UNIVERSIDAD DE CASTILLA-LA MANCHA
ESCUELA SUPERIOR DE INFORMÁTICA

INGENIERÍA
EN INFORMÁTICA

PROYECTO FIN DE CARRERA

FreeStation. Plataforma para el desarrollo de sistemas de distribución de software libre en puntos de información

Ángel Guzmán Maeso

Septiembre, 2012

UNIVERSIDAD DE CASTILLA-LA MANCHA

ESCUELA SUPERIOR DE INFORMÁTICA

Departamento de Tecnologías y Sistemas de Información

PROYECTO FIN DE CARRERA

FreeStation. Plataforma para el desarrollo de sistemas de distribución de software libre en puntos de información

Autor: Ángel Guzmán Maeso
Director: Carlos González Morcillo

Septiembre, 2012

TRIBUNAL:

Presidente:

Vocal 1:

Vocal 2:

Secretario:

FECHA DE DEFENSA:

CALIFICACIÓN:

PRESIDENTE	VOCAL 1	VOCAL 2	SECRETARIO
Fdo.:	Fdo.:	Fdo.:	Fdo.:

Ángel Guzmán Maeso

Ciudad Real – España

E-mail: angel.guzman@alu.uclm.es
Web site: `http://shakaran.net/blog/freestation/`

Agradecimientos

Al pueblo de Manzanares que me vio crecer día a día en su sol manchego. A aquellos profesores que pusieron los conocimientos en mi mente y me enseñaron que: «la mínima posibilidad de soñar una idea, podía con certeza, convertirse en una realidad». A cada granito de arena aportado por aquellos que consiguieron y consiguen hacer cada día una montaña mayor de aportes al mundo del software libre y que sin duda no hubiera podido formarme sin sus grandes contribuciones.

A mis padres,
por recordarme a diario que
«no hay nada imposible para los corazones valientes»

Resumen

La librenería o FreeStation (FS) es un software para centros o Puntos de Acceso para Distribución (PAD) de información de software libre orientado a centros de enseñanza y universidades.

La posibilidad de una herramienta genérica para la distribución de software, permite un gran abanico de posibilidades para extender de forma más sencilla el uso del software libre en diferentes organizaciones e instituciones.

En la actualidad la mayoría de sistemas similares son propietarios y suelen caer en la obsolescencia por la falta de personalizaciones propias a determinados problemas.

La erradicación de estos problemas proponiendo una herramienta robusta y configurable, cubre la futura demanda desde pequeñas a grandes empresas o organismos.

Los repositorios de software modularizables configurados bajo la preferencia del usuario, permiten a traves de un Point Of Interest (POI) un rápido acceso sin complicaciones.

Abstract

THE librenería or FreeStation (FS) is a software for centers or Access Points for Distribution (APD) to free software information oriented to teaching centers and universities.

The possibility of a generic tool for software distribution, allows a wide range of possibilities to expand more easily the use of free software in different organizations and institutions.

Currently most systems are proprietary and often similar fall into obsolescence by the lack of customization specific to certain problems.

The eradication of these problems by proposing a robust and configurable, meet future demand from small to large companies or agencies.

Modularized software repositories configured under the preference the user, let through a Point Of Interest (POI) quick access without complications.

Índice general

Índice de cuadros

Índice de figuras

Listados de códigos fuente

Listado de acrónimos

- APD - Access Points for Distribution XI
- BSD - Berkley Software Distribution 22
- CGI - Common Gateway Interface 43
- DOM - Document Object Model 31
- DTD - Document Type Definition 33
- FDL - Free Documentation License 48
- FS - FreeStation IX
- FTP - File Transfer Protocol 1
- GNU - GNU is Not Unix 3
- GUI - Graphical User Interface 56
- ICE - Internet Communications Engine 57
- IPO - Interacción Persona Ordenador 30
- IEEE - Institute of Electrical and Electronics Engineers 33
- GDK - GIMP Drawing Kit 121
- GI - Gnome Instrospection 119
- GOBJECT - GLib Object System 75
- GPL - Gnu Public Licence 22
- GTK+ - The GIMP Toolkit 49
- LGPL - Lesser Gnu Public Licence 22
- PAD - Puntos de Acceso para Distribución IX
- PHP - PHP Hypertext Pre-processor 43
- POC - Proof of Concept 119
- POI - Point Of Interest 61
- P2P - Peer to Peer 1
- MIT - Massachusetts Institute of Technology 22
- MVC - Modelo Vista Controlador 91
- NUI - Natural User Interface 26
- QoS - Calidad de Servicio 2

- RDBMS - Relational DataBase Management System 47
- RITE - Rapid Iterative Testing and Evaluation 30
- SHA - Secure Hash Algorithm 97
- SPOF - Single Point Of Failure 63
- UDP - User Datagram Protocol 50
- USB - Universal Serial Bus 66
- TCP - Transmission Control Protocol 50
- UDI - The User-Derived Interface 29
- WIMP - Windows Icons Menus Pointer 26
- W3C - World Wide Web Consortium 33
- XML - Extensible Markup Language 42

Capítulo 1

1 INTRODUCCIÓN

La distribución de la información ha sido un problema a resolver desde antaño. Desde los primeros mensajes emitidos vía oral, pasando por escritos hasta la forma más novedosa que ha logrado el ser humano, como es internet, la complejidad relativa a las necesidades de comunicación ha ido en aumento.

Con los nuevos tiempos, se ha hecho necesario transmitir esa información de forma general y distribuida. La mayor parte de esta información es procesada con ayuda de aplicaciones, cuya distribución y configuración debe ser abordada previamente de forma específica a nivel hardware y software con configuraciones generales y a cierto nivel de detalle con personalizaciones.

La técnica inicial de distribución de software fue en principio únicamente física. Tiendas especializadas en puntos muy concretos de un país, hacían posible comprar una aplicación y usarla. Con el avance de la tecnología, se ha conseguido poder descargar desde páginas especializadas de forma directa mediante el protocolo FTP hasta redes P2P empleando Bittorrent.

Normalmente estos modelos son generales y no se adaptan a las necesidades específicas de personalización de ciertas organizaciones. Es necesario por tanto, un modelo, en el que una organización o empresa pueda definir unos parámetros base generales y otros parámetros específicos para distribuir su software y configurarlo a medida de sus usuarios finales.

En esta solución, puede verse una gran utilidad para configuraciones de POIs donde un usuario, pueda interactuar y conseguir la distribución de software necesaria en un entorno determinado.

1.1. QUÉ ES LA DISTRIBUCIÓN DE SOFTWARE

La distribución del software vista como un servicio distribuido, necesita ser rápida, escalable y robusta. Según [LiB11] se trata de realizar un modelo que se ajuste al software y a ningún otro requisito adicional, para que la distribución sea tan sencilla y limpia que no sea posible cometer fallos derivados. Este tipo de modelo general puede utilizarse en multitud de sectores, como en el financiero, militares, médico, logístico, etc.

Fundamentalmente se trata de mejorar la forma de integrar y enviar datos y aplicaciones, evitando cruzar los límites del costo del desarrollo de vida de una aplicación, incluyendo el desarrollo, pruebas, integración, mantenimiento y actualizaciones.

Los mensajes y aplicaciones enviadas por P2P tienen una baja latencia y un alto rendimiento, junto una alta disponibilidad, aunque pueden tener asociada una pérdida de confiabilidad en los mensajes y suelen requerir un gran costo de computación en servidores. De este modo son una alternativa muy utilizada, cuando ciertos problemas derivados de la pérdida de datos o ruido no son críticos en el dominio de explotación.

Eliminando la complejidad de la conectividad punto a punto, se puede conseguir un sistema escalable y robusto.

Para la distribución de software un aspecto clave es la interoperabilidad y la Calidad de Servicio (QoS). Garantizar la transmisión de cierta cantidad de información en un tiempo determinado (*throughput*) es una condición relevante para un servicio de calidad. Este requisito resulta especialmente importante en aplicaciones que requieren transmisión de vídeo o voz.

Un modelo dirigido por eventos o en tiempo real, permiten definir una arquitectura orientada a servicios eficiente.

1.2. INTRODUCCIÓN HISTÓRICA

La distribución de software fue un gran reto no hace muchos años. En sus inicios, un usuario necesitaba ser probablemente un experto en el dominio[ReJ12] para poder obtener una copia funcional del software requerido.

Por ejemplo, las primeras distribuciones GNU/Linux requerían tener conocimientos avanzados en gestión de Sistemas, conociendo las bibliotecas y ejecutables que eran necesarios para construir el sistema y conseguir finalmente un sistema operativo funcional.

Los detalles específicos de configuración hacían abandonar a muchas personas en esta tarea. Aunque el crecimiento de la comunidad y el esfuerzo compartido llevaron a construir herramientas para construir sistemas GNU, más conocidas como las Autotools. Con una colección de herramientas diseñadas para hacer la vida más fácil en la distribución de aplicaciones, los costes de tiempo se redujeron y se consiguió realizar paquetes de aplicaciones más portables a otros sistemas compatibles con Unix.

Por tanto, se facilito al usuario final la posibilidad de recibir su propio software empaquetado o realizar modificaciones sobre el mismo, habilitando algunos parámetros sencillos en sus configuraciones.

La distribución de software en Berkeley con el sistema operativo BSD, debe sus siglas a Berkeley Software Distribution. En las décadas de los setenta y ochenta, la Universidad Berkeley se propuso compartir y adaptar su software desde que la compañía ATT retiró el permiso para usos comerciales. Debido a la demanda de un gran público, el proceso de distribución de software fue una gran tarea a considerar. Los usuarios demandaban personalizaciones y varias ramificaciones del mismo proyecto surgieron como: OpenBSD, FreeBSD, SunOS, etc.

Estos hitos llevaron a hacer software escalable para un uso amplio. La distribución del software empezó a convertirse en un conjunto de actividades interrelacionadas que permitían producir o consumir sistemas necesarios para producción.

La distribución por tanto incluye un proceso de obtención de software, instalación, activación/desactivación, adaptación o actualización, integración, seguimiento de cambios y desinstalación.

1.3. PROBLEMÁTICA

Los problemas derivados de la distribución en POI, conllevan grandes requerimientos de ancho de banda entre servidor/es y posibles clientes[New10]. Si la red en la que se intenta distribuir cuenta con un número elevado de usuarios, o se hace un uso intenso de los recursos puede conllevar un mal funcionamiento del sistema[Hel00].

Los costes de los equipos suelen ser elevados (normalmente los servidores) al igual que sus interconexiones.

Como problema derivado, el sistema necesita disponer de una determinada homogeneización de componentes tanto hardware como software, para que habilite una forma rápida y segura de hacer cambios de manera dinámica y evitar problemas con diferentes versiones ejecutadas.

Las actualizaciones a menudo, requieren de personal cualificado con un buen criterio estratégico que administren las prioridades necesarias para que un software esté plenamente funcional con las últimas novedades sin comprometer al sistema.

Las dificultades más comunes son la parametrización de atributos más adecuada para el trabajo específico del usuario. El desarrollo para múltiples plataformas, la interconexión robusta entre clientes y servidores y la disponibilidad online del servicio y calidad del mismo.

Solución general

Las soluciones que se pueden aportar pasan por la difusión de software empleando el método multicast [Wil99] en el que se optimiza el ancho de banda requerido del envío de la información.

Con esta solución, se optimiza la red con múltiples destinos, que reciben los datos de forma simultánea con la estrategia más eficiente.

Además se contempla implícitamente que el número de incidencias recibidas en el centro de soporte anterior debe reducirse al distribuir de forma más eficiente software específico vitales para la organización.

Para mitigar estos problemas, se propone el desarrollo de este Proyecto de Fin de Carrera, FreeStation, cuyo objetivo general puede ser descrito como la definición de una Plataforma para el desarrollo de sistemas de distribución de software libre en puntos de información. La arquitectura modular y extensible de *FreeStation* permiten su cómoda adaptación a las necesidades específicas de cada organización, facilitando al usuario final la obtención de software (o datos) específicos a cada dominio de explotación específico.

El presente de Proyecto de Fin de Carrera debe basarse en una arquitectura modular y extensible para añadir características nuevas o mejorar las ya presentes.

Se entiende que poco a poco, de forma progresiva y con la ayuda del resultado de estos primeros proyectos, se procederá a pensar en esta plataforma como medio adicional de distribución de aplicaciones diseñadas a medida o de forma generalizada, cuyo coste de mantenimiento y atención es muy inferior cuando se administra de modo distribuido.

1.4. ESTRUCTURA DEL DOCUMENTO

Este documento se ha estructurado según las indicaciones de la normativa de proyectos de fin de carrera de la Escuela Superior de Informática de la Universidad de Castilla-La Mancha, empleando los siguientes capítulos:

Capítulo 1: Introducción

En este capítulo se ha realizado una breve introducción del ámbito en el que se encuadra FreeStation.

Capítulo 2. Objetivos

En este capítulo se desglosa y se describen la lista de objetivos y subobjetivos planteados para este Proyecto de Fin de Carrera.

Capítulo 3. Antecedentes

En este capítulo se hace un repaso a los conocimientos y áreas que ha sido necesario estudiar para el desarrollo de FreeStation, junto a los principales sistemas existentes relacionados.

Capítulo 4. Método de trabajo

En este capítulo se explica y se justifica la metodología escogida para el desarrollo de este sistema. Además se describen los recursos empleados, tanto hardware como software.

Capítulo 5. Arquitectura

En este capítulo se describen los aspectos más importantes relativos al diseño e implementación del sistema, detallando los problemas surgidos y las soluciones aportadas. El capítulo se centra en la clasificación de los módulos y submódulos que componen FreeStation, realizando una descripción "*top-down*" desde un enfoque funcional hasta un nivel de detalle técnico y específico de cada uno de ellos.

Capítulo 6. Evolución y costes

En este capítulo se describe la evolución del sistema durante su período de desarrollo, detallando las etapas e iteraciones realizadas.

Igualmente se aporta información relacionada con el coste económico, el análisis de rendimiento (*profiling*), y una serie de comparativas con aplicaciones homólogas que no utilizan FreeStation, además de una encuesta realizada sobre su uso.

Capítulo 7. Conclusiones y propuestas

En este capítulo se hace un resumen a modo de conclusión del desarrollo y las metas alcanzadas. También se enumera una serie de líneas de trabajo futuro planteadas para continuar su desarrollo, y una conclusión personal.

Capítulo 2

2 OBJETIVOS

2.1. OBJETIVO PRINCIPAL

El objetivo principal de este proyecto fin de carrera es crear una plataforma con la que construir, desarrollar y desplegar sistemas de distribución de software libre de una organización adaptados a sus necesidades.

Los usuarios, a través de terminales de acceso con interfaces de manipulación directa, podrán solicitar entre un catálogo de software disponible y realizar operaciones para cumplir sus los objetivos de forma rápida y segura.

Para la consecución de este objetivo funcional general, se han identificado una serie de requisitos no funcionales que se resumen a continuación:

- Identificar y recoger los requisitos de la aplicación para dar comienzo a la fase de desarrollo.
- Realizar un estudio del panorama actual sobre distribución de software en puntos de interés.
- Analizar las diferentes tecnologías para la creación de una aplicación web: Entornos de desarrollo, bases de datos, lenguajes de programación, frameworks, bibliotecas y navegadores.
- Creación de la aplicación web con los requisitos anteriormente identificados y siguiendo la metodología de la XP.

2.2. SUBOBJETIVOS

El objetivo general descrito anteriormente será alcanzado en base al cumplimiento de una serie de subobjetivos funcionales y no funcionales que se describen a continuación:

- **Adaptabilidad**. El sistema debe permitir trabajar con diferentes configuraciones con respecto al número de nodos de la organización, así como de los requisitos particulares de cada uno de ellos. La diversidad de requisitos deberá estar presente desde el inicio del proyecto, facilitando la adaptación y ampliación a nuevas necesidades.
- **Robustez**. El sistema debe ser muy robusto, y deberá contar con mecanismos para el control y recuperación de errores, para garantizar su correcta puesta en explotación en un entorno real[Dea98]. La plataforma facilitará el registro y la medición de datos de cada actividad desarrollada en el sistema, para prevenir inconsistencia y avisar de posibles fallos producidos.
- **Heterogeneidad**. La plataforma se encargará de ofrecer una capa de homogeneización del software que abstraiga de las propiedades específicas del hardware utilizado en cada punto de información. El diseño modular del sistema permitirá su utilización de forma legible y manejable, facilitando una correcta parametrización y reutilización de componentes. Un sistema extensible, además permitirá una modificación fácil de su comportamiento sin tener que realizar cambios en todo el sistema.
- **Distribuido**. El sistema se diseñará para minimizar el impacto en el ancho de banda en red (evitando la transmisión de información redundante), permitiendo el despliegue de varios nodos, eliminando puntos únicos de fallo (SPOF) mediante el uso de componentes y objetos distribuidos.
- **Diseño NUI**. La plataforma estará orientada a la puesta en explotación de sistemas basados en el paradigma NUI, explotando de un modo eficiente las posibles capacidades de subsistemas de vídeo, audio y otros contenidos interactivos. El sistema definirá un conjunto de widgets orientados a la manipulación directa.
- **Actualizable**. El sistema debe permitir la fácil actualización en todos los nodos del a organización de un modo automático. El impacto en tiempo para el usuario debe ser mínimo, de modo que las actualizaciones o cambios en nuevas versiones del software deben realizarse de modo desatendido.
- **Libre**. La plataforma estará basada en estándares libres. En el desarrollo del proyecto se utilizarán alternativas basadas en software libre y uso de estándares consensuados.

Capítulo 3

3 ANTECEDENTES Y ESTADO DEL ARTE

En este capítulo se estudiarán las áreas que han sido necesarias para el desarrollo del presente Proyecto Fin de Carrera. La elaboración de las metas propuestas en FreeStation se apoyan en el estudio de materias previas que son afines al desarrollo web, prototipado de aplicaciones, programación distribuida y administración de servidores.

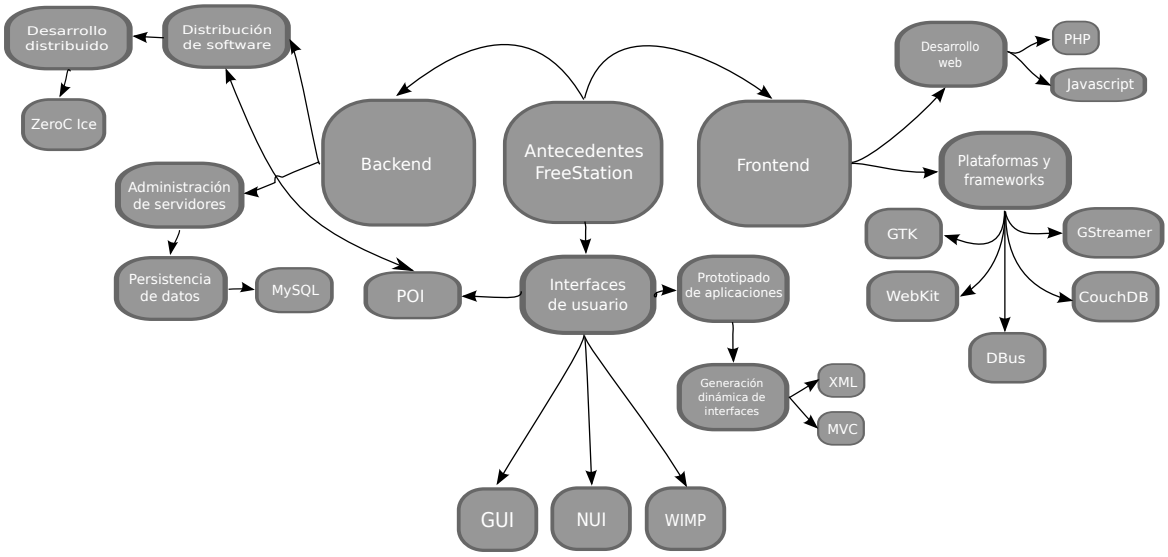


Figura 3.1: Mapa conceptual

En la Figura 3.1 se muestra un mapa conceptual de los contenidos descritos en este capítulo. Los antecedentes del proyecto se describen en base al *Frontend* del proyecto (necesario para la realización de componentes que integran la parte visual mostrada al usuario) y el *Backend* (necesario para llevar a cabo las tareas internas para los mecanismos implicados en la base de la aplicación).

La parte *Frontend* se basa en el desarrollo web (principalmente PHP y Javascript) y el uso de plataformas y frameworks como GTK, WebKit, DBus, Gstreamer o CouchDB.

Por otro lado, el *Backend* sirve al propósito de la distribución de software, aplicando métodos de programación distribuida que están basados en tecnologías de la biblioteca ZeroC Ice. Para ello se necesita de conocimientos de administración de servidores, en particular gestión de persistencia de datos (basada en MySQL).

A su vez el proyecto necesita de una base de interfaces de usuario, estudiando los diferentes enfoques de composición de aplicaciones GUI, NUI, WIMP y su utilización en POI. De esta forma se requiere del estudio de mecanismo para el prototipado de aplicaciones. Es posible aplicar dicho concepto a través de la generación dinámica de interfaces que tienen como pilares la composición en formatos XML y patrones de diseño MVC.

Dentro del desarrollo e investigación actual, la elaboración del marco de trabajo consiste en canales de distribución software generales[Lan10], con personalización sencilla para usuarios poco avanzados. Dichos medios de difusión, suelen requerir mucho esfuerzo a sus administradores y no cuentan con fáciles interfaces de administración. Pese al creciente acceso a conexiones de alta velocidad en el ámbito doméstico, los usuarios no disponen de un mecanismo cómodo de acceso a software e información directamente asociado a su perfil en una determinada organización.

Es pues, un tema abordado heurísticamente, pero con soluciones poco prácticas y elaboradas que requieren de una mejora constante atendiendo a las necesidades del usuario final.

En este capítulo se estudian algunas de las principales áreas en las que se basa el presente Proyecto Fin de Carrera. Estos contenidos serán citados en capítulos posteriores, por servir de base teórica para el diseño y desarrollo de FreeStation.

3.1. INTRODUCCIÓN

La simplicidad de un componente de acceso público facilita a comunidades de clientes la manera de obtener datos, informes o documentación de uso público.

Los residentes locales pueden ver más factible encontrar la información que necesitan en un momento preciso si ésta se encuentra orientada en el contexto de su situación actual. Esto puede ahorrar tiempo y costes, en lugar de realizar a una llamada a una operadora de información.

Éste el punto fuerte de los kioscos: ofrecer un mayor nivel de información sin requerir una gran infraestructura. De este modo es posible ahorrar en recursos manteniendo el nivel de calidad al usuario final. En los kioscos informativos, la información es actualizada al momento y no está sujeta a errores y confusiones entre operadores intermediarios.

El origen de los kioscos proviene de Persia. Los otomanos adoptaron el concepto a través de los sultanes. Estos lo usaban en sus residencias locales para disponer de un mercado cercano, sin gastar gran cantidad de tiempo en viajes extraordinarios. El concepto también fue exportado a Europa para venta de cigarrillos, flores o artículos exóticos. La gran ventaja era la variedad ofrecida en un pequeño punto en particular y cercano. Los kioscos multimedia siguen un propósito parecido en un ordenador.

Los kioskos son un gran mercado en auge. La empresa Griffin Chase Oliver lleva produciendo kioscos desde 1978. Su tasa actual de fabricación son 150 unidades por mes.[1]

Otras empresas interacciones como Four Winds Interactive (FWi) poseen el ranking de empresa número 660 en 2011 de las 500 empresas más importantes en Estados Unidos. Es una de las compañias de América que más rápido han crecido en los últimos 3 años. Los puestos directos e indirectos se estiman en 350.000 con unos beneficios de 366 millones de dolares en el año 2011[2]. FWi ha basado su negocio en desarrollar más de 65.000 kioskos digitales para hospitales, centros educativos y otros.

[1] Griffin Chase Oliver:
`http://www.griffinchaseoliver.com/Library/downloads/how_to_budget_pop_displays.pdf`

[2] Four Winds Interactive:
`http://www.fourwindsinteractive.com/press/pr-2011-inc-5000.htm`

3.2. POI Y PROCESO DE DISTRIBUCIÓN

Los terminales de tipo «Puntos de interés» (Point Of Interest POI) han avanzado significativamente con la tecnología de los últimos años, habilitando su integración en casi cualquier lugar con un uso sencillo (la empresa Griffin Chase Oliver creó desde 1990 POI analógicos para bancos).

Esta tecnología integrada ha permitido la interacción en edificios con sus visitantes. En determinadas instituciones como colegios, institutos, universidades, ayuntamientos u oficinas de turismo es común ver pequeños puntos donde se encuentran terminales mostrando información a los transeúntes. Algunos terminales incorporan mecanismos de interacción mediante teclado, ratón o incluso de forma táctil, pero estos paradigmas pueden ser mucho más complejos

En la panorámica del proceso de distribución de información, o más concretamente del proceso de distribución software, la tendencia actual marca la utilización de las redes de comunicación[Ver02]. La potencia actual de las mismas permite satisfacer las necesidades de información de los usuarios en base a arquitecturas específicas adaptadas a la constante evolución de los contenidos.

