

• Total Presupuesto	18486 €

Madrid, Septiembre 2013
El Ingeniero Jefe de Proyecto

Fdo.: Pablo Jesús Bueno Soto
Ingeniero Superior de Telecomunicación

ANEXO E: PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de una *herramienta de autor para la definición de guías de interacción adaptativas para personas con discapacidad cognitiva*.

En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.
2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.
3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.
4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.
5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.
7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.
8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.
9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.
10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.
11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que

tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.
13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con las tarifas y honorarios vigentes.
14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.
15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.
16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.
17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.
18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.
19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.
21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.
22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.
23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.
3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.
4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.
5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.
6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.
7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.
8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinar toda responsabilidad que se derive de la aplicación o influencia de la misma.
9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.
10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.
11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.