Figura 3.2: POI de la Policía para DNIe

Los últimos servicios de la administración se mueven y avanzan en la dirección de incorporar las últimas tecnologías al alcance del usuario regular. Este avance requiere de especificaciones concretas y herramientas que permitan automatizar los procesos involucrados.

Varios de estos servicios se han empezado a ofrecer al público de forma experimental en ciertos entornos de explotación específicos.

Uno de estos grandes hitos ha sido la incorporación del DNI electrónico (DNIe) y sus consecuentes POI para facilitar trámites burocráticos (pago de multas, renovación de datos, solicitud de certificados, etc).

En la imagen de la figura 3.2 se observa un POI en una comisaría de policía. Dispone de un teclado, trackball y un lector de DNI electrónicos.

Figura 3.3: POIs de la estación de trenes ADIF (a), Sescam (b) y (c)

Es interesante recalcar que cuenta con un cable de conexión a internet para descarga de información. El punto de interés se encuentra ubicado en la entrada a la comisaría. La entrada está cubierta para evitar daños ambientales por lluvia u otros elementos, pero es accesible al público en cualquier momento del día o de la noche.

La vigilancia del POI está permanentemente asegurada por cámaras las 24 horas y de forma presencial durante el horario diurno.

Como se puede apreciar en la imagen de la figura 3.2, la pantalla muestra con más detalle campos de entrada para el usuario y una simple interfaz accesible con botones. La altura del POI esta ajustada para gente discapacitada en silla de ruedas e incorpora síntesis de voz para personas ciegas.

En la imagen de la figura 3.3 se corresponde a un POI en una estación de trenes Adif. El diseño es totalmente diferente a los analizados con anterioridad. Muestra una gran pantalla en vertical y rotada. Es posible que disponga de funcionalidad táctil y conexión a internet.

Es remarcable la incorporación de un sistema de audio para persona ciegas y como posible utilización de confirmación para notificaciones de audio. Por sus características no parece ser muy adecuada para personas en sillas de ruedas, por lo que su funcionalidad puede estar limitada a un panel de información de baja interacción.

La imagen de la figura 3.3 (b) es otro ejemplo de POI interesante y bastante diferente ya que dispone de un teléfono o intercomunicador para asistencia directa, teclado, trackball, ratón y lector de tarjetas sanitarias. Existe una bandeja impresora para enviar datos útiles impresos como informes al usuario. En la imagen de la figura 3.3 (c) puede apreciarse con mayor detalle la funcionalidad del punto de interés.

Este es un conjunto reducido de ejemplos de la gran variedad de POIs. Normalmente varias unidades similares se pueden encontrar en los mismos centros, universidades, estaciones de policía, hospitales, etc. El modelo de administración actual de los POIs causan que sea difícil su mantenimiento. Es muy habitual que, con el tiempo, cada unidad de POI presente averías, fallos software, inconsistencias.

De este modo, es necesario disponer de una plataforma para gestionar y solucionar, en la medida de los posible, este tipo de problemas. El sistema de gestión debe ser capaz de poner en producción y configurar POIs con diversas versiones de software, notificaciones de estado, así como reinicio automático en caso de fallo grave, FreeStation trata de cubrir estas necesidades en gestión y mantenimiento de POIs heterogéneos.

3.2.1. GUÍAS DE DISEÑO EN POI

Como casos de estudio de gran éxito en el desarrollo de POIs pueden citarse el Sistema de Información de la Expo'92 de Sevilla[Mag07] y el Sistema de Mensajes de los Juegos Olímpicos de 1984[GoB87].

Los sistemas POI son diseñados para ser usados en modo "*walk up and use*", es decir, para caminar y usar. Ello hace mención a que los sistemas POI deberían ser tan autoexplicatorios como fuera posible. Los usuarios tienen poco tiempo para usar el sistema, por ello el sistema debe ser capaz de producir la información necesaria de forma rápida. Si un usuario queda atascado en la interacción, se frustrará y no volverá a usarlo.

Dichos sistemas, también deben detectar si el proceso de interacción queda abandonado y volver al estado inicial si no hay nuevas acciones. El temporizado para volver al estado inicial, no debería ser pequeño, para evitar frustraciones para usuarios que necesitan de un mayor tiempo para llevar a cabo sus acciones (como lectura extensa de textos).

En esta sección hemos estudiado la distribución de POIs y su evolución. En la siguiente sección se explicará la caracterización de funcionalidades posibles.

3.3. CARACTERIZACIÓN DE LOS POIS

Aparte de las guías de diseño, un POI puede estar caracterizado según las funcionalidades que estén presentes para el usuario. A continuación se describen algunas:

- **Entrada de datos por voz**
  La entrada datos por voz incrementa la fiabilidad de uso del usuario[Mag07], independientemente de la confiabilidad del sistema de reconocimiento por voz.

  En un POI pueden incorporarse teléfonos, auriculares, micrófonos u otros dispositivos que permitan mejorar la interacción del usuario y faciliten el uso a personas que no dispongan de las habilidades necesarias. El principal problema es la distinción de palabras clave para el sistema. Este campo ha avanzando significativamente en los últimos años, pero la tasa de error todavía es considerablemente alta.

Estos métodos implican también la pérdida de privacidad y contaminación de ruido en un entorno abierto al tener que producir verbalmente los datos.

A pesar de esas limitaciones, los beneficios aportados por la entrada de datos por voz son más significativos que la entrada de datos estándar de texto mediante teclados.

- **Música ambiente y sonidos de salida**
  La música en un POI puede ofrecer información extra[Mag07] para presentación multimedia. Puede complementar a textos, iconos e imágenes para impactar al usuario en mayor grado.

  Es importante que usuario no perciba como intruso la incorporación de un sonido ambiente o incluso distraiga del contenido principal. Los sonidos cortos y agudos provocarán rechazo en el usuario al igual que bucles intermitentes muy cortos de sonido ambiente.

- **Estructura y navegación**
  La estructura debe ser simple[LoM99] para que el usuario se sienta cómodo en la interacción. El sistema debe tener un punto de entrada al que el usuario debe poder volver en cualquier momento que desee. Normalmente a este punto inicial se le denomina Pantalla de bienvenida o menú principal.

  El resto de contenido idealmente debe estar estructurado en pantallas seleccionables desde esta etapa. Es recomendable si existe una jerarquía de contenidos presentar una secuencia del camino recorrido. Esto conseguirá un mejor posicionamiento conceptual de los contenidos.

  Cada pantalla debe ser fácilmente distinguible de otras secciones y representar acorde a sus contenidos la información. Se puede proporcionar un menú básico de controles como *Inicio/Fin/Atrás/Siguiente/Cancelar/Salir* que ayude al usuario a tomar el control en sus acciones. En puntos críticos debería mostrarse una doble confirmación al usuario mediante menús emergentes o reiteración de la acción.

- **Personalización**
  La personalización en la interfaz debe permitir adaptar la capa de representación de la información a los gustos de cada grupo de usuarios.

  Posibilitar configuraciones al usuario permite una mayor integración[Fis00], pero complica el desarrollo. Algunas características como los tamaños de las fuentes tipográficas, color o fondo suelen ser fácilmente adaptables ofreciendo así mecanismos básicos de personalización de interfaz.

- **Prueba de POIs en producción**
  Como sistema expuesto al público general un POI debería ser probado siguiendo una serie de factores determinantes.

En un entorno real, los sujetos tendrán una experiencia previa o serán totalmente ajenos al servicio. Los usuarios sin experiencia tienden a producir problemas desconocidos y probar más profundamente el sistema.

Si existe la posibilidad, se deberían incluir usuarios con ciertos perfiles desabilitados para algunas funciones o con funcionalidad reducida y observar su comportamiento. Esto facilitará probar partes del sistema por separado y centrarse en problemas específicos.

En pruebas de carga del sistema, el rendimiento será crítico y aparecerán errores desconocidos. Un sistema eficaz debería conocer sus límites máximos de forma precisa[Mag07].

- **Selección de idiomas**
  En una comunidad local puede ser necesario habilitar varios idiomas dependiendo de la funcionalidad del POI. Un POI orientado a turismo idealmente debería tener un número elevado de idiomas disponibles.

  El esfuerzo adicional de incorporar un idioma a menudo no es compensable al número de usuarios que pueden utilizarlo, pero evita restringir la cuota de acceso a dicho idioma. La alternativa más común es habilitar un conjunto de iconos por idioma o banderas por territorio[Mag07]. Si no es posible, una lista despegable con las opciones de idiomas o una pantalla dedicada al cambio de idioma.

  En general las opciones utilizadas con simbología o iconos son más rápidamente accesibles para uso de otro idioma, ya que una lista de texto confunde al usuario y retrasa su tiempo de percepción en el cambio de idioma. En el proceso, el usuario debe ser capaz de realizar el cambio de idioma sin necesidad de hablar un idioma complementario.

- **Posicionamiento físico y localización**
  En un comienzo la localización de un POI debe ser anunciada por el servicio del centro. Esto permite dar conocimiento exacto a sus usuarios. Elementos como indicadores en revistas, notas en puertas o ventanas adyacentes pueden ayudar al usuario.

  Se recomiendan indicadores de alto contraste en blanco/amarillo con caracteres en fondo negro[Mag07] para usuarios con visión reducida. También existe la posibilidad de proporcionar tarjetas que emitan un sonido audible hacia la localización del POI.

  La manera más efectiva de que los usuarios encuentren el POI es situándolo en el flujo normal del recorrido de usuario, por ejemplo en la entrada del edificio o en el patio central de reuniones. Un POI de tickets de tren, debería ser posicionado a la entrada de la estación y no en su final, para facilitar la compra de los usuarios antes de entrar al tren.

  Fomentar el uso de un POI por usuarios desarrolladores puede ser una buena manera de involucrar a más personal en el mismo. Facilitar previamente POIs con demostraciones

de ejemplo y obtener retroalimentación de los usuarios casuales, permite disponer de un criterio formado sobre su uso y la experiencia de visualización.

Según el objetivo de uso, pueden desarrollarse distintas formas de presentar el contenido para rangos de edad de usuarios. Por ejemplo, es muy posible que personas con edad más avanzada prefieran botones mayores con colores simples y pocas animaciones.

- **Privacidad**
  Los usuarios que utilizen un POI público pueden ser observados de cerca por otros usuarios. El nivel de privacidad depende de los datos a visualizar.

  Por ejemplo, la privacidad para consultar en un POI de turismo, suele ser baja ya que no se requieren datos personales concretos. Sin embargo, en POIs donde por ejemplo se necesiten introducir datos privados como un DNI para consultas jurídicas puede ser necesario un alto grado de privacidad.

  En el desarrollo de POIs públicos se recomienda requerir la mínima información posible por parte del usuario. En el caso de necesitar estrictamente datos confidenciales, el POI público debería de ser cerrado en una cabina donde facilitar la privacidad de la entrada de datos del usuario. Estas medidas evitarán el rechazo por parte del usuario a introducir datos.

  Si se opta por esta última opción, para personas con discapacidad motriz debería proporcionarse un acceso conveniente.

- **Sencillez en menús**
  Proveer menús de entrada sencillos para el usuario. Deben tener una redacción concisa y clara para que el usuario tenga plena conciencia de la acción a realizar. Adicionalmente los textos adicionales como leyenda pueden enriquecer la acción. Según Ben Shneiderman el número máximo de elementos a recordar en un menú por una persona tiende entre 3 y 7 de promedio[Shn09]. Un número más elevado resultará en una interfaz sobrecargada y de uso complicado. La estructura de las listas de menús debe estar cuidadosamente estudiada. Es conveniente seguir algún orden de posicionamiento, bien por relevancia o simplemente orden alfabético (por ejemplo Buscar, Imprimir, Mostrar, Salir) Es importante recalcar espacios entre opciones para ayudar a enfatizar la estructura. Si existen listas de menú numeradas, se deben evitar los huecos en la secuencia normal. Por ejemplo, del 1 al 9, no sería acorde saltar la opción 5, en su lugar renombrar del 1 al 8. De igual modo, sería conveniente evitar abreviaciones en opciones de menú que sean poco conocidas al público, etc.

3.3.1. COMPONENTE SOCIAL

Combinando el componente social[WaF94] de un objeto que se encuentra en el exterior con la inmensidad de contenidos que pueden ofrecer una red tecnológica es posible realizar una distribución de contenidos uniforme orientada a docencia, turismo o cualquier sector que requiera de un despliegue de contenidos.

Un objeto accesible en exteriores fomenta la participación del entorno. Inicialmente despierta una gran curiosidad y una rápida propagación si ofrece contenidos o información de utilidad para sus usuarios. Su practicidad queda demostrada en casos de consultas urgentes, donde no existan persona físicas que puedan facilitar la información o en horarios no laborales.

3.3.2. BENEFICIOS Y PROBLEMAS JURÍDICOS

La distribución de software conlleva una responsabilidad asociada al marco de la legalidad. Cualquier software dispone de una licencia que habilita una serie de acciones permitidas, como es la distribución. En lo particular, pueden surgir problemas jurídicos derivados si un POI no cuenta con la debida autorización. Luego, estableciendo una determinada base tecnológica con tendencia en la distribución de software libre, los problemas legales en este aspecto suelen ser nulas o muy reducidas.

Por otro lado, la distribución de software beneficia al usuario final ya que la compra individual supone requerir o disponer de unos costes asumibles por cada software particular. La elección entre un gran catálogo de software ahorra tiempo al usuario en la búsqueda y obtención del mismo.

Este enfoque añade una serie de ventajas asociadas, como la alta disponibilidad de nuevas versiones y actualizaciones de seguridad. En el caso del Software Libre, estas características de alta disponibilidad pueden traducirse igualmente en la personalización de distribuciones para diversos colectivos de usuarios.

Estos paquetes de contenido específicos (formados por software, documentación y ficheros multimedia) pueden ser adaptados a necesidades concretas (docentes, de investigación o profesionales), ahorrando gran cantidad de costes (temporales y económicos).

Por lo tanto, en un ecosistema basado en aplicaciones de software libre que basa su distribución en estándares abiertos sin prácticamente restricciones, la participación y progreso es elevado. La practicidad de licencias como GPL, LGPL, BSD, MIT, etc, permite usar el software libremente, copiarlo, publicarlo, distribuirlo, siempre que se incluya la nota de copyright en todas las distribuciones.

3.3.3. ESTUDIO DE MERCADO

El entorno en el que se desenvuelve este PFC es el ámbito empresarial y de organizaciones que necesitan comunicar y dotar a su público de puntos de interés orquestándolos de una manera cómoda y sencilla.

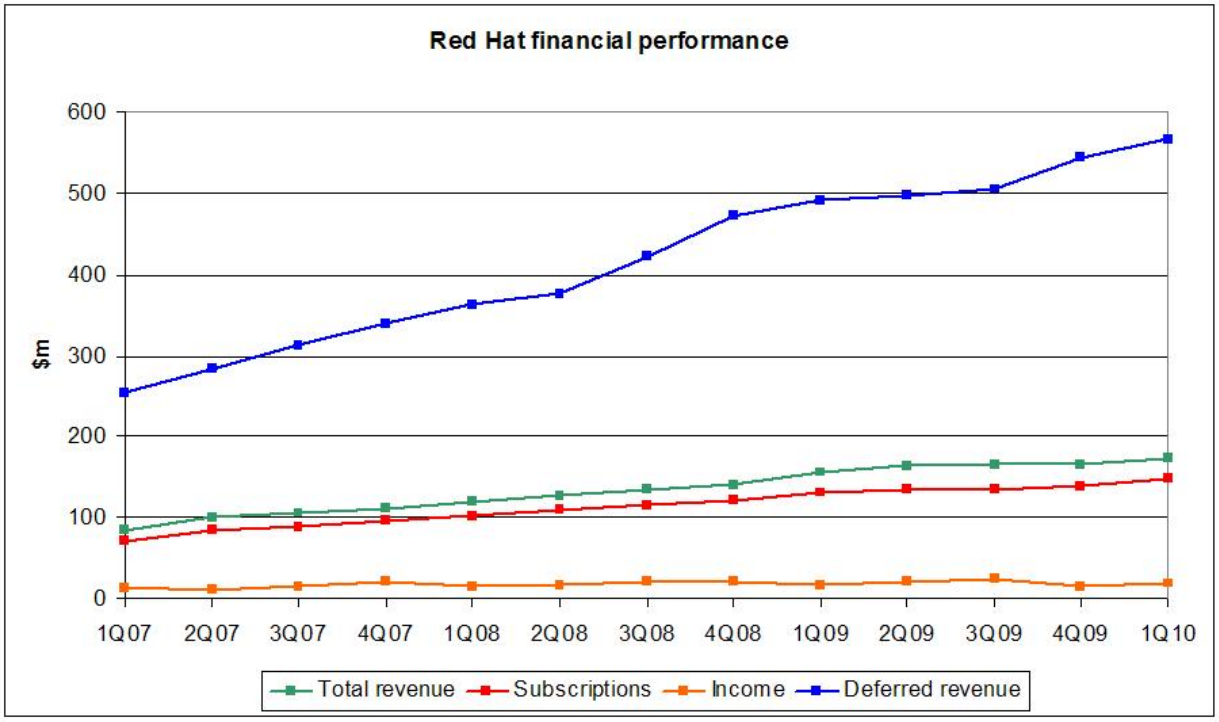


Figura 3.4: Rendimiento financiero RedHat 2007-2009

La necesidad y demanda de un mercado creciente en el sector del software libre[Asl09] garantiza la necesidad y aplicación de POIs en el futuro. La industria se está expandiendo más rápidamente a nivel mundial debido a las crecientes necesidades de los mercados emergentes. La reducción de costes y aumento de beneficios gracias al software libre es un hecho para los sectores educativos y empresariales.

Los beneficios de empresas basadas en software libre como RedHat han mostrado un crecimiento alcista desde su creación, alcanzado cifras de un billón de dolares americanos en 2012[McM12]. En el gráfico de la figura 3.4 se observa el gran crecimiento financiero para RedHat.

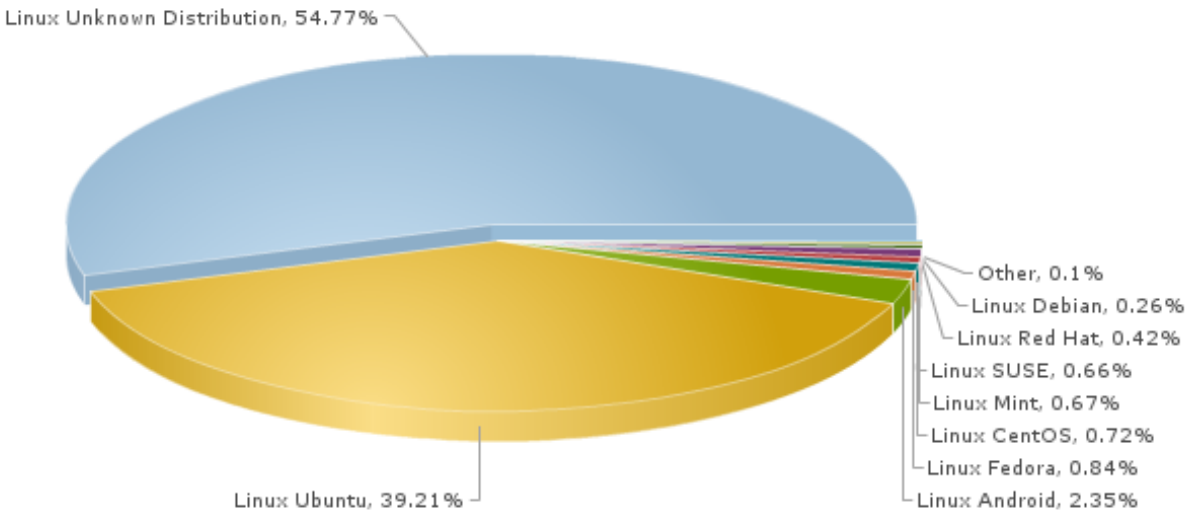


Figura 3.5: Cuota de mercado de distribuciones GNU/Linux

Otro caso de éxito es Ubuntu, perteneciente a la empresa Canonical. Como muestra la figura 3.5 las estadísticas web sugieren que el porcentaje de mercado de Ubuntu dentro de distribuciones GNU/Linux es de aproximadamente un 49 %[3], y con una tendencia a subir como servidor web.

En los últimos años las instituciones públicas poco a poco han realizados migraciones importantes hacia el software libre. Islandia aprobó un decreto en 2008 que se aplica contundentemente para la totalidad de las instituciones públicas.[4]

[3] StatOwl :Operating System Version Market Share:
`http://statowl.com/operating_system_market_share_by_os_version.php?1=1&timeframe=last_6&interval=month&chart_id=4&fltr_br=&fltr_os=&fltr_se=&fltr_cn=&limit[]=linux`

[4] Free and Open-source Software - Government Policy of Iceland:
`http://eng.forsaetisraduneyti.is/information-society/English/nr/2882`

En 2007, el Ministerio de Educación y Ciencia de la República de Macedonia desplegó más de 180.000 equipos de escritorio con Ubuntu preinstalado para su uso en las aulas, y animó a cada estudiante del país a usar computadoras con Ubuntu.[5]

La policía francesa, desde 2009, está en proceso de instalar Ubuntu en 90.000 estaciones de trabajo, demostrando un 70 % de ahorro en el presupuesto de TI sin tener que reducir su capacidad.[6]. En 2011 tuvo un importante incremento activo de 20 millones de usuarios[7].

La demanda de necesidad es creciente en el ámbito empresarial y las administraciones públicas. Aproximadamente el 39 %[8] de empresas de España afirman haber comercializado productos bajo una licencia de software libre o fuentes abiertas o haber realizado actividades relacionadas con dicha tecnología.

La cifra de negocio estimada derivada de la venta de productos de software de fuentes abiertas alcanzó en 2010 los 776 millones de euros. Esta cifra representa el 3,36 % del total del sector de Servicios Informáticos y dio trabajo a cerca de 40.000 personas que estuvieron implicadas en proyectos de consultoría o desarrollo de software de fuentes abiertas.

La formación también es un aspecto clave en el negocio del software libre. Aproximadamente la mitad de las empresas (52 %) han proporcionado a sus empleados formación sobre software de fuentes abiertas.

La tendencia en general es que la mayor parte de las empresas que en la actualidad comercializan productos o servicios basados en software de fuentes abiertas, el 86 %, afirman que en el medio plazo, de aquí a 5 años, seguirán comercializando este tipo de tecnología.

[5]Every Student in the Republic of Macedonia to Use Ubuntu-Powered Computer Workstations:
`http://www.ubuntu.com/news/macedonia-school-computers`

[6]French police: we saved millions of euros by adopting Ubuntu:
`http://arstechnica.com/information-technology/2009/03/french-police-\`
`saves-millions-of-euros-by-adopting-ubuntu/ http://www.canonical.com/sites/default/`
`files/active/Casestudy-GendarmerieNationale.pdf`

[7]Retail Stores in China:
`http://blog.canonical.com/2011/10/27/retail-stores-in-china/`

[8]CENATIC: El Software Libre en el Sector Español de Servicios Informáticos
`http://observatorio.cenatic.es/index.php?option=com_content&view=article&id=741:`
`el-software-libre-en-el-sector-espanol-de-servicios-informaticos&catid=13:`
`empresas&Itemid=23`

Entre las empresas que actualmente no comercializan soluciones libres, una de cada cinco empresas tiene previsto empezar a trabajar con este software con el fin de poder ofrecérselo a sus clientes. Casos de éxito como en Andalucía, donde se instalaron 220.000 equipos con Ubuntu [9]. El software Lliurex (derivado de Ubuntu) permite ahorrar 30 millones a la Generalitat desde 2005[10]

Las empresas de software libre tienen un mayor crecimiento en sectores empresariales, servidores o en educación donde se exploran formas interactivas para dotar a los alumnos de mayores facilidades.

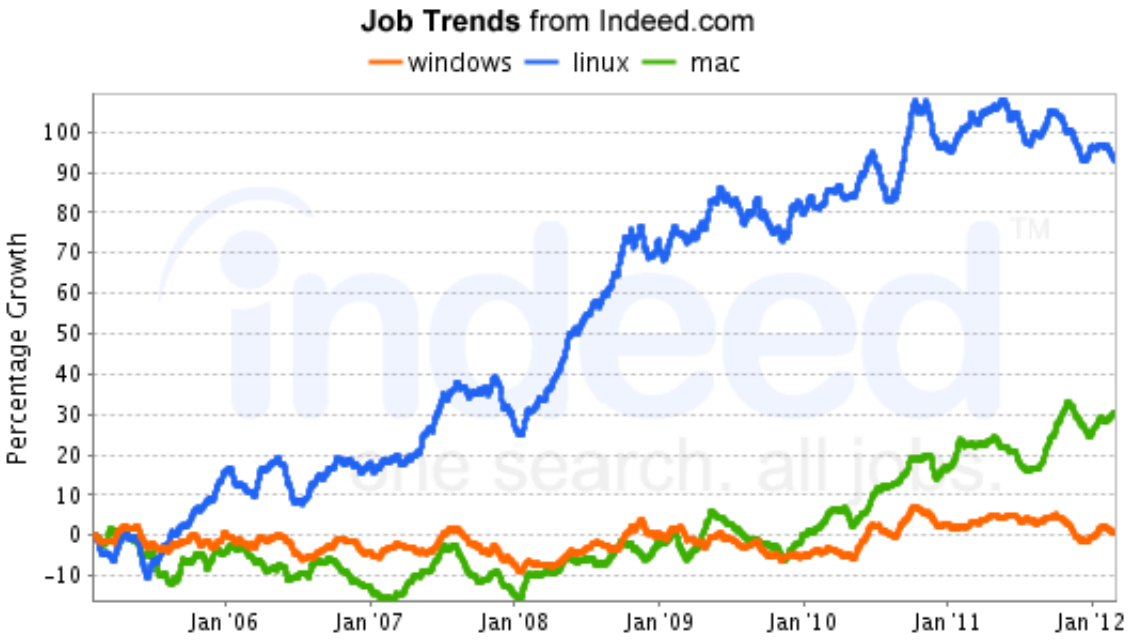


Figura 3.6: Gráfico de tendencias de trabajo para Windows, Linux, Mac 2006-2012

Asimismo, un ecosistema de aplicaciones derivadas puede surgir como forma de personalizaciones y adaptaciones al software original, lo que garantiza un buen mercado de explotación para los desarrolladores. En la gráfica de la figura 3.6 se puede observar una comparación entre los trabajos más demandados para diferentes plataformas. Según [Ind12] en los últimos años la mayor demanda de trabajo ha sido experimentada para la plataforma GNU/Linux.

[9] Andalusia deploys 220,000 Ubuntu desktops in schools throughout the region:
`http://www.canonical.com/about-canonical/resources/case-studies/andalusia-deploys-220000-ubuntu-desktops-schools-throughout-r`

[10] CENATIC: El software Lliurex permite ahorrar 30 millones a la Generalitat desde 2005:
`http://www.cenatic.es/hemeroteca-de-cenatic/3-sobre-el-sector-del-sfa/40024-el-software-lliurex-permite-ahorrar-30-millones-a-la-generalitat-desde-2005`

En esta sección hemos estudiado el mercado relativo al software libre. Uno de los objetivos de FreeStation es facilitar su distribución en POIs, que cuentan con elevados requisitos relativos a la facilidad de uso y a los mecanismo de interacción. En la siguiente sección estudiaremos los principios de diseño de interfaces naturales.

3.4. EL DISEÑO DE INTERFACES NATURALES

Una interfaz es el medio de comunicación entre un usuario y un ordenador. Requiere de un lenguaje preciso para comunicar al usuario con el ordenador. La construcción de una interfaz de usuario natural o NUI sigue unas líneas guía de diseño específicas. El término natural mimetiza con el significado de "mundo real". Es un proceso iterativo que crea un producto asumiendo que el usuario debe sentirse libre. Una interfaz debe hacer que usuario actúe de forma natural y que esta se sienta natural. La utilización de interfaces naturales a través de un diseño riguroso y aprovechamiento potencial de tecnologías modernas permiten una utilización con mayor similitud a las habilidades humanas.

Un ejemplo de ello, son las interfaces de usuario desarrolladas en un ambiente de dispositivos multitáctiles o *multi-touch* que mantienen una entrada de usuario muy adecuada a una forma natural de interacción. Comparándolo con las interfaces desarrolladas para ratón, permiten la implementación de nuevos paradigmas de interacción, potenciando la expresividad de las acciones del usuario. Al contrario que las interfaces WIMP diseñadas para una interacción basadas en ratón, las interfaces *multi-touch* aprovechan mejor el concepto de memoria espacial del usuario y tienen una gran aceptación en la información que realmente llega a percibir el usuario en su uso.

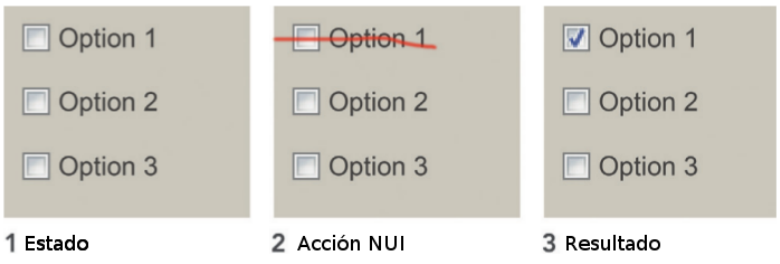


Figura 3.7: Selección NUI de una caja de comprobación: (1) Estado inicial con los elementos desactivados (2) Interacción del usuario con un gesto natural para selección horizonal (3) Resultado con caja de comprobación seleccionada.

Las interfaces NUI aprovechan el contexto como principio fundamental de diseño, empleando mecanismos de sincronismo y favoreciendo la componente social de la interacción (colaboración múltiple).

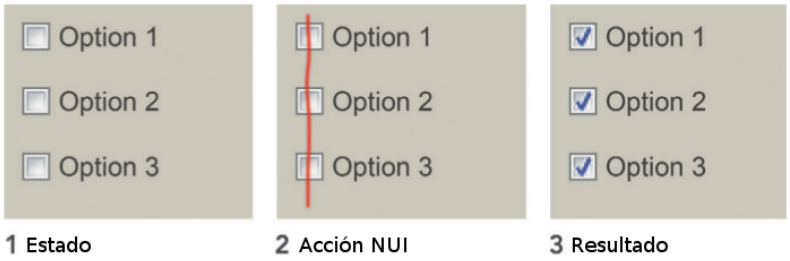


Figura 3.8: Selección NUI de múltiples cajas de comprobación: (1) Estado inicial con los elementos desactivados (2) Interacción del usuario con un gesto natural para selección vertical (3) Resultado de todas las cajas de comprobación seleccionadas.

En las figuras 3.7 y 3.8 se muestran ejemplos ilustrados de hipotéticas interacciones NUI donde el usuario tendría un beneficio de interfaz adaptada a un uso más natural. La desventaja de utilizar una primitiva de interacción como la selección cruzada es que puede existir un alto grado de falsos positivos debido a una interacción descuidada por parte del usuario. Por ello, este nuevo paradigma permite el desarrollo de interfaces que ofrezcan un comportamiento más cercano al que espera intuitivamente el usuario.

Por tanto, Las interfaces de usuario se deben elaborar, investigar y someterse a un proceso de ingeniería como parte de una aplicación. La mayoría de creadores de interfaces no analizan realmente una interfaz de usuario, sino que se limitan a reaprovechar el diseño de otras interfaces similares y adaptar de forma iterativa la interfaz a sus requisitos.

3.4.1. LA COEXISTENCIA DE MODELOS DE INTERFACES

Los conceptos de NUI y GUI probablemente coexistan prósperamente en el futuro. Por ejemplo, una NUI puede estar bien adaptada en un nicho de mercado como el entretenimiento. Contribuye a destacar en la visualización de contenido, ejemplos interactivos, animaciones y juegos. Al mismo tiempo, el contenido viene transferido desde el mundo GUI, donde se necesita para interacciones como acciones e integración de contenido.

Los elementos entre ambos son transferibles. Un botón de GUI de una aplicación de escritorio, puede extrapolarse fácilmente para una pantalla de navegador en un coche, adaptando la idea de interacción mediante dispostivo táctil o comando de voz.

Pero esto no es cierto en la mayoría de ejemplos de NUI a GUI. Una interfaz NUI que disponga de un diseño horizontal y vertical, no adapta bien al modelo simultáneo en una GUI. Las interfaces NUI toman el fuerte principio de la acción del usuario y adaptan la interfaz al uso tradicional.

3.4.2. MENOS ES MÁS

Iniciar un diseño simple es una oportunidad para construir interacciones simples que ayuden a aplicar tareas complejas. En cada generación de interfaces, los desarrolladores y diseñadores tratan con el desafío de construir aplicaciones para explotar las ventajas que ofrecen las nuevas interfaces.

El riesgo de construir nuevas aplicaciones reside en reutilizar los principios que fueron válidos y mayormente usados en las aplicaciones anteriores. Olvidar los estilos de interacción antiguos es necesario para innovar en aplicaciones tradicionales. Sin embargo, debe tener el uso fundamental y asumido por la experiencia de usuario anterior.

Las guías de diseño de interfaces NUI recomiendan:

- Probar las mecánicas fundamentales de las interacciones primarias antes de reconstruir o desechar cualquier interacción. Si el usuario acepta los cambios de forma positiva, continuar el desarrollo con cambios leves en cada iteración.
- Estudiar el diseño de aplicaciones NUI existentes. En general, los usuarios se comportaran de forma similar para procesos NUI ya probados y aceptados.
- Restringir el dominio de mecánicas de interacción. Un buen enfoque para desarrollar una tarea compleja de interacción, es dividir la tarea en regiones y tratar cuidadosamente cada parte de diseño.
- La interacción debe ser divertida y recompensar al usuario con algún beneficio por su uso.
- El usuario debe focalizarse en el contenido. Por ejemplo la compartición de fotos, la exploración de configuraciones de productos, etc.

- Tener al contexto de la aplicación como ciudadano de primera clase. La simbiosis entre el entorno y el usuario conllevan varias implicaciones para el diseño y su evaluación.

3.4.3. LA INTERFAZ DERIVADA DE USUARIO

> All science is experiential; but all experience must be related back to and derives its validity from the conditions and context of consciousness in which it arises, i.e., the totality of our nature. — Wilhelm Dilthey

Traducido al español:

> Toda la ciencia es experimental, pero toda la experiencia debe estar relacionada con retroceder y deriva su validez de las condiciones y el contexto de la conciencia en las que surge, es decir, la totalidad de nuestra naturaleza. — Wilhelm Dilthey

El enfoque correcto para crear una interfaz natural NUI debería ser basado en un sistema democrático. Permitiría a los usuarios definirlo desde su origen.

Un método para llevara a cabo este propósito puede ser ofrecer a los usuarios varios estados finales de la aplicación y sus funcionalidades y permitir al usuario elegir los más convenientes. Destilando las acciones por la mayoría de usuarios debería resultar en una síntesis de opciones bastante correcta.

Al enfoque de crear una interfaz de usuario basada en el contexto proporcionado por usuarios y sus acciones se denomina “Interfaz derivada de usuario” o del inglés The User-Derived Interface (UDI)

El enfoque UDI ha sido bastante útil y exitoso para interfaces pequeñas y que no requieren de entrenamiento. Por ejemplo para interfaces de comando por voz, suelen ser útiles para conocer que palabras exactas demandan los usuarios para realizar una acción (“Enviar un correo a Juan”)

En un proceso iterativo el analizador incluiría sinónimos, desambiguación o palabras similares con ligeros errores de pronunciación. En resumen, crear una interfaz NUI requiere de mucha retroalimentación de usuarios, para elegir buenos métodos de desarrollo y evaluación.

Normalmente un método efectivo de evaluación es el conocido como RITE (Rapid Iterative Testing and Evaluation). Es un método diseñado para un desarrollo y diseños rápidos. Anima a interactuar con el aprendizaje de nuevos paradigmas y hacer esfuerzos en la maneja de pensar el diseño con retroalimentación perpetua.

En esta sección hemos estudiado el diseño de las interfaces naturales y como sus principios afectan en los objetivos de Freestation para el desarrollo de interfaces. En la siguiente sección estudiaremos la generación dinámica de interfaces.

3.5. GENERACIÓN DINÁMICA DE INTERFACES

La inteligencia artificial trata de simular las capacidades humanas mediante el uso de máquinas y algoritmos. La comunicación es un campo imprescindible para la simulación de las capacidades humanas. Puede darse entre máquinas y máquinas o bien entre máquinas y seres humanos. El el segundo caso, la materia IPO trata con el análisis, diseño, implementación y evaluación de la interfaz de usuario[LoM99].

La investigación de Ben Shneiderman[Shn09] está relacionada en este campo y ha contribuido con importantes conceptos como la “Usabilidad Universal” o las 8 reglas de oro [11] que regulan el diseño de interfaces de usuario.

Existe una enorme comunidad de investigadores dedicados al estudio de técnicas que mejoren la usabilidad[Shn89], ya que una buena gestión de la interacción supone la utilización de forma eficiente de una aplicación.

La generación dinámica de interfaces surge de la necesidad de disponer de un enfoque generalizado durante la etapa de diseño de una aplicación[Bar09]. Se entiende como “generación dinámica de interfaz“ aquella que es creada en tiempo de ejecución.

[11] Las 8 reglas son: coherencia, usabilidad universal, retroalimentación, prevenir errores, reversión de acciones de forma sencilla, control por el usuario, carga cognitiva, diálogos resolubles.

Normalmente, las interfaces son generadas a través de soluciones estáticas. Estas son desarrolladas con unos requisitos base especificados que normalmente no varían en modelos normales. La especificación de requisitos se convierte en una de las partes de mayor relevancia para la obtención de software de calidad.

Debido a que son desarrollos a medida y muy específicos, se crea una situación en la que se hace prácticamente imposible la reutilización de interfaces[Fis00], al menos sin implicar un gran coste de desarrollo en tiempo y recursos para adaptar las interfaces a las nuevas necesidades. Una interfaz de fácil aprendizaje conlleva muchas iteraciones para refinar un resultado final.

Por lo tanto, el enfoque más lógico es evitar las dependencias particulares de las interfaces y tratar de aislar lo máximo cada componente. Las interfaces gráficas se generadas de forma dinámica se adaptan en la medida de lo posible a las características de cada proyecto.

Esta abstracción se consigue normalmente a través de estructuras de datos asociadas a metadatos (datos que describen otros datos). Los metadatos permiten explicitar relaciones entre datos y visualización de los mismos.

Existen enfoques prácticos de desarrollo de interfaces de usuario basados en modelos de esquemas XML[IEE05] con fácil acceso mediante una jerarquía DOM[W3C00].

Por ejemplo, enGTK+, se utiliza un visor de interfaces dinámicas llamado Glade, que permite generar una interfaz en XML[ZaM06] y cargar los componentes de forma dinámica y en tiempo de ejecución desde una aplicación.

De igual forma, las aplicaciones desarrolladas en el sistema operativo Android de Google, permiten la generación de vistas de aplicación basadas en XML. Para este tipo de enfoque, los pilares fundamentales deben ser definidos de forma muy concreta. De esta forma, una misma estructura puede utilizarse como base para proyectos con requisitos muy similares o incluso convertirse en un patrón específico para el desarrollo[LoM99].

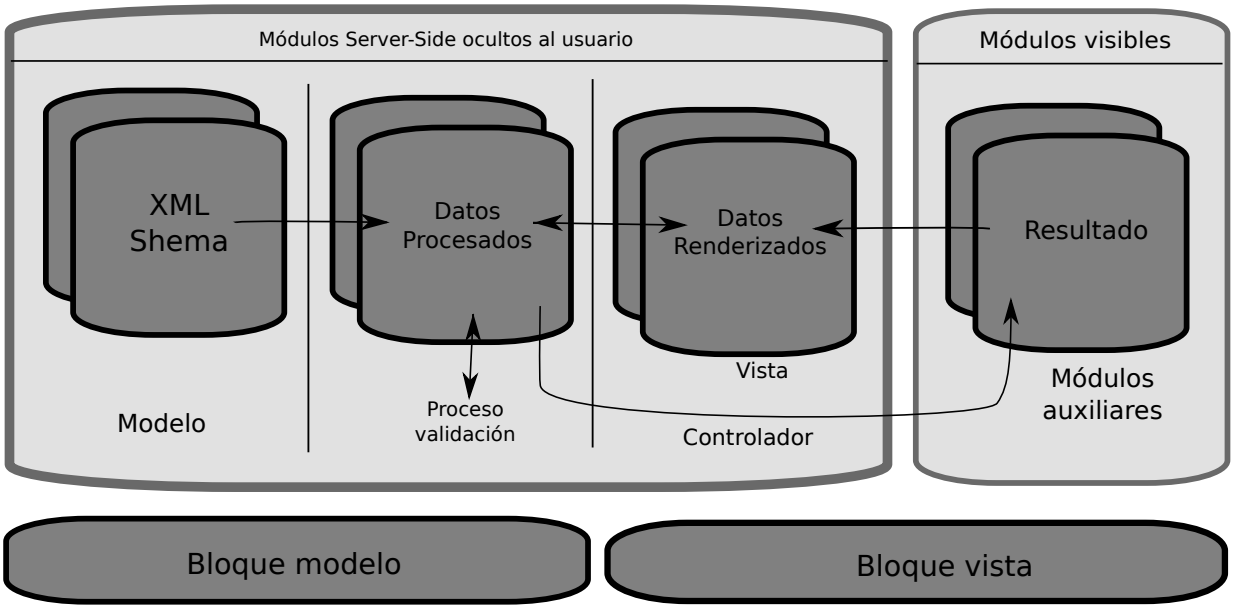


Figura 3.9: Esquema de modelo-vista-controlador con XML

La representación del modelo origen contiene los datos o requisitos de la interfaz. Estos pueden incluir campos, tipos de datos y multiplicidades, entre otros, que deben ser generados e incluidos en la interfaz propuesta.

La aplicación debe ser capaz de comprender dichos datos y generar de forma completa y correcta una interfaz. Incluso es posible que la aplicación deba adaptarse de forma dinámica a cambios en el modelo de entrada. Así el procesado de forma iterativa comtempla un comportamiento adaptativo.

De esta forma la aplicación no es susceptible de ser dependiente de modelos particulares y se evita por tanto una generación de interfaz estática o invariable.

Generalmente este modelo y enfoque es utilizado en esquemas de Modelo-Vista-Controlador o MVC. Es MVC es un patrón de estructura para el diseño software, que constituye tres componentes para proporcionar independencia con cada uno de ellos.

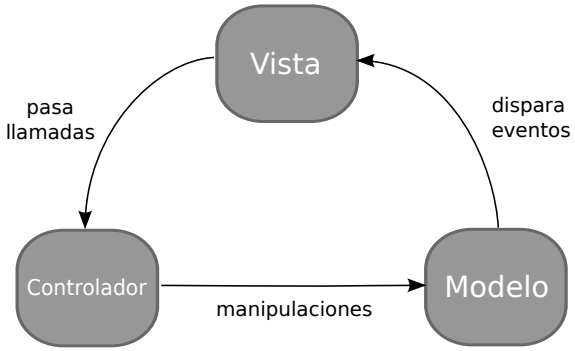


Figura 3.10: Interacción MVC

El modelo es la especificación XML o DTD (almacena la información y describe su formato), que posteriormente es analizado y validado en el controlador para constituir un modelo correcto y que sirve como componente intermedio para la comunicación y la gestión de los datos. Normalmente el formato del modelo sigue algún estándar definido por alguna organización como la W3C, IEEE o similares.

El análisis y validación puede realizarse en varias fases. Por ejemplo para omitir tipos de datos no requeridos o complejos. Una vez finalizado es expuesto o renderizado en la vista que realiza una representación gráfica de la información.

En esta sección hemos comprobado como la generación dinámica de interfaces son una opción más eficiente en aplicaciones donde no existen necesidades estáticas. En la siguiente sección estudiaremos un caso de estudio que modificará la interfaz para aprovechar las características de un buen diseño.

3.6. CASO DE ESTUDIO DE AMERICAN AIRLINES

En 2007 la compañía de aerolíneas “American Airlines” decidió realizar un completo estudio[Hud10] y rediseño de sus kioscos de compra de billetes y check-in.

La aplicación anterior no resultaba completamente efectiva. Se ejecutaba sobre un kiosco con una pantalla interactiva donde los pasajeros confirmaban su llegada con número de ticket, comprobaban el estado de sus equipajes, etc.

Su diseño inicial nunca tuvo en mente características como: el flujo real del proceso que necesitaba la aplicación o patrones de diseño involucrados. El objetivo fue realizar una interfaz más consistente, fácil de traducir y más usable y amigable para los pasajeros de American Airlines.

3.6.1. MENSAJES Y TRADUCCIONES

En la primera etapa, se abordó rediseñar la interfaz teniendo en cuenta la internacionalización. Durante la internacionalización del proyecto se idearon textos instruccionales e indicativos en la parte superior de la pantalla. La usabilidad fue probada en varias iteraciones en una atmósfera real como un aeropuerto.

Reducir la cantidad de texto y su complejidad en el diseño ayudo a la traducción sencilla a otros idiomas. Los pasajeros demostraron una mayor interacción con la interfaz y durante un tiempo menor. De forma adicional, para idiomas específicos se añadieron textos en contexto para el idioma nativo que permitían aclarar elementos especiales.

3.6.2. PANTALLA DE INICIO

En la segunda etapa, se valoró realizar un diseño de la pantalla de inicio. La pantalla de inicio presentaba inconsistencias en el tamaño de los botones y representación de iconos.

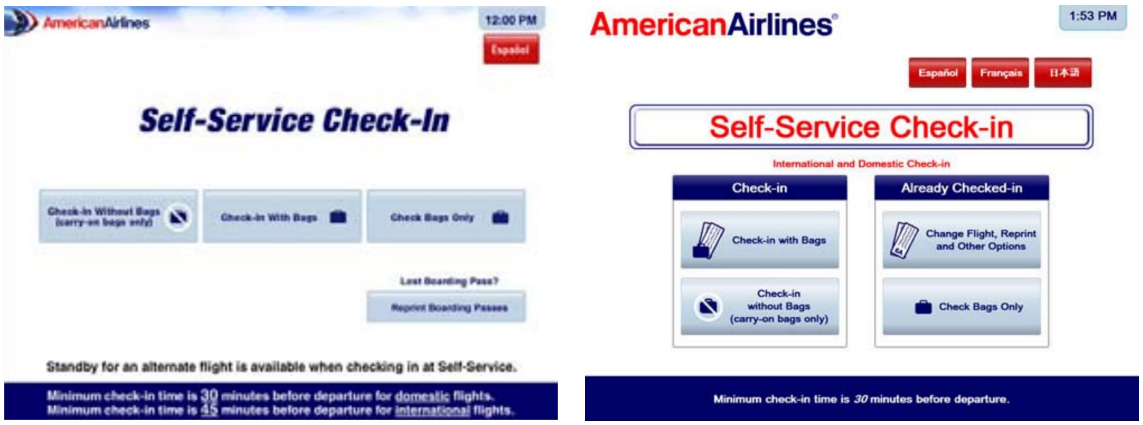


Figura 3.11: American Airlines: Pantalla de bienvenida original (a) y mejorada (b)

Su agrupación no seguía un foco lógico para el usuario identificándolos con botones extraños o poco utilizables. Esto confundía especialmente a nuevos usuarios de la aplicación que desconocían complementamente su funcionamiento.

Otro cambio en este apartado fue el cambio de un diseño con fondo de elementos distractorios a uno simplificado en tonos suaves y uniformes para toda la aplicación.

En la figura 3.11 se puede apreciar el cambio entre la pantalla de inicio original y su posterior rediseño con una interfaz más definida.

3.6.3. COMPROBACIÓN DE EQUIPAJES

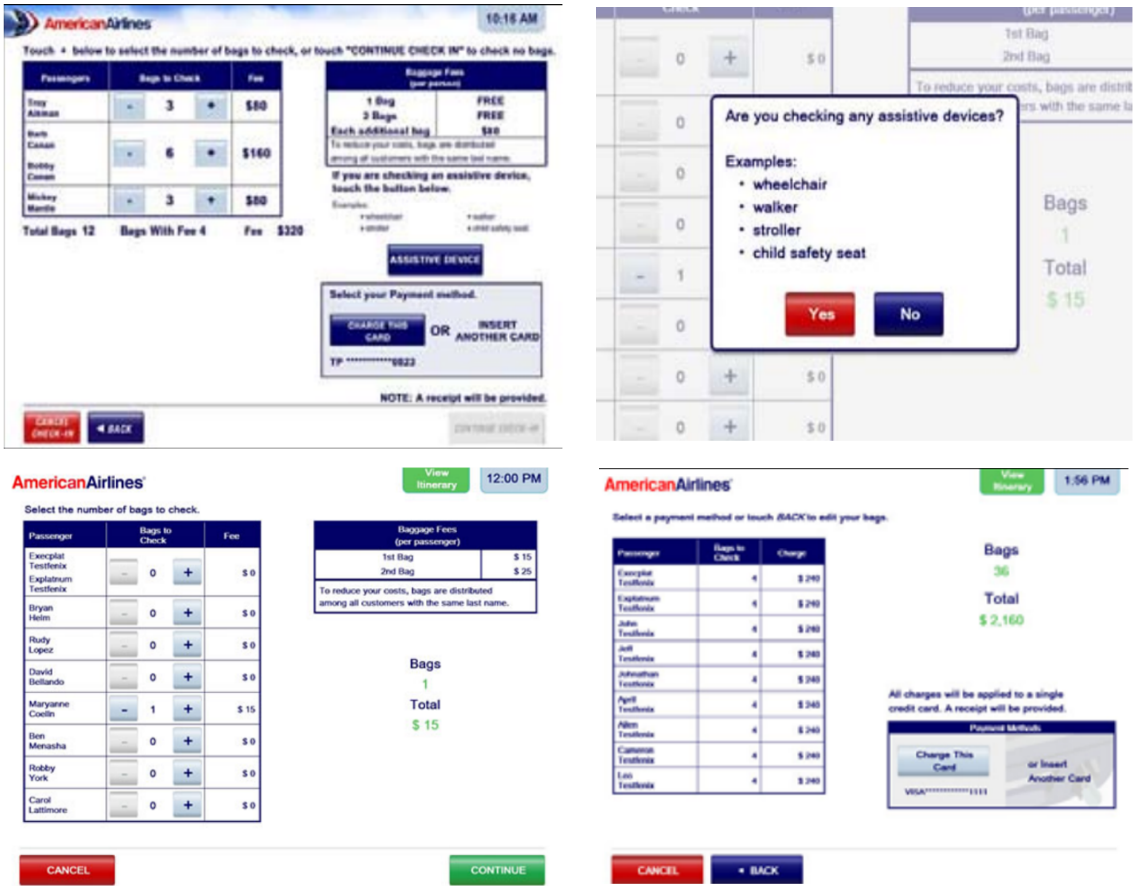


Figura 3.12: Pantalla de equipajes original (superior izquierda) y división en mejoradas (resto)

Durante la flexibilización del rediseño, se experimento con varias mejoras en el campo de usabilidad en la sección de comprobación de equipajes. La pantalla original contenía tres puntos separados de decisión.

Las buenas prácticas de desarrollo de interfaces recomiendan no introducir elementos ambiguos en las decisiones de una pantalla. Por tanto se eliminó la opción de pagos y se modificó de forma consistente con otras pantallas de pago.

La sección de dispositivos de ayuda se construyo con un mensaje (popup) emergente accionado sólo por un criterio específico que definió las reglas de la compañía. Para las transacciones monetarias se cambio al color verde en el cuerpo de la pantalla.

3.6.4. OPCIONES DE ITINERARIO

En el caso de los itinerarios de usuario la pantalla presentada era una de las más complejas de la aplicación. En ella residía la mayoría de características del kiosco interactivo.

Se eliminó el banner superior establecido de fondo en la pantalla en consecuencia con lo establecido en la pantalla de inicio. El usuario se dirigió a un itinerario de ventanas emergentes y de pantallas seguidas en consecuencia de las opciones elegidas. Aunque este cambio conllevó un mayor número de pantallas y espacio, el usuario fácilmente identificaba cada característica con una nueva pantalla.

La sección de opciones del lado derecho, tenían unos colores en consistencia al resto de la aplicación. En el estado de selección los botones fueron oscurecidos. Se añadió una barra de desplazamiento para que todos los elementos pudieran encajar en la pantalla mediante deslizamiento. En el pasado era necesario cambiar entre pantallas para ver el contenido.

Con el soporte del cliente interno y la retroalimentación del departamento de IT la compañia American Airlines consiguió una grandiosa mejora en el aspecto de usabilidad del kiosco interactivo, mejora de consistencia, incremento de espacios en blanco, robustez, internacionalización con múltiples idiomas, reducción del número de clicks para realizar tareas y ajuste del flujo de la aplicación.

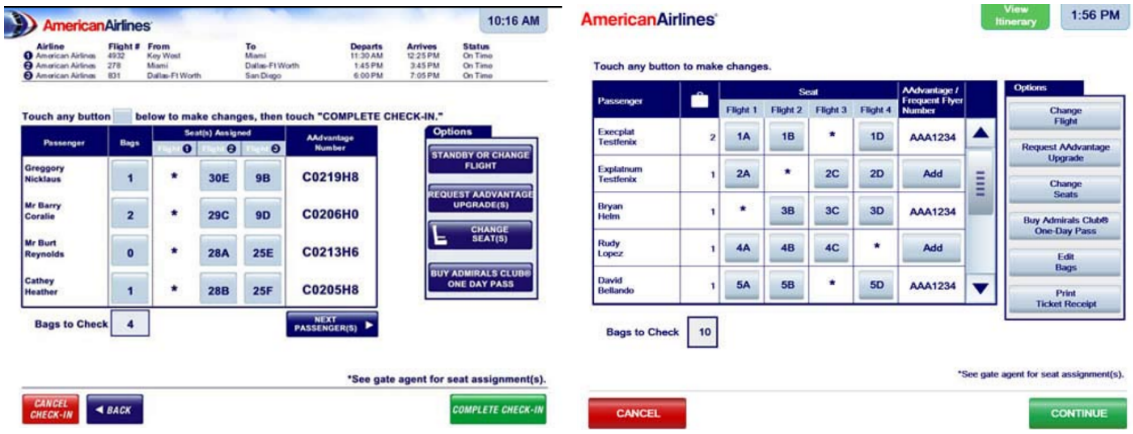


Figura 3.13: Opciones de itinerario original (a) y mejorado (b)

En conclusión, el estudio meditado de una interfaz es una tarea significante el el desarrollo de una aplicación. En FreeStation se ha facilitado la construcción y adaptación de interfaces dinámicamente, de modo que la vista ofrecida al usuario pueda ser fácilmente modificada, minimizando los enormes costes relativos a la adaptación de interfaces en POIs.

Capítulo 4

4 MÉTODO DE TRABAJO

En este capítulo se describe la metodología utilizada durante el desarrollo de FreeStation. En el Capítulo 6 se describen las iteraciones de este desarrollo, junto a la complejidad y resultados de cada una.

4.1. METODOLOGÍA DE TRABAJO Y DESARROLLO

El marco usado para estructurar y planificar el proceso del desarrollo del software distribuido seguirá las herramientas y modelos establecidos en la metodología *Extreme Programming*[Bec01] donde dicho enfoque permitirá de forma ágil experimentar una adaptabilidad sobre los requisitos naturales del sistema.

Cualquier cambio necesario en algún punto de vida del proyecto será más realista y controlable con esta metodología basada en paradigmas de simplicidad (refactorización de código iterativa, con autodocumentación del código), comunicación (pruebas unitarias de la funcionalidad), retroalimentación (valoración de resultados de salida en cada iteración).

Puesto que la arquitectura cumple con una estructura modular e incremental, el trabajo pudo ser dividido en iteraciones, valorando el resultado final en cada entrega.

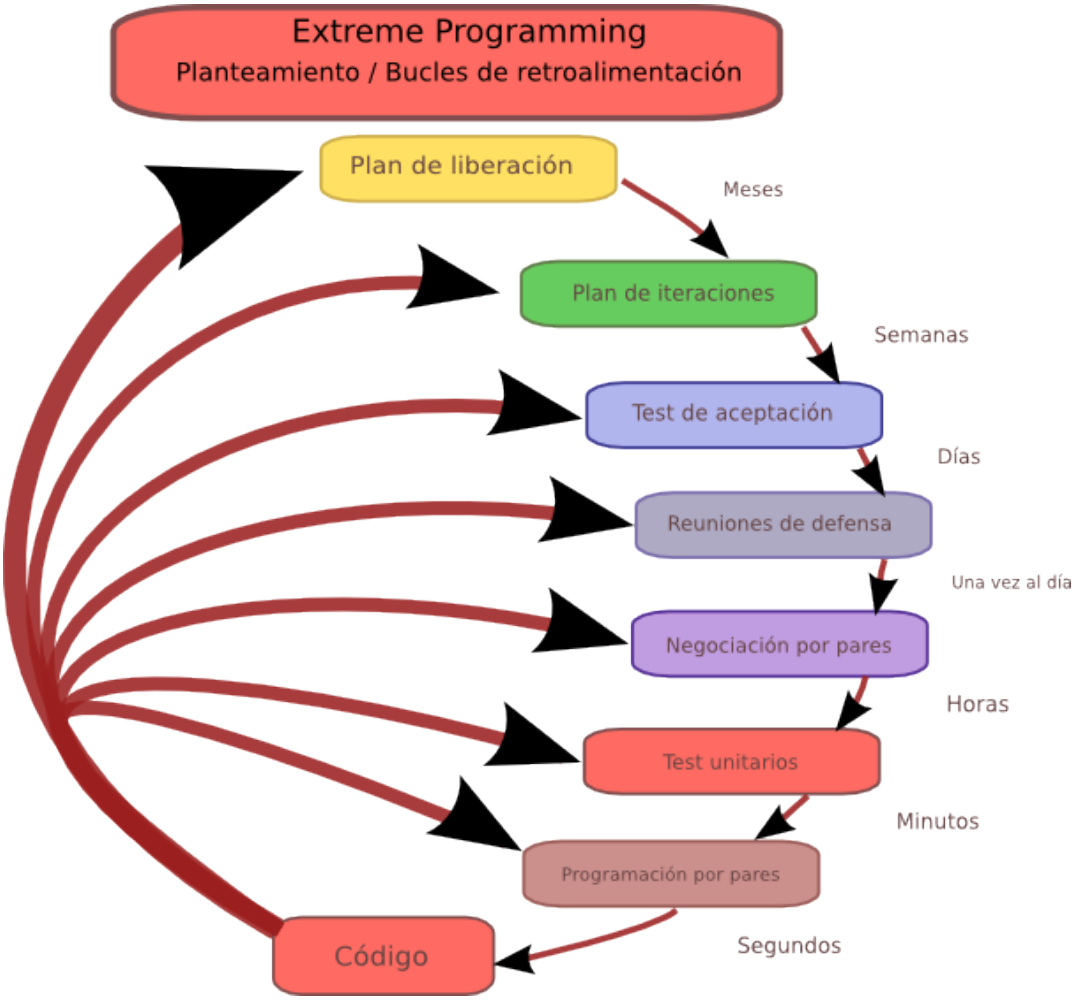


Figura 4.1: Ciclo de Extreme Programming

Este enfoque permite una mayor adaptabilidad con principales valores como comunicación, sencillez y retroalimentación[Bec00].

En el proceso se realizarán pruebas, que permitirá afianzar el trabajo y tener una mejor respuesta a introducción de fallos inadvertidos.

4.2. REQUISITOS FUNCIONALES

Para establece unos requisitos funcionales, por un lado es necesario efectuar un estudio previo de la situación actual. Esto implica el análisis de otros sistemas con funcionalidades similares, buscar ventajas e inconvenientes de cada uno de ellos y determinar a partir de ahí las funcionalidades deseables en nuestro sistema.

Por otro lado: definir un objetivo fijo, establecer los límites de definición del proyecto y evaluar y definir las funcionalidades del sistema.

Los usuarios finales demandan una atención al tiempo de respuesta final del proceso solicitado. La actualización del software obtenido en una red de distribución debe minimizarse al máximo[Bau10] y requerir de una atención mínima por sus supervisores para una gran red.

Esto requiere a su vez de una interfaz modularizable por la parte final de los usuarios, donde con simples acciones e intuitivas se permita realizar las acciones requeridas.

4.3. REQUISITOS ESTRUCTURALES

El sistema debe contar con una estructura muy tolerante a fallos, donde un simple error no comprometa al sistema e incluso se recupere del mismo si fuera posible. En caso de no ser posible, este al menos debería contar con un buen informe para los supervisores de la red, para que en cualquier momento un problema sea identificado y reparado en el menor tiempo posible.

La demanda de software por los usuarios, debe ser realizada en imágenes (ISO) personalizadas y parametrizadas para cada subsistema en un centro de software distribuido. Estas imágenes deben ser regeneradas y actualizadas cuando el sistema lo requiera o sus supervisores lo necesiten.

4.4. CASOS DE PRUEBA

La gestión de calidad de software dependerá de las transformaciones sufridas en su distribución. La seguridad y conformidad del despliegue software permitirán una alta solución al problema donde los requisitos sea cumplidos sin ningún defecto.

Al encontrarse el sistema automatizado, el escenario de uso común de un usuario estará plenamente abordado por el sistema permitiendo distribuir el software ordenador por el usuario final sin ninguna complicación.

El rendimiento y la escalabilidad del sistema permitirán una amplia gama de aplicaciones disponible convirtiendo al sistema en un catálogo completo personalizado con la precisión establecida.

Partiendo de un conjunto de funcionalidades en forma de requisitos es posible especificar un diseño global del sistema. Este define su estructura, componentes, módulos y datos del sistema.

4.5. MEDIOS UTILIZADOS

Los medios requeridos para el desarrollo tienen en cuenta la utilización de la infraestructura de repositorios, servidores, clientes, etc, que serán usados en la red de la Universidad de Castilla-La Mancha (UCLM) en colaboración con los equipos de grupo de investigación ORETO[1].

Se utilizara repositorios de control de versiones GIT[2] para la administración del desarrollo del código utilizado.

El sistema cliente y servidores funcionarán con sockets escritos mayormente en python, donde la comunicación mediante datos crudos, datos estandarizados (XML) o invocaciones remotas, definirán actuaciones del sistema y sus posibles consecuencias. Posteriormente se utilizarán frameworks de objetos distribuidos para envío y recepción de datos remotos.

La supervisión y monitorización[Mar11] sera realizada tanto de forma interna por el sistemas como con la ayuda de aplicaciones terceras que informen a este de posibles errores en el mismo o elementos de aviso para sus administradores.

La documentación general del sistema será especificada y parametrizada para clientes y servidores con el fin de obtener una gran modularidad y adaptabilidad del sistema ante requisitos que puedan surgir en la posible vida del proyecto en un centro de distribución software.

[1] ORETO: `http://oreto.esi.uclm.es`
[2] GIT: `http://git-scm.com`

4.5.1. Herramientas

A continuación se enumeran y describen los recursos hardware y software utilizados durante el desarrollo, indicando las versiones específicas empleadas en la construcción de la versión de FreeStation adjunta en el CD que acompaña a la presente memoria.

El software y hardware durante el desarrollo ha seguido dichas especificaciones por lo que se recomienda realizar una configuración similar con el fin de obtener los mismos resultados. Existe la posibilidad de utilizar versiones superiores de software u otros componentes hardware.

No se aseguran la exactitud de los mismos resultados de ejecución, aunque el software ha sido desarrollado de forma adaptativa para tratar de minimizar el impacto de errores o fallos derivados de distintas configuraciones.

4.5.2. Lenguajes

Para la realización de FreeStation se han utilizado diversos lenguajes de programación, que se listan a continuación:

- **PHP**[3]: es el lenguaje utilizado principalmente en el desarrollo del proyecto para la parte Frontend del modelo servidor web. El proyecto ha sido implementado usando una versión compatible con PHP 5.3.X.

  PHP es un lenguaje creado en 1994 por Rasmus Lerdorf para entornos de binarios CGI.

  Esta escrito en el lenguaje C y gracias a su simplicidad creció rápidamente en popularidad hasta convertirse en el lenguaje con mayor cuota de mercado de creación para webs.

- **Python**[4]: es el lenguaje utilizado para realizar scripting y la parte Backend del modelo servidor y modelo cliente. La versión adjunta en el CD está compilada con soporte para Python 2.7.

  Python es un lenguaje creado en 1989 por Guido van Rossum con el principal objetivo de realizar programas legibles y de sintaxis clara. Es un lenguaje para prototipado rápido y aplicaciones críticas con mucho potencial matemático.

[3] PHP: `http://php.net`
[4] Python: `python.org`

4.5.3. Hardware

La mayor parte del desarrollo de FreeStation se ha realizado usando varios nodos cliente y servidor.

Las especificaciones de hardware no son elevadas para uso de nodos clientes, aunque la especificación para un nodo servidor, viene determinada por el número de clientes que se conecten y el tráfico simultáneo en pico que deba soportar el nodo servidor.

Las características más notorias para la aplicación son la concurrencia de CPUs y la capacidad de memoria. La capacidad de disco duro, operaciones de entrada/salida u otros factores son importantes, pero quedan establecidos según configuraciones particulares que se demanden.

Para los nodos clientes se han usado 4 portátiles y un ordenador de sobremesa:

- Acer Aspire Ethos 5943G (laptop): 4 GB RAM, 4 núcleos.
- Acer Aspire One (netbook): 2 GB RAM, 2 núcleos.
- Compaq Presario C700: 1 GB RAM, 1 núcleo.
- HP 620: 4 GB RAM, 2 núcleos.
- Sobremesa: 4 GB RAM, 4 núcleos.

Para el nodo servidores se han usado varias máquinas virtualizadas:

- Bajo virtualización OpenVZ[5]: 512 MB ram, 2 núcleos, para entorno de desarrollo y pruebas y de bajas características durante el desarrollo.
- Bajo virtualización VMWare[6]: como entorno de producción bajo de 4 cores y 8 GB RAM.

[5] OpenVZ: `http://openvz.org`
[6] VMWare: `http://vmware.com`

4.5.4. Software

A continuación se lista todas las herramientas y bibliotecas software utilizadas para el proyecto. Estas herramientas justifican la importancia crucial para un ámbito de desarrollo, con el fin de ofrecer unas condiciones similares para este proyecto de fin de carrera. Se describen por subapartados y tecnologías específicas.

4.5.4.1. Sistemas operativos

Las preferencias se basaron en sistemas operativos basados en software libre y de amplia distribución.

Se han usado los siguientes:

- **Ubuntu**[7]: como sistema operativo principal se ha utilizado Ubuntu 11.10, 12.04 y 12.10 durante el desarrollo, siendo imprescindible ubuntu 12.10 para la versión cliente. Se ha elegido por considerarse una de las distribuciones de GNU/Linux más usadas y sencillas para desarrolladores y estar basada en Debian. La totalidad de aplicaciones de este proyecto han sido desarrollado bajo este sistema operativo.
- **CentOS**[8]: como sistema operativo para nodos servidores, basada en RedHat.

  La versión utilizada ha sido la rama 5, con actualizaciones hasta la versión 5.8. La elección fue por la estabilidad y conocimiento previo de la distribución para servidores.

[7] Ubuntu: `http://ubuntu.com`
[8] CentOS: `http://centos.org`

4.5.4.2. Entornos de desarrollo

La totalidad del proyecto ha sido desarrollada con **IDE!**s que permitieran un rápido desarrollo. Los más utilizados han sido:

- **Eclipse**[9]: se utilizó la plataforma de desarrollo Eclipse 3.7 y 3.8, junto al plugin desarrollo para PHP (PDT, PHP Development Tools v 3.0) para el frontend web en PHP, el plugin PyDev v2.2 para el backend en Python y desarrollo en ICE, y los plugins ShellEd 2.0 y TeXlipse 1.5 para scripts y documentación en LaTeX.
- **Geany**[10]: es un editor de texto que usa GTK con las características básicas para un entorno de desarrollo. Fue utilizado para edición de archivos simples y configuraciones de forma rápida al ser un editor rápido y ligero basado en un motor escrito en C.

  La versión utilizada fue la 1.22 con el nombre en código "Tavira".

4.5.4.3. Entorno de depuración y perfilado

Para desarrollar un software depurado, que evite en la mayor medida fallos y cuellos de botella, se utilizaran herramientas de depurado y perfilado como las siguientes:

- **Xdebug**[11]: es el depurador y perfilador por excelencia de PHP escrito por Derick Rethans. Tiene una gran variedad de parámetros de configuración, depuración remota, integración con muchos IDEs (Eclipse, Netbeans, etc)
- **KCacheGrind**[12]: es un programa de escritorio (en especial para diseñado para KDE, pero compatible para GNOME) escrito para visualizar los archivos de depuración generados por xdebug. Es un frontend de Callgrind[13], que a su vez usa en tiempo de ejecución al framework de Valgrind para simulación de caches y generación de las llamadas gráficas.

[9] Eclipse: `http://eclise.org`
[10] Geany: `http://geany.org`
[11] Xdebug: `http://xdebug.org`
[12] KCacheGrind: `http://kcachegrind.sourceforge.net/html/Home.html`
[13] Callgrind: `http://valgrind.org/docs/manual/cl-manual.html`

- **Webgrind**[14]: es un perfilador gráfico de código PHP vía web escrito en PHP por Joakim Nygård y Jacob Oettinger. Tiene como dependencia Xdebug ya que es en realidad un wrapper web no completo de las funciones de KCacheGrind como reemplazo de frontend web de xdebug.

  Se ha utilizado para depurar código, optimización de cuellos de botella y mejoras de rendimiento.

4.5.4.4. Entorno de pruebas unitarias

- **PHPUnit**: es un entorno para realizar pruebas unitarias en el lenguaje de programación PHP creado por Sebastian Bergmann. Se ha utilizado para detectar los errores en el código durante el desarrollo y evitar la introducción de nuevos en partes probadas.
- **PyUnit**[15]**/unittest**[16] : es un estándar probado como framework simple y elegante de pruebas unitarias en programas Python (módulo unittest).

  Se ha utilizado para realizar una pequeña batería de pruebas unitarias en algunas de las clases y código más crítico del proyecto, en particular la parte backend de cliente y servidor escrita en Python.

4.5.4.5. OTRAS APLICACIONES

- **MySQL**[17]: es el sistema de gestión de base de datos relacional (RDBMS) más usado en el mundo que es ejecutado en modo servidor proporcionando un acceso multi-usuario a un gran número de bases de datos.

  Se ha utilizado como servidor de base de datos en la capa de persistencia de backend y frontend servidor.

  La versión utilizada ha sido MySQL 5.1.

[14]Webgrind: `https://code.google.com/p/webgrind/`
[15]PyUnit: `http://pyunit.sourceforge.net`
[16]Unittest`http://docs.python.org/library/unittest.html`
[17]Mysql: `https://mysql.com`

4.5.4.6. DOCUMENTACIÓN Y GRÁFICOS

- **Make**[18]: Herramienta que se ha utilizado para el proceso de compilación de los archivos fuente del proyecto (en mayoría fuentes relacionadas con la documentación, ya que archivos fuente en php y python no requerían de esta utilidad). La versión que se ha utilizado es la 3.81.

- **Bash**[19]: para creación de scripts de compilación, generación de documentación y otras utilidades relacionadas en el proyecto. La versión utilizada ha sido 4.2.10.

- **LaTeX**[20]: el framework de composición de documentos libres y profesionales que se ha usado para generar este proyecto de fin de carrera. Se ha utilizado la versión 3.1415926-1.40.10-2.2 (TeX Live 2009).

  El código fuente del proyecto y documentación están disponibles en fuentes para este programa bajo licencia FDL.

- **LibreOffice Draw**[21]: herramienta de dibujo vectorial utilizada para la generación de figuras e imágenes para la documentación, en su versión 3.4.

- **LibreOffice Calc**[22]: herramienta de creación y manipulación de hojas cálculo, utilizada para la generación de gráficos estadísticos de la documentación. Se ha aplicado la versión 3.4.

- **The Gimp**[23]: es una potente herramienta de manipulación de gráficos de software libre utilizada para la documentación y generación de gráficos para demostraciones.

  Se ha utilizado la versión 2.6.11.

- **Inkscape**[24]: es un editor de Gráficos Vectoriales, similar al Adobe Illustrator, que busca ser compatible con SVG, de código abierto, sensible y extendible.

  Se ha usado para la creación de diagramas y vectorizado de aplicaciones.

- **MySQL Worbench**[25]:MySQL Workbench es una herramienta visual unificada para arquitecturas de bases de datos y desarrolladores.

  Permite un modelo de datos basado en el desarrollo SQL de datos en crudo y un conjunto de herramientas comprehensivas para la administración.

[18] make: `https://gnu.org/software/make/`
[19] bash: `https://www.gnu.org/software/bash/manual/bashref.html`
[20] LaTeX: `http://latex-project.org`
[21] LibreOffice Draw: `http://es.libreoffice.org/caracteristicas/draw/`
[22] LibreOffice Calc: `http://es.libreoffice.org/caracteristicas/calc/`
[23] The Gimp: `http://gimp.org`
[24] Inkscape: `http://inkscape.org`
[25] MySQL Worbench: `www.mysql.com/products/workbench/`

Se utilizo la versión 5.1 para el modelado del diagrama E/R del esquema de base de datos utilizado en la capa de persistencia de mysql.

- **Gnome Dia**[26]: es una herramienta de dibujo para diagramas y casos de uso.

  Genera todo tipo de diagramas, se ha utilizado para desarrollar el esquemas de clases.

  Se ha utilizado la versión 0.97.1.

4.5.5. BIBLIOTECAS, PLATAFORMAS Y FRAMEWORKS

4.5.5.1. GTK

GTK+ o The GIMP Toolkit[27]: es un conjunto de bibliotecas multiplataforma para desarrollar interfaces gráficas de usuario (GUI).

Las principales ventajas de GTK+ son:

- Activamente desarrollado y mantenido con una comunidad fuerte.
- Ofrece extender la funcionalidad básica de widgets.
- Soporte de framework para internalización y accesibilidad.
- Software libre.
- Multiplataforma y portable.

La biblioteca que se ha utilizado para la creación y la gestión de las interfaces gráficas del proyecto.

En particular se ha usado su binding para Python, llamado PyGTK y que en el desarrollo derivo a la versión de PyObject

[26] Gnome Dia: `https://live.gnome.org/Dia`
[27] GTK: `http://gtk.org`

4.5.5.2. ZeroC ICE

ZeroC ICE (ZeroC Internet Communications Engine) es un middleware muy adecuado para desplegar y desarrollar aplicaciones de sistemas distribuidos.

El objetivo de éste es hacer que la invocación a objetos remotos sea vista por parte del programador como si se tratase de una invocación a objetos locales.

La forma de llevar a cabo esta enmascaración es mediante un proxy en la parte del cliente que actúe de intermediario entre el cliente y el objeto remoto al que se quiere invocar.

Y en la parte del servidor se necesita un esqueleto que traduzca los eventos que se van produciendo en la red a las correspondientes invocaciones de los métodos remotos.

Esta traducción es necesaria debido a que la comunicación entre el cliente y servidor, no deja de ser una comunicación que sigue un protocolo de red como TCP, UDP, etc.

Debido a que ZeroC ICE permite que el cliente y el servidor estén implementados en distintos lenguajes de programación. El lenguaje para definir la interfaz que debe de ser idéntica a ambas partes se define mediante Slice: lenguaje de especificación de interfaces para ZeroC ICE.

Las razones principales para usar ICE fueron:

- Es una buena herramienta de trabajo para entornos que sean bastante heterogéneos, donde el cliente y servidor puede estar escritos en lenguajes de programación muy diferentes y ejecutarse sobre plataformas, sistemas operativos o arquitecturas muy variadas.
- Su fuerte radica en un gran soporte de tecnologías de red con interacciones a bajo nivel de red, enfoque en la lógica de las aplicaciones y portabilidad de entornos muy diversos.
- La abstracción de red para detalles como abrir conexiones de red, serializado y deserializado de datos transmitidos por red, reintentos fallidos de conexión y muchos otros detalles más de bajo nivel que ayudan a simplificar la capa de red[Fou05].

- Soporta bindings con bastantes lenguajes de programación como: C++, .NET, Java, Python[Val06], Objective-C, Ruby, PHP, y ActionScript.
- Integra en su propia implementación seguridad, que hace sencillo de usarla en redes públicas inseguras.
- La curva de aprendizaje[Hen06] es sencilla y existe una gran infraestructura de aplicaciones técnicas que se ha desarrollado.
- Ice es software libre bajo licencia GPLv2 y comercializado bajo la marca ZeroC[28].

Por lo tanto, se consigue evitar complejidad innecesaria, haciendo la plataforma fácil de usar y aprender. La implementación es eficiente para red respecto a ancho de banda, uso de memoria y CPU.

4.5.5.3. DBus

DBus[29] es una implementación de alto nivel de IPC similar a Corba o ICE para las aplicaciones de Gnome. Se base en un sistema de mensajes enviados a un bus principal que permite a las aplicaciones enviar mensajes para interactuar entre ellas.

Se ha utilizado por su integración con Gnome y eficiencia de implementación con GNU/-Linux probada en aplicaciones reales por un gran período de tiempo.

La principal ventaja es su utilidad para la detección de Hardware, como nuevos dispositivos USB.

Su binding a python facilita la programación orientada a eventos y señales a través de la intercepción de mensajes en el bus principal.

[28] ZeroC: `http://zeroc.com/`
[29] Dbus:`http://www.freedesktop.org/wiki/Software/dbus`

4.5.5.4. Gstreamer

GStreamer[30] es una biblioteca para construcción de gráficos o interacción de componentes multimedia.

Soporta un gran rango para de componentes como reproducción de vídeo OGG/Vorbis, streaming de audio/vídeo con efectos, editado y procesado de vídeo no linear, etc.

Tiene una gran facilidad ofrecida por esta biblioteca es la capa de abstracción a la hora utilizar códecs o formatos. La transparencia al programador permite un desarrollo rápido enfocado al propósito real de la aplicación sin perder eficiencia o versatilidad.

Es compatible con un gran rango de sistemas operativos, procesadores y compiladores. Su desarrollo es muy activo y esta respaldado por grandes empresas como Collabora[31], Fluendo[32] y la práctica totalidad de las distribuciones GNU/Linux lo incluyen en sus repositorios o de forma nativa en el sistema.

El traspaso de datos es extremadamente liviano lo que permite repercutir en una baja latencia y alto rendimiento.

Como característica adicional, posibilita la carga dinámica de complementos, lo que hace posible la personalización de características deseadas en cualquier momento.

Durante el desarrollo se uso la versión 0.10, pero posteriormente se cambio a 0.11 (conocida como 1.0) por problemas de integración en el binding de python. Ver iteraciones del capítulo 6 para mayor detalle.

[30] GStreamer:`http://gstreamer.freedesktop.org/`
[31] Collabora:`http://collabora.com`
[32] Fluendo: `http://fluendo.com`

Capítulo 5

5 ARQUITECTURA

5.1. INTRODUCCIÓN

Figura 5.1: Logo de Free Station

FreeStation o «Librenería» es un software para centros o puntos de acceso de distribución de información de software libre orientado a centros de enseñanza y universidades.

El sistema esta compuesto por un armario de diseño con un ordenador empotrado con pantalla táctil y una cámara para reconocer movimientos y gestos. En dicho dispositivo se encuentra un software cliente que alberga repositorios de software modularizables configurados por el centro de enseñanza o universidad, con software como imágenes de distribuciones GNU/Linux, software específico para los alumnos o aplicaciones educativas para interactuar con el dispositivo.

El dispositivo puede incorporar una unidad de CD-ROM y USB con la que el usuario/-docente puede grabar los datos proporcionados por el sistema en pocos segundos y hacerse una copia de todo software gratuito y opensource disponible.

Además el sistema cuenta con otra aplicación servidor, que distribuye diariamente a todos los clientes el software y los módulos actualizados.

5.2. DESCRIPCIÓN GENERAL

La arquitectura de FreeStation está principalmente basada en el modelo cliente-servidor. Cada componente es modularizable y fácilmente personalizable. Su diseño esta pensado para no requerir especiales conocimientos técnicos para su uso y administración diarios.

FreeStation o «Librenería» es el software cliente y servidor para la organización de puntos de acceso e interés de software libre catalogado y modularizado en estaciones libres.

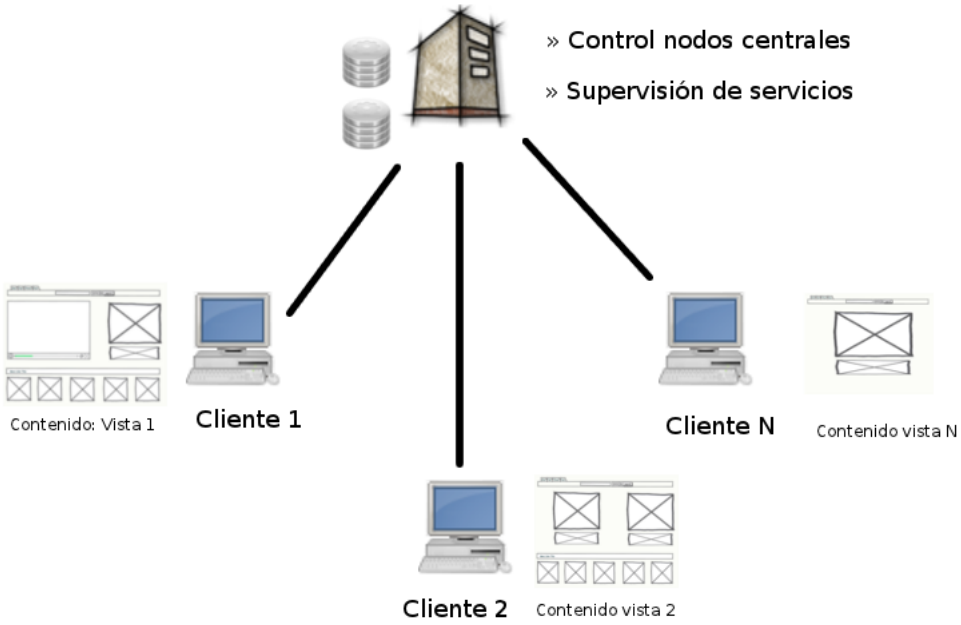


Figura 5.2: Arquitectura propuesta

La arquitectura propuesta en la figura 5.2 habilita un nodo principal servidor que dispone de métodos de interacción para la supervisión sencilla del servidor además del control de los nodos centrales restantes.

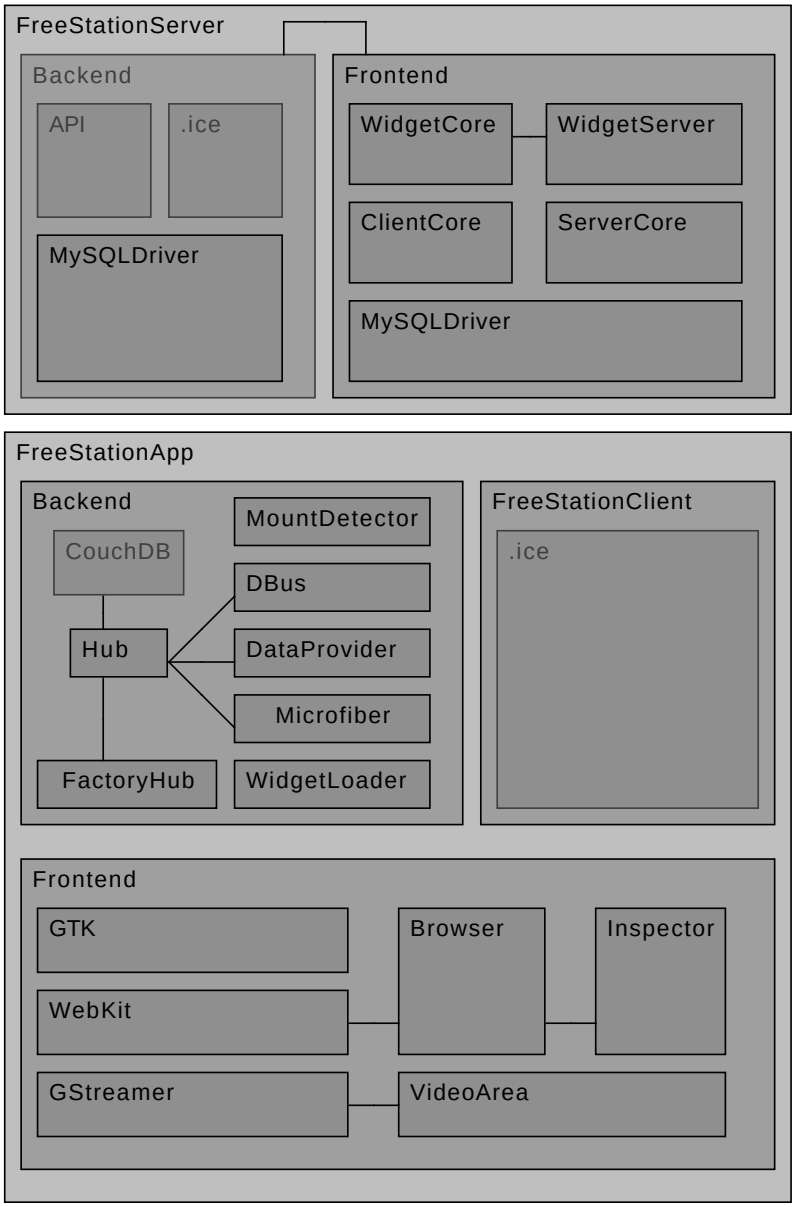


Figura 5.3: Diagrama de la arquitectura general y subsistemas

El servidor actúa como replica de la información de consulta y difunde o distribuye los contenidos en los nodos clientes solicitados en los que el usuario dispone de una interfaz para su uso.

La figura 5.3 muestra un diagrama de la arquitectura general y subsistemas que indican los módulos que lo componen de forma general. En posteriores secciones serán detallados más profundamente.

5.3. ARQUITECTURA DEL SERVIDOR

El servidor FreeStation funciona sobre una pila de software consistente en varias tecnologías y componentes. El servidor es el encargado de administrar la comunicación con cada cliente.

5.3.1. Descripción general

El servidor dispone de un un panel de interfaz (GUI) basado en PHP y Javascript.

Figura 5.4: FreeStation Server Webserver GUI

Como se puede observar en B.8, existen funciones de *iniciar/apagar/reiniciar* el servidor desde la interfaz y ver el estado real del mismo con un listado de registro de salidas de errores útiles. Mediante estos registros es posible analizar y diagnosticar cualquier actividad o evento en el servidor o errores que se produzcan.

Asimismo, permite fácilmente la administración de tareas sencillas para administrar y configurar tareas comunes para un número elevado de clientes con POIs.

5.3.2. Composición de capas

Bajo la interfaz gráfica del servidor web, se ejecuta un proceso demonio que actúa como backend y que esta basado en Python.

De esta forma, cada acción en la interfaz web se traduce como una tarea interna para el backend que es despachada de forma rápida.

El backend puede realizar comunicaciones asíncronas con los clientes. Para ello, se basa en ICE (Internet Comunication Engine de ZeroIce).

Esto permite transferir grandes conjuntos de información a un gran número de clientes con un sólo servidor de una forma fiable y robusta.

Tanto frontend como backend disponen de una capa de persistencia en la que almacenar los datos no volátiles de operaciones. En esta capa, se encuentra un servidor de base de datos mysql, que es el encargado de almacenar de forma persistente los datos (Mysql Storage).

De esta forma los datos pueden ser escritos de forma concurrente por el backend o el frontend sin generar inconsistencias en los datos.

Los datos accedidos por el backend son un conjunto limitado a datos internos para operar con clientes, mientras que los datos accedidos por el frontend se limitan a la configuración básica de administración.

Toda configuración que sea necesaria consultar con posterioridad por el servidor es almacenada en la capa persistente.

Subconjuntos de esta configuración es transmitidas a los clientes que la requieran para desplegar sus ejecuciones.

Por ejemplo, es posible visualizar un conjunto de los clientes actuales manejados por el servidor y las estadísticas de los mismos:

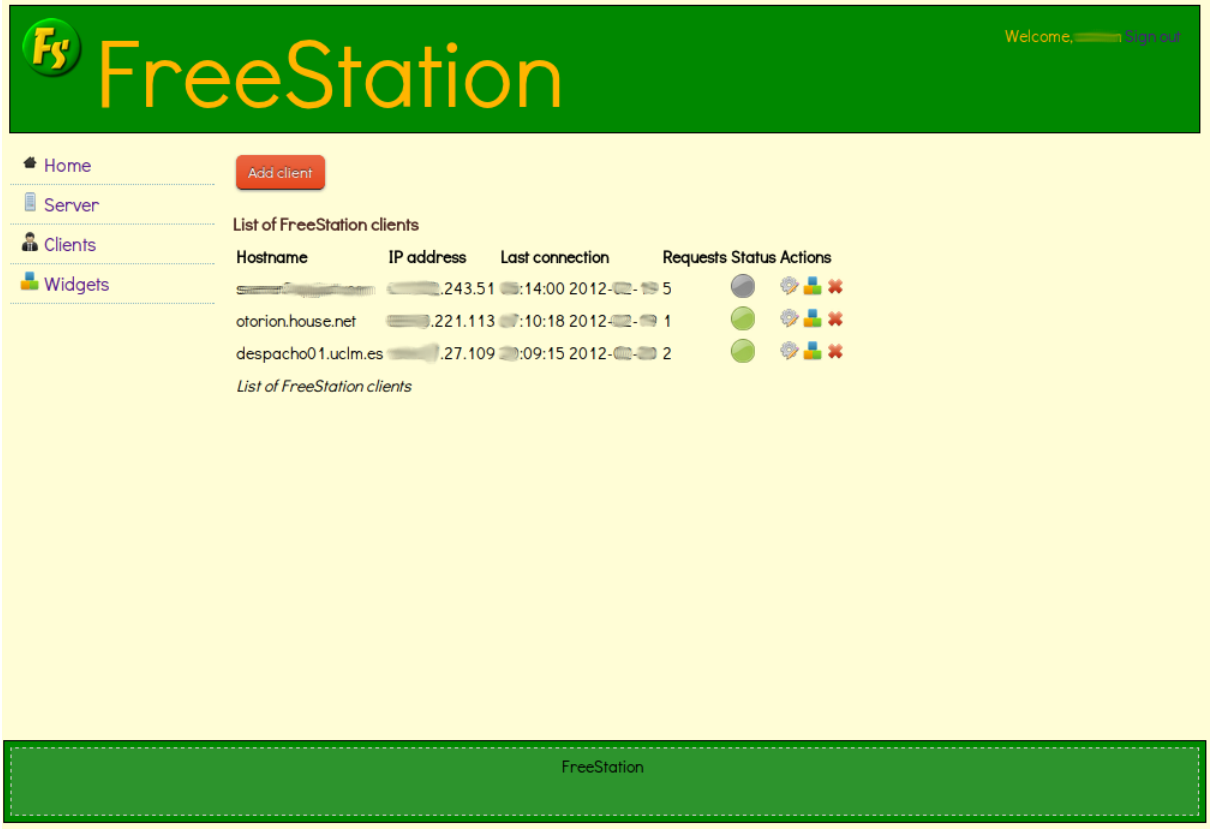


Figura 5.5: FreeStation Server GUI - Listado de clientes

Como se muestra en el listado de la figura 5.5 para cada cliente se visualizan estadísticas de:

- Hostname (el nombre de la máquina cliente)
- Direción IP en formato IPv4.
- Tiempo registrado de la última conexión realizada
- Número de peticiones realizadas (acumuladas) en total por el cliente.
- Iconos de acción para configuración

Los iconos de acción permiten añadir/eliminar/editar clientes y todos los datos derivados para ese cliente que estén vinculados serán actualizados de forma asíncrona.

5.4. ARQUITECTURA DEL CLIENTE

El modelo cliente se ejecuta normalmente sobre un ordenador de escritorio o terminal que servirá como referente para un POI (Point Of Interest).

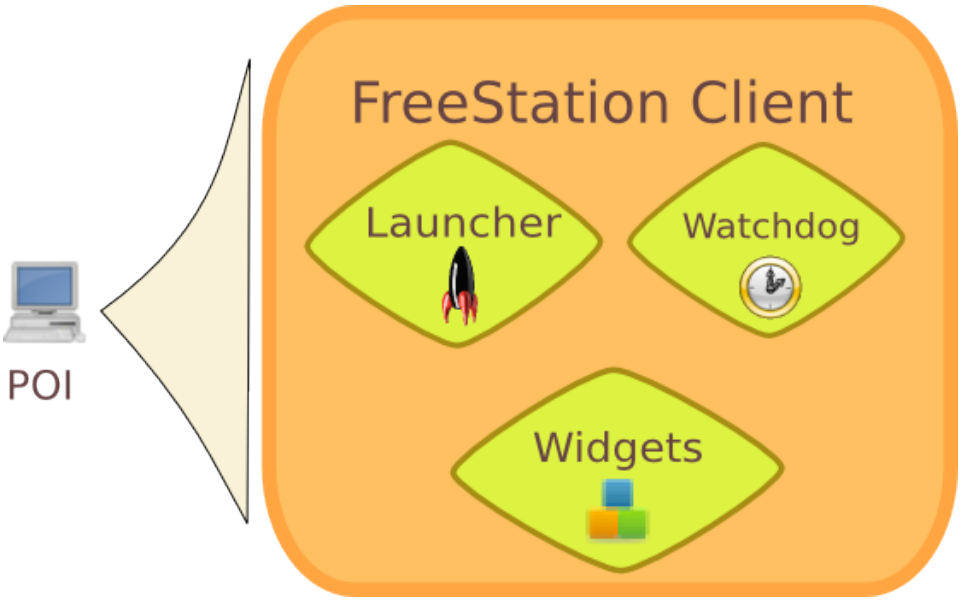


Figura 5.6: Componentes de arquitectura cliente

Para obtener la configuración general, el cliente contacta con su servidor central y el cliente recibe la configuración general y composición de widgets desde el servidor central y este carga deacuerdo a sus especificaciones.

Esta configuración sólo será recibida si el cliente esta autorizado en el servidor y si este esta habilitado en el servidor.

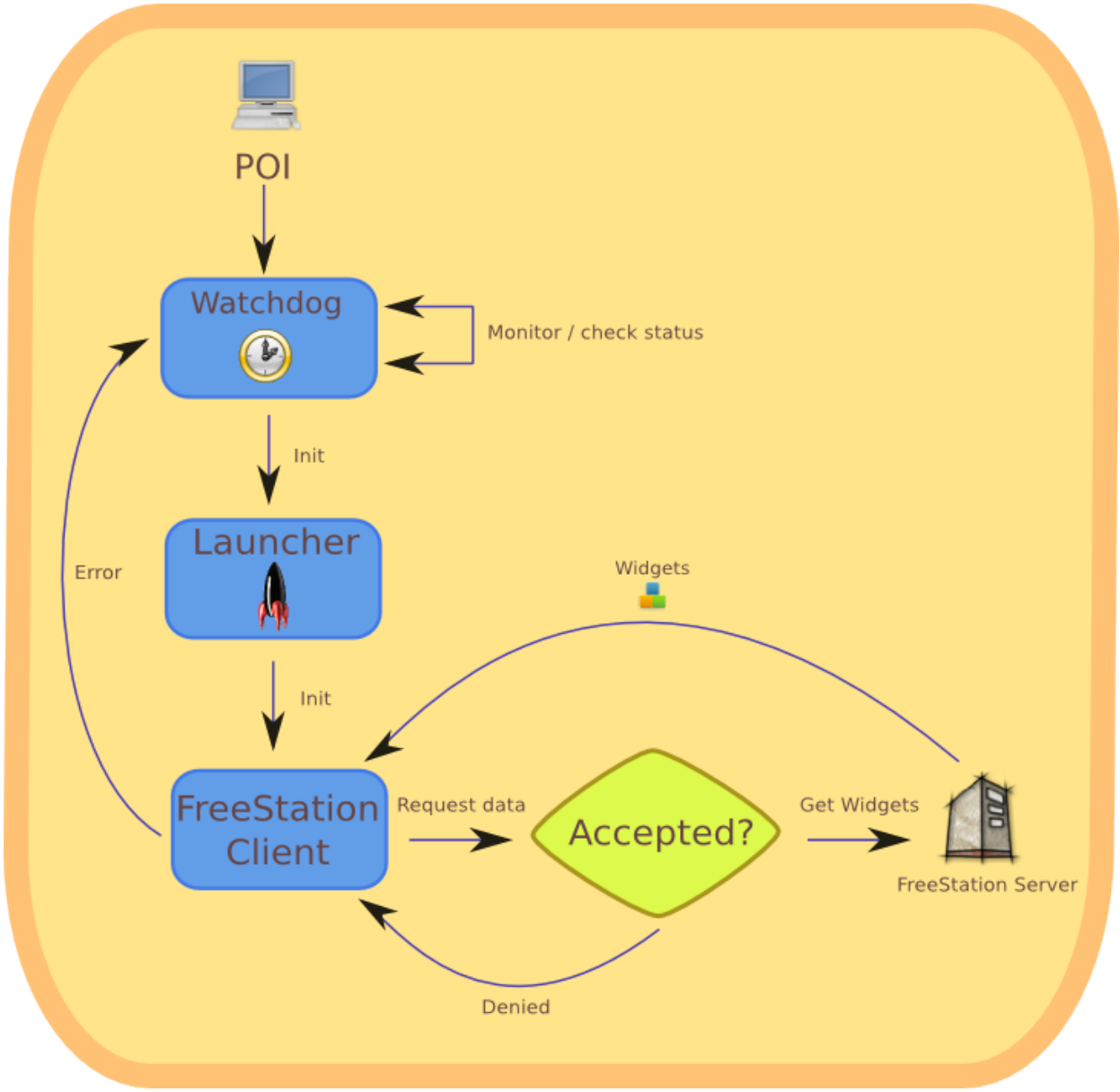


Figura 5.7: Diagrama de flujo - Cliente

En el diagrama de flujo de la figura 5.7 puede observarse dicha interacción.

El modelo cliente esta programado en el lenguaje Python, por ser un lenguaje rápido y de prototipado, normalmente muy útil para realizar interfaces de escritorio bajo la plataforma GNU/Linux.

Un cliente se ejecuta sólo en un POI y periódicamente consulta al servidor para recibir actualizaciones de contenidos o información. Los clientes necesitan las especificaciones para configurar los widgets o componentes y los datos asociados para cada widget.

Un POI debe ser accesible continuamente para los usuarios que lo requieran y por ello debe ser robusto ante fallos y minimizar todos los riesgos posibles de seguridad en acceso al mismo y sus componentes.

El cliente dispone de un programa lanzador «launcher» que inicia el programa principal y se encarga de monitorizar en intervalos de tiempo regulares mediante un programa watchdog la aplicación del modelo cliente y asegurarse de que siga completamente funcional.

En el momento que se produce algún fallo crítico o se compromete el modelo cliente, el watchdog reinicia la aplicación y establece los valores funcionales sin que se adviertan problemas para el usuario.

5.5. ARQUITECTURA DE WIDGETS

5.5.1. Definición de widget y su aplicación

Un widget en la terminología de FreeStation es un elemento abstracto que lleva a cabo operaciones atómicas sobre un dominio reducido de actuación y que provee una comunicación para manejar la información de los datos que almacena el widget.

Un widget esta representado simbólicamente con el icono de la figura 5.8.

Figura 5.8: Icono de representación para widgets

En los datos de un widget puede encontrarse la configuración del widget que será tratada por el cliente.

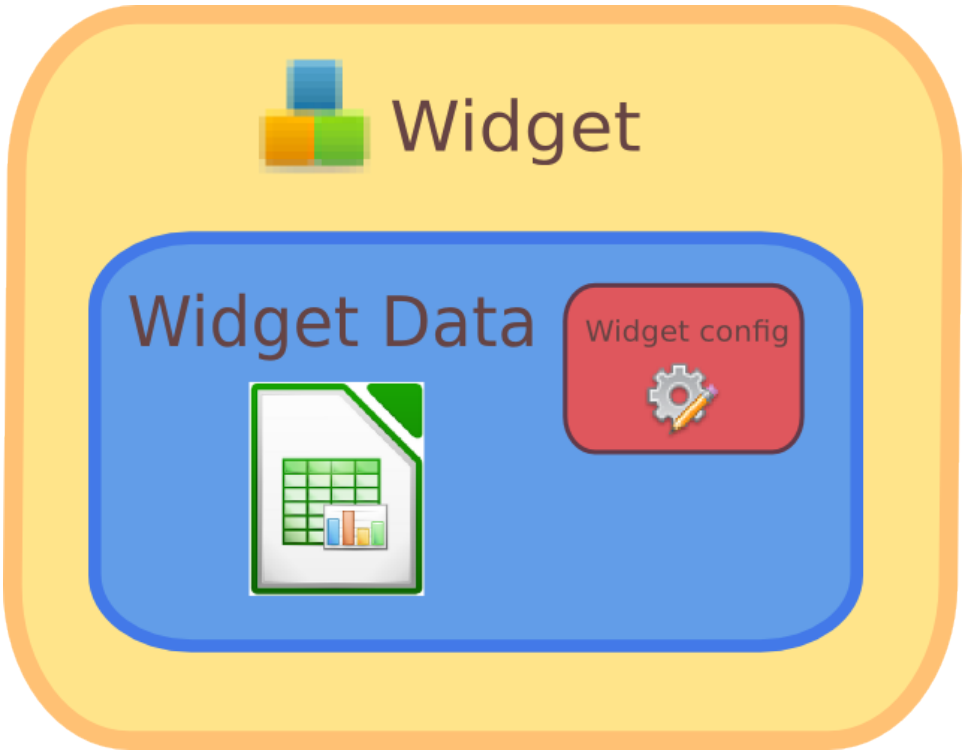


Figura 5.9: Contenedor Widget - Composición

Para operaciones complejas un widget puede asociarse con más widgets para llevar a cabo tareas sofisticadas o simplemente comunicar datos con otros widgets. Los widgets son sólo ejecutados en el cliente, pero todos ellos son únicamente configurados desde el servidor.

5.5.2. Modelo orquesta

La configuración de widgets entre cliente-servidor describe lo que se conoce como modelo orquesta[RoS04] donde un servidor actúa como dirigente/administrador (coreógrafo) de un conjunto de clientes con widgets que actúan como un coro de orquesta.

Esta situación se traduce en la restricción de que un cliente sólo puede comunicarse con el servidor y no tiene permitido comunicarse con otros clientes.

El inconveniente de este modelo arquitectural, es que puede ser fácilmente vulnerable a fallo al contar con un único punto de fallo (SPOF).

Por ello, esta planeado en un futuro, añadir la capacidad de replicación de varios servidores master como método failover (antifallo) y que estos sincronicen sus propios datos entre servidores masters replicando la información.

5.5.3. Listado de widgets iniciales

En un servidor, se dispone de un listado de widgets iniciales para configurar un cliente. El propósito del mismo es ofrecer una base inicial para configuración mínima. El usuario puede disponer de un mayor número si los incorpora en el desarrollo.

Figura 5.10: FreeStation Server GUI - Listado de widgets iniciales

Los widgets cargados inicialmente en la configuración base son:

- ***MountDetector**: Detecta la presencia de unidades USB (montaje/desmontaje) y recopila la información del nombre de la unidad, espacio disponible y total.

Figura 5.11: Icono Widget MountDetector

- ***MountInfo**: Configura una interfaz a partir de los datos de MountDetector con una barra de progreso para mostrar el espacio disponible/total.

Figura 5.12: Icono Widget MountInfo

- ***VideoArea**: Carga un vídeo en reproducción continua y se activa en pantalla completa cuando FreeStation pasa a modo idle.

Figura 5.13: Icono Widget VideoArea

- **SliderArea**: Paneles informativos deslizantes con información para banners, transparencias, etc.
- ***GUI**: Inicializa la aplicación y lee los componentes habilitados para su carga, lanzándolos posteriormente.
- **CategoryArea**: Zona de información de listas verticales de categorías de información.

- ***Browser**: habilita la navegación de una pagina web externa o documento html local.

Figura 5.14: Icono Widget Browser

- ***BrowserView**: renderiza templates según la vista (MVC: Modelo Vista Controlador) y trata con la lógica de la información proporcionada por el componente Browser.
- ***MenuActionsArea**: menú formado por flechas de anterior/siguiente/inicio para interactuar con las posibles zonas.

Figura 5.15: Icono Widget MenuActionsArea

- ***LogoArea**: Carga un logo por defecto para la institución, universidad, escuela, oficina de turismo, ayuntamiento, etc.

Figura 5.16: Icono Widget LogoArea

- **ClockArea**: Renderiza la hora actual.
- **ItemArea**: Muestra la información sobre algún elemento (puede ser software, documento, etc) con elementos como logo, descripción, dependencia, título.
- ***TitleDisplay**: Carga un título central por defecto para la institución, universidad, escuela, oficina de turismo, ayuntamiento, etc.

Figura 5.17: Icono Widget TitleDisplay

Los widgets marcados con * han sido implementados completamente. El resto sólo parcialmente o no son suficientemente estables en la versión presentada de este documento.

5.5.4. Composición de widgets y asociaciones

Los widgets pueden componerse o asociarse.

Por ejemplo, se puede elegir distribuir y configurar un widget de almacenamiento USB (USB Storage) y un widget de montaje (Mount Device) en un cliente.

Esos widgets puede asociar operaciones para lista varios archivos de libros sobre medicina. Cuando un usuario en un cliente POI, seleccione un libro y escoja guardarlo en el USB, el widget de montaje detectará la presencia de un USB conectado, e informará del espacio disponible gracias al widget de USB Storage.

El widget de USB Storage se encargará de escribir los datos y esto resultará en un usuario contento que ha obtenido la información rápidamente y sin problemas en su dispositivo USB.

Pueden asociarse los widgets necesarios para un cliente, permitiendo configuraciones personalizadas para cada cliente a través de parámetros especializados en el widget. Por ejemplo, sólo permitir 5 elementos para guardar en el widget de Mount Device).

Figura 5.18: FreeStation Server - Asociar un widget a un cliente

La figura 5.18 muestra la asociación de un widget cliente en el servidor. Otro ejemplo, se puede elegir un widget de vídeo y habilitar que se muestre un vídeo en un POI en modo inactivo.

Cuando el usuario toque la pantalla o interactúe de algún modo con el POI, puede descargarse dicho widget y cargar un widget de navegador con una página concreta.

De forma adicional, una institución podría elegir un simple logo y establecer el widget para logos y que estuviera presente en toda pantalla mostrada usuario.

Cuando un cliente esta totalmente configurado, entonces puede ser desplegado en el POI (como un ordenador de escritorio personalizado).

Figura 5.19: Cliente FreeStation - Pantalla de bienvenida configurada

En la figura 5.19 se puede apreciar un ejemplo de pantalla de cliente desplegada. En esta pantalla desplegada en el cliente existen varios widgets configurados.

Se ha configurado un widget Browser asociado a otro BrowserView para cargar una pagina web HTML en un fichero local.

Por otro lado, se ha configurado un MenuActionsArea widget para mostrar los botones de *atrás/adelante/inicio* en la pantalla de bienvenida. A ello se añaden un widget TitleDisplay para mostrar el título de bienvenida y un widget LogoArea para cargar el logo de FreeStation.

Se presenta en el margen inferior izquierda un widget MountDetector asociado a un widget MountInfo para mostrar la información actual. Este es sólo un ejemplo posible de las muchas variaciones de configuraciones de widgets disponibles. El usuario debe analizar los widgets necesarios para su aplicación y configurarlos según su criterio.

Por ejemplo para el widget BrowserView pueden configurarse el título de la web y la web a cargar desde la configuración web en el servidor.

Figura 5.20: FreeStation Server - Configurar un Widget para un cliente

Como muestra la figura 5.20 el usuario puede introducir los datos específicos para su cliente POI.

5.5.5. Fácil de escalar a nuevas necesidades

No existe límite definiendo o usando nuevos widgets. En teoría el modelo de widgets escala para cada tipo de necesidad personal para una organización, institución, etc.

Con FreeStation algunos widgets básicos son ofrecidos por defecto, pero en el futuro podría habilitarse un repositorio de widgets asociados, widgets de empresas, widgets de asociados, etc y ofrecer un rico y amplio mercado para desarrolladores que estén interesados en desarrollar widgets personalizados para compañias, instituciones, hospitales, etc.

5.5.6. Arquitectura Watchdog

Como se explico en el diagrama de flujo de la figura 5.7 el watchdog se apoya en un módulo que registra el estado de la aplicación principal (logger) para registrar toda actividad interrumpida o posibles fallos provocados en la aplicación.

Almacena los identificadores de procesos relativos a las instancias monitorizadas y capta señales de interrupción de la aplicación como SIGTERM o excepciones no capturadas en niveles superiores.

En el momento de la implementación del lanzador se valoró usar el módulo multiprocessing nativo de Python, pero el rendimiento y legibilidad en el código empeoraban considerablemente el watchdog resultando poco efectivo y tolerante a fallos.

En una implementación posterior del lanzador, se implemento como un hilo (Thread) para aprovechar las ventajas del COW (Copy-On-Write) en GNU/Linux, además de separar la interfaz y lógica de negocio de la aplicación en diferentes procesos.

El watchdog también identifica cada hilo con un nombre único compuesto por:

$$< CLIENTE > - < TIMESTAMP >$$

Siendo CLIENTE, el nombre del cliente POI, por defecto «FreeStation» y TIMESTAMP el instante de tiempo en epoch unix que fue lanzado. De esta forma, a través del módulo logger puede identificarse el hilo lanzado que tuvo un fallo y analizar el mismo a través de depuración de la aplicación.

Para propósitos de pruebas (test) el módulo incluye un parámetro temporizador de tiempo con vida «alive time» desactivado por defecto para que sea posible monitorizar solo por un intervalo de tiempo y finalizar el watchdog.

Asimismo se incluye otro parámetro/flag para test que permite ejecutar el watchdog sin la presencia de interfaz gráfica para ejecutar test unitarios de forma más rápida y sin intervención manual.

El tiempo de monitorización por defecto esta establecido a 1 segundo. Este valor resulta óptimo ya que en caso de producirse fallo de la aplicación y se lance de nuevo, el usuario no tendrá capacidad de reacción para intervenir en el fallo e instantáneamente tendrá de nuevo usable la aplicación.

5.5.7. Arquitectura del lanzador

El lanzador tras ser invocado por el watchdog inicialmente es cargado como una subclase Thread del módulo python que recibe el nombre de FreeStationApp.

En su inicialización, se activa el módulo logger guardando en ficheros de registro cada evento de información importante.

Se hacen las comprobaciones previas de bibliotecas necesarias como GTK 3.0, limitación por lógica de una sola instancia de aplicación (ya que el cliente sólo podrá ser ejecutado una vez por POI).

Se guarda el PID del proceso padre, que coincidirá con el proceso Watchdog, al igual que también se guarda el PID del propio thread y el nombre asignado al thread que como se comentó en la sección 5.5.6 recibe el nombre $< CLIENTE > - < TIMESTAMP >$.

Se comprobará si se ha establecido un tiempo de vida para el hilo (para propósitos de pruebas unitarias) y si se ha habilitado el flag de GUI para iniciarlo (deshabilitado en caso de pruebas unitarias).

Si se cumplen las condiciones, se iniciará el módulo GUI que interactuará con el bucle principal de GTK.

Si la aplicación finaliza por algún motivo inesperado, el lanzador tratará de escribir los datos en el registro y el watchdog en su comprobación llamará de nuevo al lanzador para que reinicie la aplicación.

5.5.8. Manejo de excepciones

Las excepciones en la programación de FreeStation están presentes en el flujo del programa. La gestión de excepciones permite un mayor control del mismo e interpretar de forma más correcta cualquier posible error para su depuración.

La excepción base para cliente y backend basado en python esta implementada en la clase *FSBaseException*.

Es la encargada de establecer los métodos comunes y establecer un patrón de especialización (subclassing) para el resto de excepciones a partir de la clase nativa Exception de python. Se establece también el formato de errores de impresión para el resto de excepciones. Por tanto, se consolida el mismo concepto base de excepciones y comportamiento para todas las excepciones derivadas en la aplicación.

De esta forma excepciones de más alto nivel pueden utilizarse a partir de dicha implementación base. Un ejemplo son las excepciones siguientes:

- ClientStatusDisabled: lanzada cuando un cliente FreeStation accede al servidor con un estado marcado como desabilitado.
- NoAuthorized: lanzada cuando un cliente FreeStation accede sin autorización al servidor.

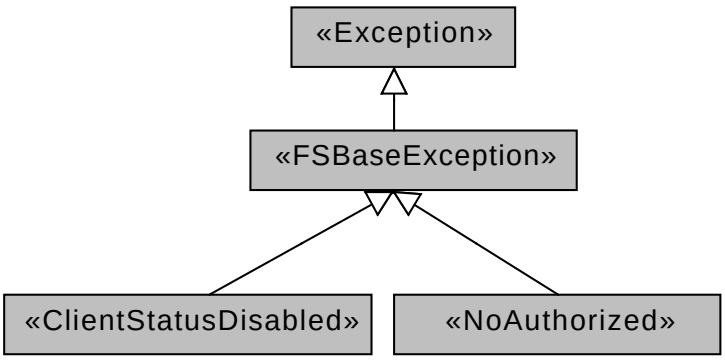


Figura 5.21: Diagrama de clases para excepciones

El diagrama de clases de la figura 5.21 representa la estructura de clases para dichas excepciones.

Estas excepciones proveen los medios para separar los detalles en el momento de producirse un comportamiento no esperado y fuera de la lógica principal del programa.

Cada cliente FreeStation debe conocer todas las excepciones posibles derivadas en llamadas al servidor. Con ese objetivo es posible un total control de excepciones (para consolidar un sistema probado como se explico en la sección 3.3). Los casos no contemplados pueden tratarse genéricamente y establecer una acción de recuperación o vuelta al estado original.

5.5.9. Test unitarios

El conjunto de pruebas permiten evaluar el correcto funcionamiento del sistema y cómo se adapta a las especificaciones descritas en la definición de requisitos.

Durante el desarrollo de este proyecto de fin de carrera se han realizado varios tipos de test. Pruebas de características con ejemplos simples de funcionamiento y pruebas unitarias.

Las pruebas de características se encuentran en el directorio tests de la carpeta raíz del proyecto. En el caso del cliente se realizan pruebas como funcionamiento de vídeo en GTK2 y GTK3, ejemplo de vídeo mínimo, ejemplo de carga para Webkit, temporizador, diferentes pruebas de características de Gstreamer (factoría de elementos multimedia, reloj multimedia, sincronización de vídeo, etc)

Por otro lado, los test unitarios se encuentran en directorio freestation/test/unit los test unitarios permiten el desarrollo continuo de una aplicación asegurando su calidad en cada pequeña modificación. Todos los test unitarios comparten la especialización (subclassing) de la clase BaseUnitTest.

A partir de la clase TestCase del módulo unittest establece los métodos comunes inicializadores (setUp) y finalizadores (tearDown) del resto de test unitarios.

5.6. PATRONES DE INGENIERÍA DEL SOFTWARE UTILIZADOS

Los patrones software[GHe06] principalmente usados en este proyecto de fin de carrera han sido:

- **Singleton**: su propósito es asegurar que sólo exista una instancia de una clase. Es un patrón creacional que garantiza la existencia de un mecanismo de acceso global a dicha instancia. Se ha utilizado en clases de *Frontend* del servidor como *CMS*, *Session*, *Error* y *ClassHash*.
- **Factory o Fábrica**: es un patrón creacional que centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear. Se ha utilizado en clases relacionadas con CouchDB como *HubFactory* y la creación de widgets.
- **Mosca o peso ligero(Flyweight)**: es un patrón estructural que reduce la redundancia cuando gran cantidad de objetos que poseen idéntica información. Se ha utilizado en la creación de widgets para una carga más ligera.
- **Facade (Fachada)**: es un patrón estructural que provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema. Es utilizado en las clases de *Frontend* del servidor para la creación de la *Api* y su administración con la clase *ApiManager*
- **Proxy**: es un patrón estructural que mantiene un representante de un objeto. Se ha utilizado en la conexión de Zero C ICE para comunicación con el cliente y servidor.
- **Adapter (Adaptador)**: es un patrón estructural que adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla. Se ha utilizado en la conexión de Zero C ICE para comunicación con el cliente y servidor.

Como patrón de diseño se ha utilizado MVC para el *Frontend* del servidor como se explico en la sección 3.4.3.

5.6.1. GUI: Clase de interfaz gráfica

Esta clase posee toda la lógica de control de la interfaz de usuario. Es la encargada de generar dinámicamente la interfaz a través de los datos proporcionados y configurados en el cliente FreeStation.

Proporcionado los interfaces genéricos lleva a cabo un cargado inicial de cada componente. Por un lado, es capaz de integrar en un thread, que servirá para establecer un cliente del tipo *FreeStationApp* y capturar la recepción y envío de mensajes.

La clase *FreeStationApp* inicializa en el thread. En el mismo se establecen los eventos para el sistema de objetos de GLib (GOBJECT) y GDK necesarios para los bindings de python y posteriores widgets.

Es importante recalcar que en el thread es necesario inicializar como thread independiente GDK a través de la función $Gdk.threads\_init()$ y establecer las condiciones de guarda de entrada y salida con $Gdk.threads\_enter()$ y $Gdk.threads\_leave()$ respectivamente. Dentro de estas condiciones es posible inicializar el principal bucle de eventos de GTK con $Gtk.main()$.

En ese momento la clase *GUI* es cargada como un meta widget general desde $widgets.GUI$. La clase *GUI* es una subclase o especialización del elemento Window de GTK. Aplica los atributos de una ventana principal a tamaño completo por defecto.

Establece asociaciones con diferentes atajos de teclado. Dispone de los atajos de teclado pulsando la tecla f minúscula para cambiar el modo de pantalla completa (a modo depuración, debe ser deshabilitado en producción), al igual que la tecla q, para simulara el cerrado de la aplicación y que el Watchdog lo detecte reiniciando la aplicación.

Esta clase se encarga principalmente de cargar la configuración XML proporcionada por el cliente a través de la clase WidgetLoader. A través de la ejecución del método privado $\_\_load_widgets$ se realizará una petición de datos en la que se invoca y delega la tarea a la clase *WidgetLoader*.

Una vez la configuración es proporcionada por WidgetLoader, se realiza una carga dinámica de la clases de forma metaprogramada.

Listado de código fuente 5.1: Carga dinámica de clases - Metaprogramación

```
self.widgets = WidgetLoader().get_widgets()

for (name, properties) in self.widgets:

    widget_error = False

    mod = __import__('freestation.widgets', fromlist = [str(name)])

    try:
        mod_class = getattr(mod, name)
    except AttributeError, e:
        print 'Error: could not load the widget ' + name
        widget_error = True

    if not widget_error:
        klass = getattr(mod_class, name)

        temporal_object = klass()

        temporal_attribute = temporal_object.__class__.__name__

        # Create lowercase with underscore
        # attribute class from name widget
        name_attribute = ''
        i = 0
        for letter in temporal_attribute:
            if letter.isupper():
                if i == 0:
                    letter = letter.lower()
                else:
                    letter = '_' + letter.lower()

            name_attribute += str(letter)
            i += 1

        # Create attribute on class with metadinamic class created
        setattr(self, name_attribute + '_widget', temporal_object)
```

En el listado de código 5.1 se expone la metaprogramación implicada para la carga de widgets. Mediante el módulo importlib (función *__import__*) de python se cargan todos los widgets configurados en la lista XML recibida. Esta lista XML recibe un formato específico que es procesado y devuelto en formato de listas y tuplas de python, conteniendo el nombre del widget y las propiedades asociadas al mismo. En el proceso de carga de datos se realizan algunas adaptaciones de los nombres de clase nombrados en formato camelCase[1], a formato con guiones bajos para python y sus módulos.

[1] camelCase: `http://es.wikipedia.org/wiki/CamelCase`

Los widgets son cargados dinámicamente en tiempo de ejecución desde los módulos existentes en freestation.widgets. Se realiza una comprobación si se intenta cargar un módulo de widget por XML que el cliente no haya recibido del servidor.

En caso de no existir errores en la carga de widgets, se carga dinámicamente la clase del widget desde el módulo de widget y se invoca en tiempo de ejecución para crear un objeto. Para guardar la referencia, se crea un atributo a la clase GUI en tiempo de ejecución. Posteriormente la clase invocará el empaquetado y disposición del los widgets cargados a través de dichos atributos creados dinámicamente.

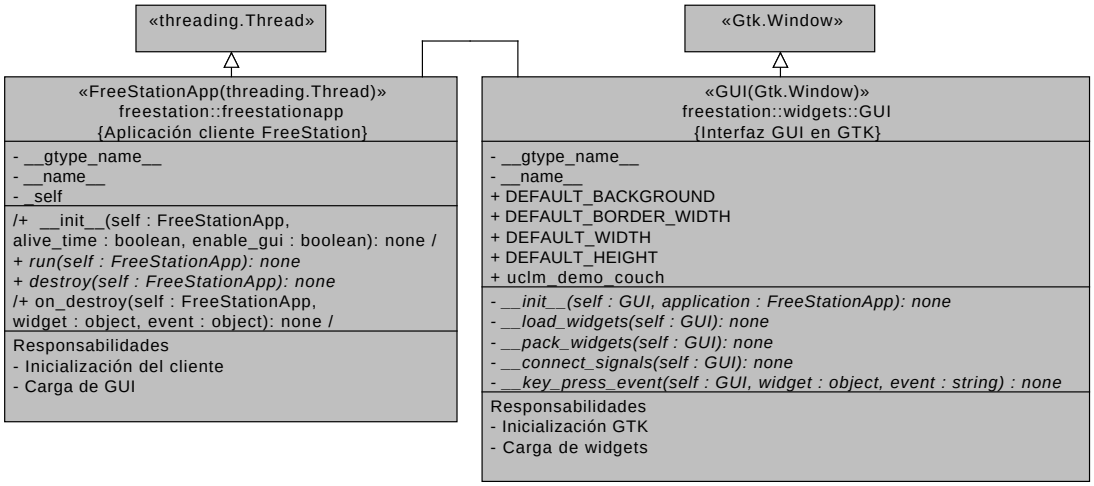


Figura 5.22: Diagrama de clases FreeStationApp y GUI

En el diagrama de clases de la figura 5.22 se detalla el diseño y estructura de las clases *FreeStationApp* y *GUI*.

5.6.2. WidgetLoader: cargador dinámico de widgets

Es la clase encargada de procesar el archivo widgets.xml recibido del servidor. El formato de widgets.xml sigue un formato especial.

Listado de código fuente 5.2: Ejemplo de configuración XML para archivo widgets.xml

```
1
2  <interface>
3
4      <widget>
5          <name>TitleDisplay</name>
6          <properties>
7              <position type="relative" child="0" pack="None|Start|End">MainWindow</
                   position>
8              <homogeneous>1</homogeneous>
9              <spacing>5</spacing>
10             <width>200</width>
11             <height>300</height>
12             <data>UCLM - FreeStation</data>
13         </properties>
14     </widget>
15
16     <widget>
17         <name>BrowserView</name>
18         <properties>
19             <position type="absolute"></position>
20             <width>200</width>
21             <height>300</height>
22         </properties>
23     </widget>
24
25     <widget>
26         <name>MountInfo</name>
27         <properties>
28             <position type="absolute"></position>
29             <width>200</width>
30             <height>300</height>
31         </properties>
32     </widget>
33
34 </interface>
```

En el ejemplo 5.2 puede observarse que todo widget necesariamente debe ir etiquetado bajo la marca de entidad XML “interface”. Posteriormente una serie de conjuntos ilimitados de widgets pueden ser listados mediante la entidad “widget”. Toda entidad widget debe contener al menos una entidad “name” y otra entidad “properties” en su interior.

Las propiedades definen un conjunto de características según el widget. Estas características son dependientes de la implementación en python que se realice. Sólo serán reconocidas las características que la implementación en python del widget reconozca.

Por ejemplo para un widget pueden definirse posiciones relativas o absolutas a otros widgets, pero esta funcionalidad no esta completamente implementada. De igual forma pueden establecerse el ancho y alto del widget, espacio entre widgets, distribución homogénea y datos de preconfiguración cargados.

Todo evento de información que suceda es también guardado mediante la clase Logger en el registro general de eventos.

La clase WidgetLoader además realiza comprobación de integridad del fichero widgets.xml. Si por ejemplo existen errores al definir las entidades interface, widgets, etc.

Si algún error es detectado, como se comento en la sección 5.5.8 se lanza una excepción a la clase superior, en este caso GUI, para que trate adecuadamente el widget o conjunto de widgets que no pudieron ser cargados.

En particular existen excepciones de alto nivel que son lanzadas si se detecta algún problema y con las que el cliente FreeStation debe capturarse.

Todas las excepciones lanzadas son especializaciones de la clase *FSBaseException*.

Las principales excepciones son:

- **WidgetXMLFileNotFound**: lanzada cuando el archivo widgets.xml no esta presente o no tiene permisos de lectura.
- **WidgetXMLBadFormed**: lanzada cuando el archivo widgets.xml no contiene un formato XML válido o es imposible continuar el análisis del mismo.

La clase WidgetLoader basa su implementación en el módulo minidom[2] de python que le permite diferenciar entre nodos, datos, entidades XML.

Inicialmente se intenta leer y cargar el archivo XML widgets.xml desde la función $\_read\_data$ y $\_load\_xml$. Sino se encuentra un éxito se lanzará la excepción *WidgetXMLFileNotFound*, de lo contrario se intentará analizar el modelo de documento de objetos (DOM).

Si se dispone de un DOM, se intentarán validar etiquetas como nodos por ejemplo como las etiquetas interface, widgets, etc. En caso fallido se lanzará la excepción *WidgetXMLBadFormed* y se intentara seguir leyendo el archivo para análisis de otros posibles widgets.

Listado de código fuente 5.3: Tipos de nodos XML

```
ELEMENT_NODE                = 1
ATTRIBUTE_NODE              = 2
TEXT_NODE                   = 3
CDATA_SECTION_NODE          = 4
ENTITY_REFERENCE_NODE       = 5
ENTITY_NODE                 = 6
PROCESSING_INSTRUCTION_NODE = 7
COMMENT_NODE                = 8
DOCUMENT_NODE               = 9
DOCUMENT_TYPE_NODE          = 10
DOCUMENT_FRAGMENT_NODE      = 11
NOTATION_NODE               = 12
```

[2]Módulo minidom python:
http://docs.python.org/library/xml.dom.minidom.html

El listado 5.3 ofrece todos los tipos de nodos XML disponibles con sus valores designados y que son analizados en el proceso. En particular los elementos de información útil serán $ELEMENT\_NODE$ que son filtrados a partir de la función $\_filter\_element\_nodes$.

En ellos, se encuentra la información del valor del nodo, que será almacenado en formato de tuplas python y devuelto como resultado a capas superiores.

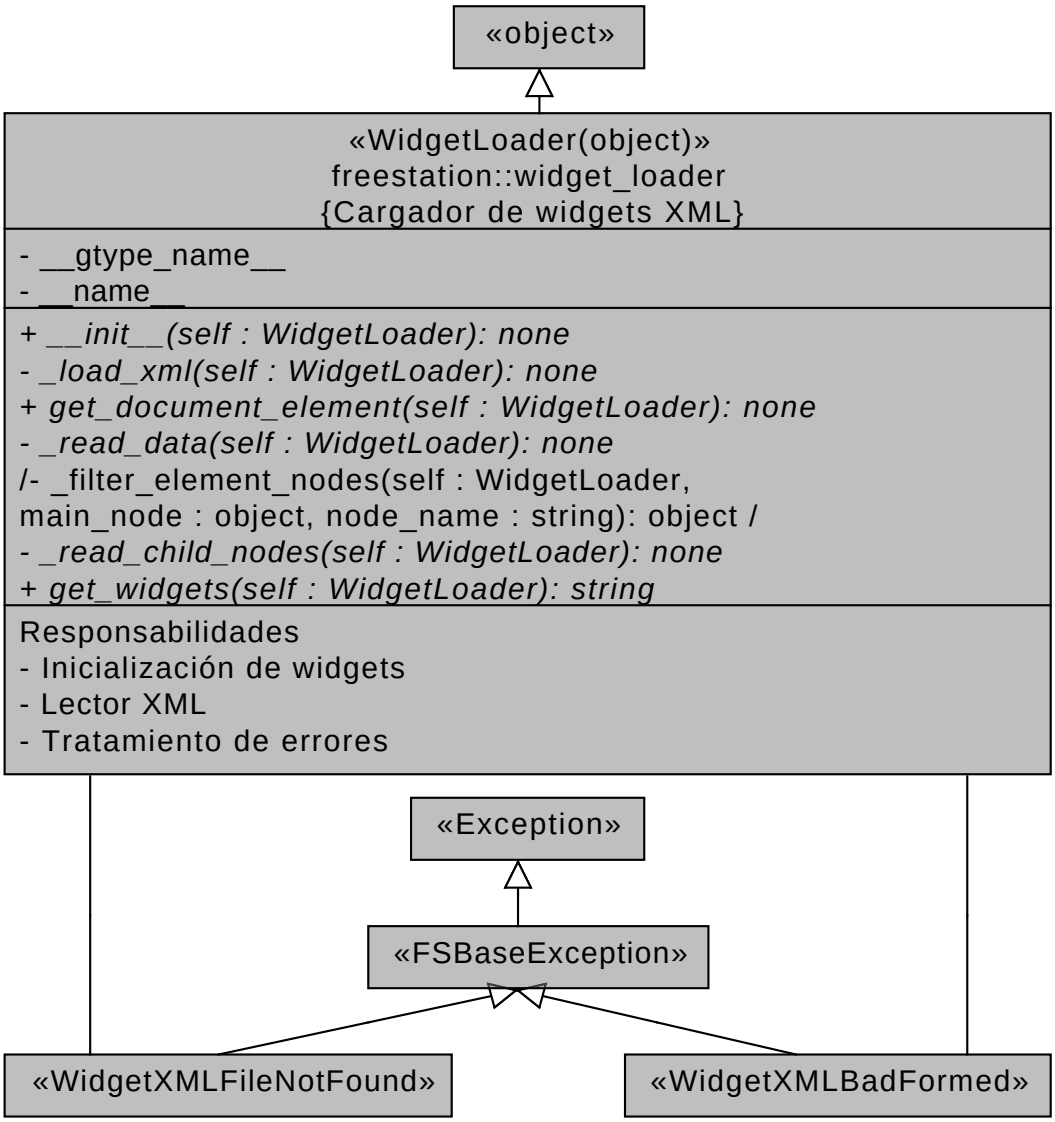


Figura 5.23: Diagrama de clases WidgetLoader

El diagrama de clases 5.23 ilustra la clase WidgetLoader y sus excepciones.

5.6.2.1. Backend

El *Backend* esta desarrollado como un módulo independiente. Dicho módulo esta basado en una aplicación Ice llamada *FreeStationServer*. Por otro lado, el *Backend* es inicializado mediante el servidor web *Frontend* en PHP.

Backend ZeroC ICE

En esta sección se describirá el subsistema de *Backend* para lado servidor basado en ZeroC ICE, su cometido, su funcionalidad y su implementación.

ZeroC ICE es el middleware por excelencia para sistemas distribuidos como se explico en la sección 4.5.5.2.

La potencia de invocación de objetos remotos como si fueran objetos locales hace posible el funcionamiento de la clase *FreeStationServer*. La aplicación internamente realiza una enmascaración mediante un proxy para la parte del cliente.

El desarrollo distribuido permite un gran potencial. Una vez es inicializado por parte del *Frontend* se invoca una instancia de *FreeStationServer*, que se basa en una especialización de la clase *Ice.Application*.

Dicha clase, necesita de una configuración inicial de propiedades que por defecto son almacenadas en la clase controladora.

Como se explicó en la sección 3.5 FreeStation usa un MVC y una capa de datos basada en XML. El servidor necesita un esqueleto que traduzca los eventos entre la comunicación con el cliente. La comunicación no deja de ser una comunicación que sigue protocolos de red estándar como TCP o UDP.

Por este motivo, FreeStation utiliza esta estructura internamente para representar las configuraciones. La clase ICE llamada *Ice.InitializationData* permite establecer propiedades de configuración.

A través del método $init\_properties$ las configuraciones se establecen en tiempo de

ejecución desactivando la posibilidad de carga externa de configuraciones por archivos de configuración de ICE. La directiva ICE utilizada es $Ice.Config$.

Con propósitos de identificación de la aplicación se establece un nombre del proceso ICE con la directiva de configuración $Ice.ProgramName$

Para evitar problemas de limites de memoria ($Ice :: MemoryLimitException$) se establece un tamaño máximo de mensaje de 10 MB en lugar de 1 MB establecido por defecto con la directiva $Ice.MessageSizeMax$.

Si esta activo el modo depuración de FreeStationServer (activado con el flag $DEBUG$) se imprimirá el PID del proceso con $Ice.PrintProcessId$

Por otro lado se establecen los ficheros de salida estándar y error con $Ice.StdErr$ e $Ice.StdOut$ y el modo NOHUP para el proceso con $Ice.Nohup$. Las trazas de la aplicación están activadas con el fin de mejorar la información mostrada en el *Frontend* para ello se utiliza la directiva $Ice.PrintStackTraces$ y un modo excesivamente de depuración si trazas si esta activo el flag $DEBUG$. De la misma forma se activan otras configuraciones con el fin de mejorar el rendimiento de la aplicación.

Una vez se ha establecido toda la configuración necesaria, se procede a la carga de ficheros slice .ice.

La carga de la interfaz es realizada por el método $load\_ice\_interface$. Este buscará una especificación de archivo slice en $ice/fs.ice$ y cargará el slice en tiempo de ejecución con $Ice.loadSlice$. Para ello debe estar previamente generado. Si se detecta que no ha sido generado, se procederá a generarlo con un comando del tipo:

```
[["cpp:include:list"]]
slice2py --underscore --output-dir . ice/fs.ice
```

Esto generará un directorio FS (módulo) con el slice generado como código Python.

Especificación slice para ICE

Un archivo de especificación Slice es un lenguaje de definición de interfaces[3] usado por ICE. Es un mecanismo de abstracción para separar las interfaces de objetos de sus implementaciones. Define un contrato entre servidor-cliente para una aplicación, incluyendo las interfaces, operaciones, parámetros, tipos de datos y excepciones.

La descripción es independiente del lenguaje de implementación. La sintaxis de Slice tiene un gran parecido con la sintaxis de C++ o Java por lo que es fácilmente reconocible.

A dichos contratos se les denomina en inglés “push model” o “pull model” dependiendo del lado que lo emita (emisor o receptor). En ocasiones un mismo modelo puede ser push/pull de ambos tipos a la vez.

Las definiciones del Slice se compilan en particular para el lenguaje en el que son implementadas por un compilador. Esto permite al compilador traducir definiciones escritas en un lenguaje independiente a un lenguaje especifico de definiciones de tipos y APIs. Es por tanto un lenguaje puramente declarativo.

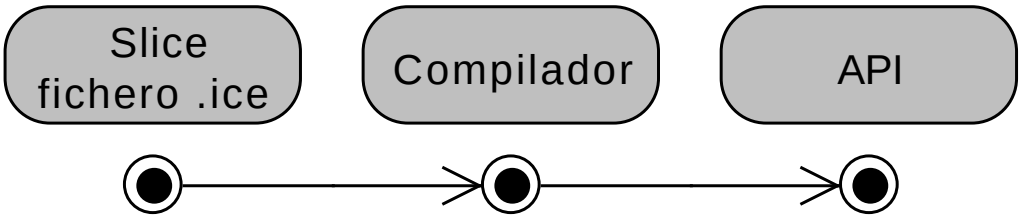


Figura 5.24: Diagrama de estados compilación Slice]

Su principal enfoque esta en las interfaces de objetos y las operaciones que definen. Los objetos también pueden ofrecer persistencia de objetos a través de la definición de tipos de datos. Esto es posible gracias al intercambio de datos entre cliente y servidor, de esta forma se evitan intercambios de datos arbitrarios y sin estructurar.

[3]The Slice Language:
`http://doc.zeroc.com/display/Ice/The+Slice+Language`

El módulo FS es autogenerado a partir del archivo slice fs.ice que actúa de igual forma como contrato modelo push/pull para cliente y servidor. El archivo de especificación es el siguiente:

Listado de código fuente 5.4: Especificación slice para ICE

```
1
2  module FS
3  {
4     sequence<byte> File;
5
6     sequence<byte> FileBlock;
7
8     interface Api
9     {
10      void getWidgets();
11      string getXMLWidgets();
12      void version(out string sout);
13      int getFileSize(string path);
14      File getFile(string path);
15      FileBlock getFileChunk(string path, int pos, int size);
16      void isAuthorized(out bool sout);
17      int getClientId();
18    };
19
20    interface Widget
21    {
22      void getName();
23    };
24
25    exception GenericError
26    {
27      string reason;
28    };
29
30    exception NoAuthorized extends GenericError{};
31 };
```

En el se define un módulo FS, con varias interfaces, tipos de datos y excepciones.

Los tipos de datos $File$ y $FileBlock$ son definidos como secuencias de bytes y serán utilizados para la transferencias de ficheros de las interfaces. Las principales interfaces definidas son Api y $Widget$. La interfaz Api define unos métodos para la administración y comprobación de privilegios al conectar al misma con $isAuthorized$. Métodos como $getClientId$, $getWidgets$, etc son métodos usados en la Api para comunicación entre cliente y servidor.

Creación del comunicador

Una vez el servidor es configurado, creado el fichero slice, se ejecuta con el método $start$ y este llama al $main$ de la clase padre *Ice.Application*. Tras ello, el método invocado es run que es sobreescrito en la clase hija *FreeStationServer*.

En ese momento se crea un comunicador siguiendo el modelo de hilos de Zero C ICE[4]. El comunicador es el encargado de crear un contexto implícito y gestionar los eventos de la clase *Ice::Router*. Este contexto será accesible posteriormente y servirá para guardar datos de utilidad como número de peticiones servidas para cada cliente. Las propiedades del comunicador podrán visualizarse con el método $show\_communicator\_properties$.

Creación del adaptador y puntos finales de conexión

Cuando se ha establecido la comunicación, es necesario crear un objeto adaptador. Su principal funcionalidad será mapear los objetos ICE a los sirvientes (servants) que comparten los hilos creados por el comunicador. También es posible configurarlo como individual para disponer de una reserva de hilos privada. El método $create\_adapter$ creará el adaptador y tratará con cualquier error que pueda suceder.

Los puntos finales de conexión son gestionados con la clase *Ice.Endpoint*. FreeStation usa por defecto el protocolo UDP en el puerto 10000 ya que no necesita confirmación (ACK) de los mensajes, ya que ICE asegura la recepción. De esta forma se consigue una mayor tasa de transferencia en el envío de archivos y conexiones. La información sobre puntos finales de conexión puede visualizarse con el método $show\_endpoints\_info$. En caso de existir varias IPs en la máquina servidor o conexión a varios puertos, la conexión aumentará de rendimiento al disponer de mayores puntos de conexión.

Creación del sirviente

Una vez creado el adaptador se creará una instancia de sirviente. Un sirviente mantiene todos los objetos de ICE en memoria. Los sirvientes están optimizados para no almacenar demasiados estados en memoria y evitar el exceso de recursos escalando apropiadamente. El método $create\_servant$ es el encargado de crear el sirviente.

[4]The Ice Threading Model:
`http://doc.zeroc.com/display/Ice/The+Ice+Threading+Model`

Módulo FS.Api

Este módulo es autogenerado en código python como resultado de la compilación del slice. Mapea mediante clases temporales las interfaces con $Ice.createTempClass()$. Establece el proxy para comunicación entre cliente y servidor mediante *Ice.Object* y *Ice.ObjectPrx*. Las operaciones de las interfaces son mapeadas en la autogeneración con a través de $IcePy.Operation$ indicando los parámetros de la operación y el modo de operación (normal o idempotente). Las excepciones sin embargo son mapeadas con la clase *Ice.UserException*.

Es importante recalcar que si la especificación del slice cambia, el cambio no se propaga en tiempo real en el servidor. Es necesario que el módulo se vuelva a autogenerar y se reinicie el servidor para carga del nuevo módulo compilado.

Api basada en módulo FS.Api

Una vez el módulo es autogenerado para utilizarlo es necesario hacer una especialización del mismo. Para ello se crea la clase Api heredando de FS.Api. La API se encarga de sobreescribir cada método y añadirle funcionalidad a las operaciones teniendo en cuenta el contexto creado por el comunicador y recibiendo un parámetro extra con el objeto current (Ice.Current[5]). Este objeto permite almacenar el contexto de operación por cada invocación en el servidor. Es un parámetro implícito y usado intensivamente por la API.

La API en su inicialización delega toda lógica de negocio a la clase *ApiManager*. Su tarea es básicamente establecer el objecto current, comprobar la autorización del cliente y actualizar el número de peticiones realizadas por el cliente.

ApiManager

La clase *ApiManager* trata con la persistencia de la API y su lógica de negocio. Recopila información del cliente como IP de conexión y actualiza datos de última conexión realizada o número de peticiones. Su funcionalidad principal es la descarga de archivos para el cliente *FreeStationClient*, configuración de widgets disponibles y generación de configuraciones en XML para que estas sean interpretadas por el *WidgetLoader* como fue explicado en la sección 5.6.1.

[5] Ice.Current object:
`http://doc.zeroc.com/display/Ice/Ice-Current`

ThreadNotification

Con el fin de mejorar las notificaciones de hilos creados en el contexto de Zero C ICE se habilita una clase que sobreescribe la funcionalidad básica de la clase nativa *Ice.ThreadNotification*. La clase que lo hace posible es *ThreadLogger* y notifica cuando un thread es iniciado o parado en la salida estándar del servidor.

Estados

La clase *States* proporciona los tipos de estado del servidor. Los estados permitidos son los siguientes:

- STANDBY: nodo habilitado, pero no se encuentra en uso.
- CONNECTED: nodo habilitado y en uso.
- BLOCKED: nodo habilitado, pero bloqueado
- PARK: nodo habilitado, pero no configurado

Configuración de persitencia

La clase *DBConfig* es la encargada de proporcionar los parámetros de conexión para almacenamiento en persistencia. La persistencia esta conectada a MySQL para almacenar datos de configuración de clientes como widgets usados, clientes autorizados, etc.

5.6.2.2. Cliente ICE

El cliente ICE se basa en la clase *FreeStationClient* su principal acometido es la descarga de configuración de widgets en XML para el cliente que se conecta al servidor. También habilita la funcionalidad de descarga de ficheros necesarios que debe ser configurada manualmente. En toda operación el cliente realizará una petición de autorización previa y comprobación del estado de cliente habilitado. Estas operaciones son realizadas con el método $is\_authorized$ que previamente creara una base y proxy de conexión al servidor.

La configuración de directivas para ICE en el cliente es una versión muy simplificada del backend servidor. Únicamente es necesario establecer el tamaño máximo de mensaje y el tiempo de timeout (desconexión) para el servidor.

Descarga de archivos

La descarga de archivos por parte del cliente es realizada a través de las operaciones definidas en Slice de la API. La operación $getFile$ y getFileSize están encapsuladas en el cliente con la invocación del método $request\_file$. Esta se encarga en solicitar los archivos por tamaños (chunks) según el tamaño del archivo a descargar. El tamaño de los chunks es determinante, ya que a un menor tamaño se realizarán mayor número de mensajes con un mayor tráfico de red, pero a mayor tamaño, existirá mayor riesgo de pérdida de paquetes por UDP y reinicio del chunk del mensaje.

Estadísticas de descargas de archivos

Pruebas de rendimiento por tamaño de chunk			
Tamaño archivo	Tiempo	Tamaño chunk	Tasa transferencia
1.5 MB	137 sec	1 KB	10,94 KB/s
1.5 MB	16 sec	10 KB	93,75 KB/s
1.5 MB	4 sec	50 KB	375,00 KB/s
1.5 MB	2.84 sec	100 KB	528,16 KB/s
1.5 MB	2.67 sec	150 KB	561,79 KB/s
1.5 MB	2.26 sec	500 KB	663,71 KB/s
700 MB	961 sec	150 kb	728,40 KB/s

Cuadro 5.1: Transferencias de chunks para diferentes tamaños de archivo

La tabla 5.1, muestra pruebas de rendimiento (asumiendo una conexión sin errores y dedicada al propósito) de descarga de archivos de 1.5 MB y 700 MB según el tamaño del chunk se pueden observar diferentes tiempos de descarga. Tras las pruebas la conclusión de valor por defecto para chunks se estableció en 150 KB ya que el rendimiento fue plenamente aceptable y se garantizaba un buen tamaño para evitar reenvío de mensajes demasiado grandes.

Ejemplos de descargas de archivos

En el anexo de instalación y configuración se facilitan ejemplos de descarga de archivos para clientes.

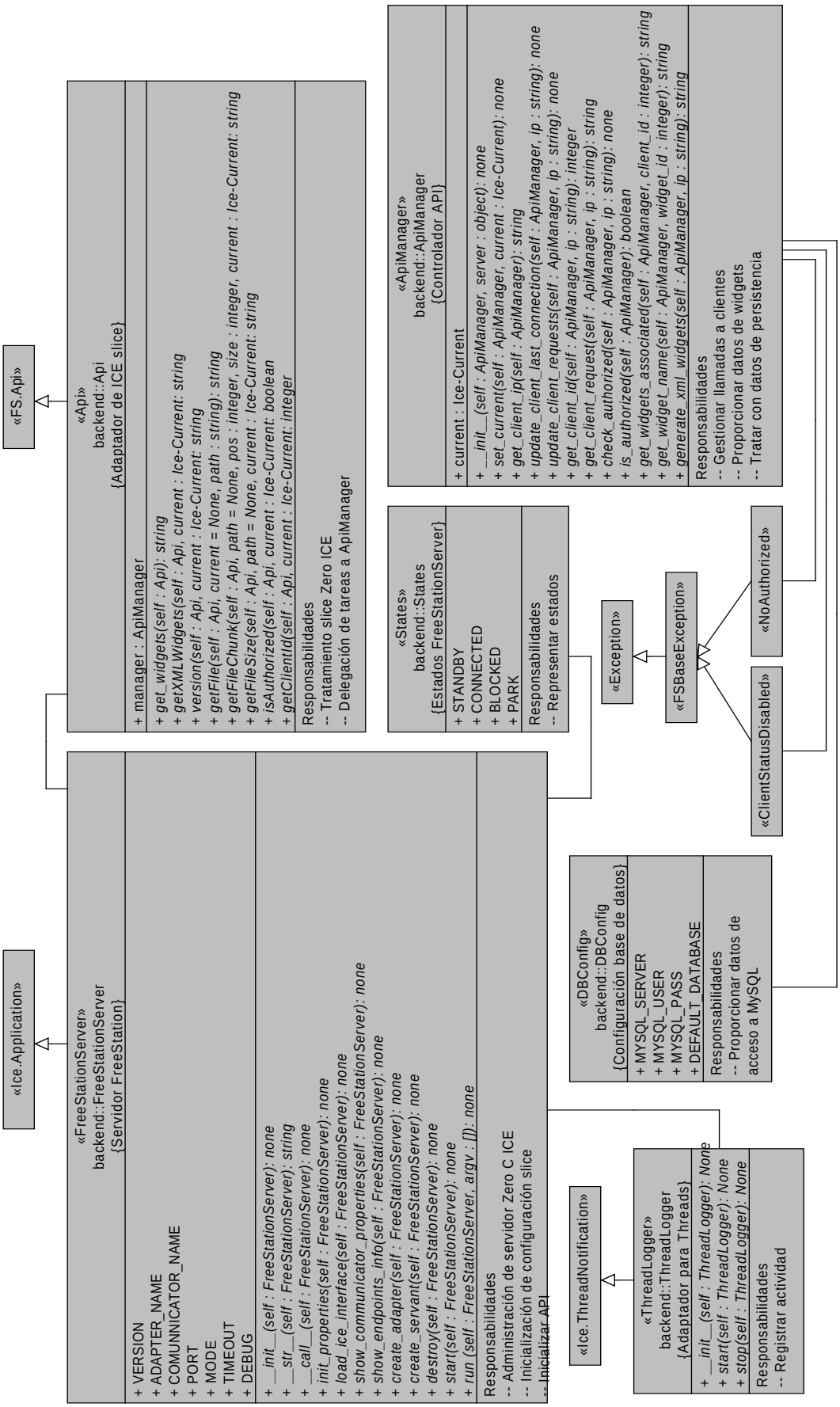


Figura 5.25: Diagrama de clases para Backend Zero C ICE

La figura 5.25 muestra en conjunto el diagrama de clases implicado en este sistema backend servidor de la aplicación que ha sido explicado en anteriores secciones.

5.6.2.3. Frontend

El *Frontend* esta basado en PHP y maquetado con HTML[W3C99], CSS[W3C06] y Javascript. Es completamente modular y los subsistemas que intervienen están desacoplados. La forma en la que interactúan los subsistemas es sencilla y transparente al programador o usuario.

La aplicación web esta orientada a la administración del servidor y configuración de widgets y clientes. Es una capa de software que facilita la edición y configuración sencilla. Estas tareas puedan realizarse generando los datos manualmente, pero conllevan mayor tiempo y complejidad.

Los ejemplos de uso del *Frontend* son detallados en el anexo de Apéndice B: Manual de usuario. En esta sección se detallará su arquitectura y funcionamiento para la generación dinámica de interfaces a través de widgets y su comunicación con backend servidor.

El frontend esta diseñado como una arquitectura con tres subsistemas o módulos. Estos son: ClientCore, ServerCore y WidgetCore. Posteriormente un módulo de CMS los conecta y accede a la persistencia de datos utilizando el patrón MVC como se explico en la sección 3.4.3. A continuación se detallan los subsistemas.

Subsistema de widgets

Los widgets tienen un peso importante dentro del *Frontend*. Este subsistema es el encargado de la gestión de operaciones con widgets.

La clase *WidgetCore* escrita en PHP construye las bases para manejar las operaciones básicas de gestión de widgets. Permite obtener todos los widgets disponibles en el servidor con el método $getList()$. También puede obtenerse la configuración de un widget conociendo el identificador numérico y único del mismo. Para ello se usa la función $get()$. Sin embargo, puede ser necesario obtener los datos del widget a través del nombre y para ello es necesario la función $getFromName()$. La comprobación previa de la existencia de un widget puede

realizarse mediante su identificador con la función $exists()$.

Una vez se disponen de los datos necesarios del widget, la clase *WidgetCore* puede renderizar la información del mismo con el método $render()$. Para creación de widgets personalizados el programador sólo debe limitarse a implementar las interfaces de los objetos que desee crear y estos serán cargados según su funcionalidad.

Subsistema de clientes

El subsistema de clientes está organizado por la clase *ClientCore*. Esta clase encapsula el tratamiento y gestión de clientes proporcionando un acceso sencillo a sus estructuras y datos.

Los métodos de la clase *ClientCore* permiten obtener los datos de un cliente dado su identificador numérico único con el método $get()$. Comprobar la existencia de un cliente a través de un identificador con el método $exist()$. Por otro lado, para obtener la lista de widgets asociados a un cliente se utiliza el método $getWidgetsAssociated()$ y para comprobar si un widget está asociado a un cliente $isWidgetsAssociated()$ o borrar un determinado widget de un cliente con $deleteWidgetsAssociated()$.

Asimismo, un cliente puede darse de baja usando el método delete() y el identificador de cliente. Es importante recalcar que si el cliente no existe la excepción *ClientNoExist* será lanzada. Para habilitar o deshabilitar el estado de un cliente, se puede proceder al cambio con la función $changeStatus()$.

Subsistema de gestión del servidor

La gestión del *Frontend* para el servidor se realiza mediante la clase *ServerCore*. Es un clase núcleo (core) que habilita las funciones más básicas para operar con el *Backend*. Entre sus operaciones están el inicio del servidor con la función $start()$, que previamente vaciar los registros de log con la función privada $resetLogs()$ para iniciar unos nuevos en la ejecución. La parada del servidor con $stop()$. Para conocer el PID de ejecución del proceso servidor se proporciona la función $getServerPid()$. Esta función devolverá vacío en caso de no encontrar un proceso en ejecución.

Los registros de log que son almacenados corresponden según el nivel de importancia del mensaje o si proceden de la salida estándar del proceso o error.

Los niveles asociados son:

- **RUNNING_LOG**: registro para la salida estándar de ejecución. Los datos se almacenan en el fichero running.log.
- **STANDARD_OUTPUT_LOG**: registro para la salida estándar normal del proceso *Backend* de ICE. Los datos se almacenan en el fichero ice_output.log.
- **STANDARD_ERROR_LOG**: registro para la salida estándar de error del proceso *Backend* de ICE. Los datos se almacenan en el fichero ice_error.log
- **WARNING_LOG**: registro para mensajes de aviso en la salida estándar. Los datos se almacenan en el fichero ice_error.log
- **TRACE_LOG**: registro para mensajes de trazas de depuración en la salida estándar. Los datos se almacenan en el fichero ice_error.log
- **OTHER_LOG**: registro para otros tipos de mensajes sin categorizar en la salida estándar. Los datos se almacenan en el fichero ice_error.log

Esta clase es la base de operaciones para una clase de más alto nivel como *WidgetServer*. Es el “widget"para la gestión del servidor. Esta clase actúa como vista para la anterior clase controlador *ServerCore* y renderiza los datos de registros de logs en cajas animadas y pestañas. Para ello se basa en la lectura de ficheros indicados en los anteriores niveles de log y los analiza según el tipo de mensaje. Por otro lado, obtiene la representación del estado del servidor y datos del PID de proceso, ofreciendo botones de iniciar, parar o reiniciar el servidor.

Subsistema de administración de contenidos

FreeStation se basa en un framework propio y original para la administración de contenidos o CMS (Content Management System). Se basa ampliamente en el patrón de diseño MVC reuniendo en una sola biblioteca las utilidades necesarias. La clase principal permite abrir y cerrar páginas y centrarse sólo en la vista del contenido. Para ello utiliza métodos como $openPage()$ y $closePage()$. Realiza una carga dinámica de clases en ejecución a través

de una clase *Loader* que detecta en tiempo de ejecución la clase a cargar e importa el código. De esta forma el programador ahorra tiempo en estructurar dependencias o fallos por falta de archivos o módulos.

Para esta carga dinámica se basa en la clase *ClassHash* que es un hash de nombres de clases apuntando a sus rutas. De esta forma el CMS puede cargar cualquier clase preestablecida en este hash. Gracias a esta tecnología, los archivos son precargados rápidamente y se evitan importaciones de código redundantes. En caso de fallo se lanzará una excepción como *ClassNotExist* que ayudará al programador a conocer fácilmente que clase no ha sido añadida.

El CMS también incorpora una gestión de errores en tiempo de ejecución para errores fatales en PHP. PHP por defecto aborta la ejecución del script en caso de un error fatal, pero la clase *ErrorManager* activa manejadores para realizar un tratamiento del error, ya sea archivando el error para posterior análisis o mostrándolo en pantalla de forma agradable para su depuración.

Para el tratamiento de persistencia el CMS incorpora una clase llamada *MySqlDriver* que trata con consultas a base de datos. Para limpieza y seguridad de los datos se incorpora una clase *Sanitizer*. De igual modo para el tratamiento de sesiones de usuario se incorpora la clase *Session*.

Para la creación de un menú vertical con los distintos apartados la funcionalidad es ofrecida por la clase *WidgetVerticalMenu*. Cabe destacar que el desarrollo del *Frontend* es mucho más sencillo a partir del CMS propio construido.

Estos subsistemas son el núcleo del *Frontend* del servidor web y permiten realizar todas las operaciones de forma básica e independiente. De este modo, la estructura de un subsistema permite un uso completamente modular del mismo, pudiéndose reutilizar en otras partes de la aplicación. Por otro lado, es posible añadir más funcionalidad sin tener que cambiar ninguna otra parte de la arquitectura. Como conjunto inicial, estos subsistemas cubren la mayoría de las necesidades básicas.

El siguiente diagrama de clases muestra las utilidades que brinda el sistema en conjunto y sus subsistemas para el *Frontend*.

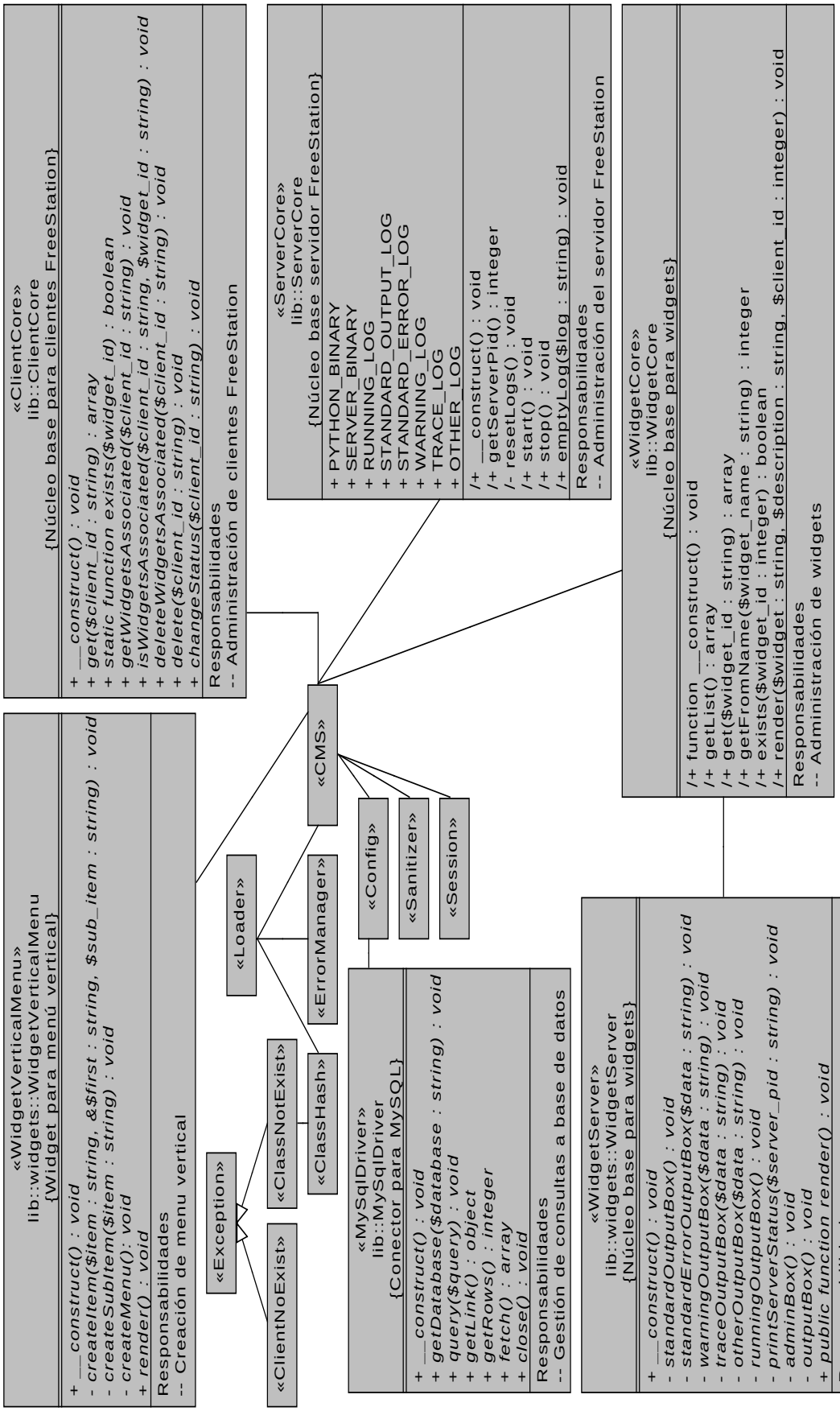


Figura 5.26: Diagrama de clases para Frontend PHP

5.7. CAPA DE PERSISTENCIA

En la capa de persistencia se usa un servidor de base de datos MySQL 5.1 para almacenar los resultados no volátiles.

Todos los datos almacenados han utilizado el motor de almacenamiento InnoDB[6] de MySQL de manera transaccional.

El diagrama ERR tiene el aspecto siguiente:

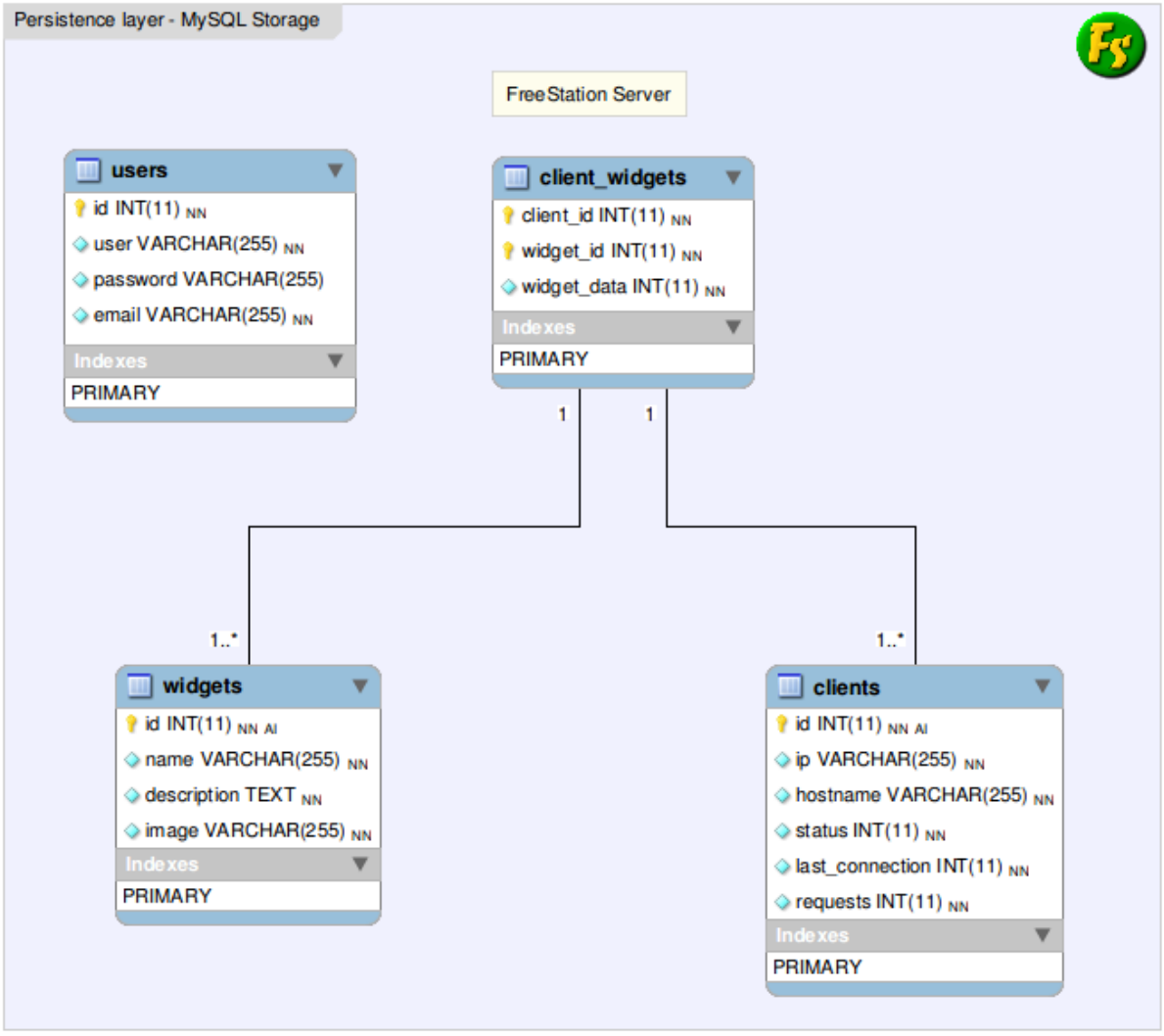


Figura 5.27: Diagrama ERR

[6]InnoDB: `http://dev.mysql.com/doc/refman/5.1/en/innodb-storage-engine.html`

Se pueden apreciar 4 tablas diferentes.

- Tabla usuarios: es la encargada de almacenar los datos de los usuarios que acceden a la aplicación frontend mediante sus credenciales.

  Se almacenan datos como usuario, contraseña y email. La contraseña es cifrada con el algoritmo SHA-1[7]

- Tabla clientes: se guardan los datos pertenecientes a clientes Freestation. Cada cliente posee un identificador único e IP permitida para el acceso. Puede asociarse un alias como hostname para fácil identificación humana. Se guardan datos como última conexión y número de peticiones realizadas en total.

- Tabla widgets: almacena los widgets existentes en el servidor con un identificador único, nombre, descripción e imagen.

- Tabla clientes-widgets: utilizada para almacenar las relaciones entre widgets asociados a clientes y datos personalizados por cada widget.

5.8. ARQUITECTURA APACHE COUCHDB

CouchDB™ es una base de datos documental basada en JSON, que permite realizar consultas MapReduce mediante Javascript y dispone de una API REST normalizada que puede ser gestionada mediante el protocolo HTTP[8] (veáse la sección 5.8.2.1).

Su desarrollo está orientado totalmente a recursos web[AnJ10]. Funciona incluso en aplicaciones móviles. El acceso a documentos se realiza generalmente mediante un navegador web vía HTTP. Las consultas combinadas transforman los documentos con Javascript.

La sincronización y replicación están integradas por defecto. Para ello, utiliza una replicación incremental para distribuir los datos eficientemente, soportando incluso configuraciones de maestro-maestro con detección automática de conflictos.

La configuración maestro-maestro permite que dos instancias CouchDB repliquen los

[7] SHA-1: Secure Hash Algorithm es cifrado por funciones hash seguro `http://www.ietf.org/rfc/rfc3174.txt`

[8] Wiki CouchDB: `http://wiki.apache.org/couchdb/`

mismos datos de forma sincronizada para obtener una alta disponibilidad y como solución ante fallos de caídas por un único nodo. De esta forma, si un nodo couchDB se pierde, siempre existe una copia exacta y original en otro nodo.

Por otro lado, permite un alto número de particiones para datos siendo consistente en todo momento con los datos proporcionados. La tolerancia de fallos está asegurada anteponiendo los datos como objetivos de mayor prioridad dentro del flujo de la aplicación. Por otro lado, habilita una consola web a modo de interfaz llamada Futon que permite una cómoda depuración[9].

Su principal objetivo es operar en entornos distribuidos de datos orientados a aplicaciones web. Los cambios, al ser replicados constantemente, permiten un alto grado de fiabilidad. Para ello dispone de la habilidad realizar cambios offline. La ventaja de operar de esta forma, es ser más tolerante a fallos y permitir cambios offline que sean sincronizados en el momento que sea posible.

Por este motivo, la adopción de CouchDB se ha hecho muy popular en aplicaciones web basadas en HTML 5.

5.8.1. Propagación de señales en CouchDB

La interfaz (GUI) y el cliente FreeStation no se comunican directamente entre sí. Aunque es posible realizar una codificación directa, la configuración de Webkit para la interfaz HTML no puede enviar señales de forma cómoda al cliente FreeStation basado en GTK. El mecanismo base, es la ejecución y evaluación por retrollamadas de Webkit basadas en javascript. Para su recepción, el código javascript debe ser evaluado en el cliente FreeStation y analizado detallamente. Este enfoque hace complicado el mantenimiento y añadir nuevo código siendo poco escalable en el tiempo.

Por ello se adapta un enfoque de una aplicación intermediaria. En este enfoque se utilizan la emisión de señales a través de CouchDB. CouchDB escucha estas señales y las propaga como eventos en ambas direcciones[10].

[9] Proyecto CouchDB: `http://couchdb.apache.org`

[10] Couchbase Client Library Python 1.0:
`http://www.couchbase.com/docs/couchbase-sdk-python-1.0/index.html`

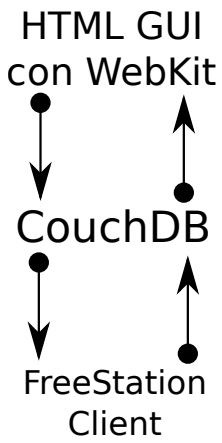


Figura 5.28: Propagación de eventos y señales en CouchDB

La figura 5.28 muestra el flujo de propagación de señales explicado anteriormente.

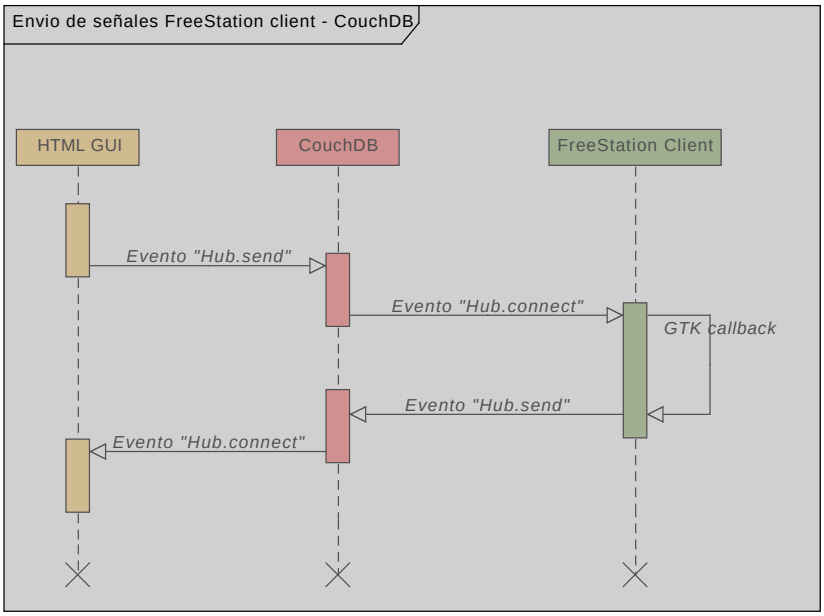


Figura 5.29: Diagrama secuencia entre componentes

El diagrama de secuencia explicado por la figura 5.31 muestra la especificación de llamadas entre componentes y las señales emitidas y recibidas. El componente Hub o concentrador sirve de mediador entre componentes. Las señales del hub como send() permiten enviar los datos y el tipo de señal. Estas a su vez son conectadas en el componente (ya sea en javascript para html, python en couchdb o python en GTK).

5.8.2. Diagrama de clases para CouchDB

Las clases de código que intervienen en el proceso son principalmente la interfaz GTK del cliente FreeStation que iniciara la incorporación e importación de un widget basado en CouchDB. Dicho widget conectara sus señales GTK para mantenerse a la espera de nuevos eventos. Posteriomente inicializará un wigdet browser que permite carga la composición de arranque de una interfaz HTML5 basada en elementos web.

En ese momento una instancia de CouchDB con un usuario único se creará de forma transparente en la aplicación y adoptará las señales que fueron conectadas anteriormente. Con propósitos de depuración, se iniciara un analizador (inspector) a modo consola web.

Paralelamente se pondrá a la escucha un widget DBus para otros widgets como MountDetector, para detectar cambios de hardware como dispositivos USB.

Una vez están preparados estos componentes, la interacción en la interfaz basada en HTML5 estará a la espera de eventos de usuario.

Una vez se produzcan como se ha comentado en el apartado 5.8.1 la clase Hub iniciará una factoría de señales en HubFactory y los datos podrán ser recibidos mediante la clase DataProvider. La interacción encapsulada de la clase Hub, permitirá conectar con la biblioteca Microfiber y la biblioteca javascript main.js de la interfaz HTML5.

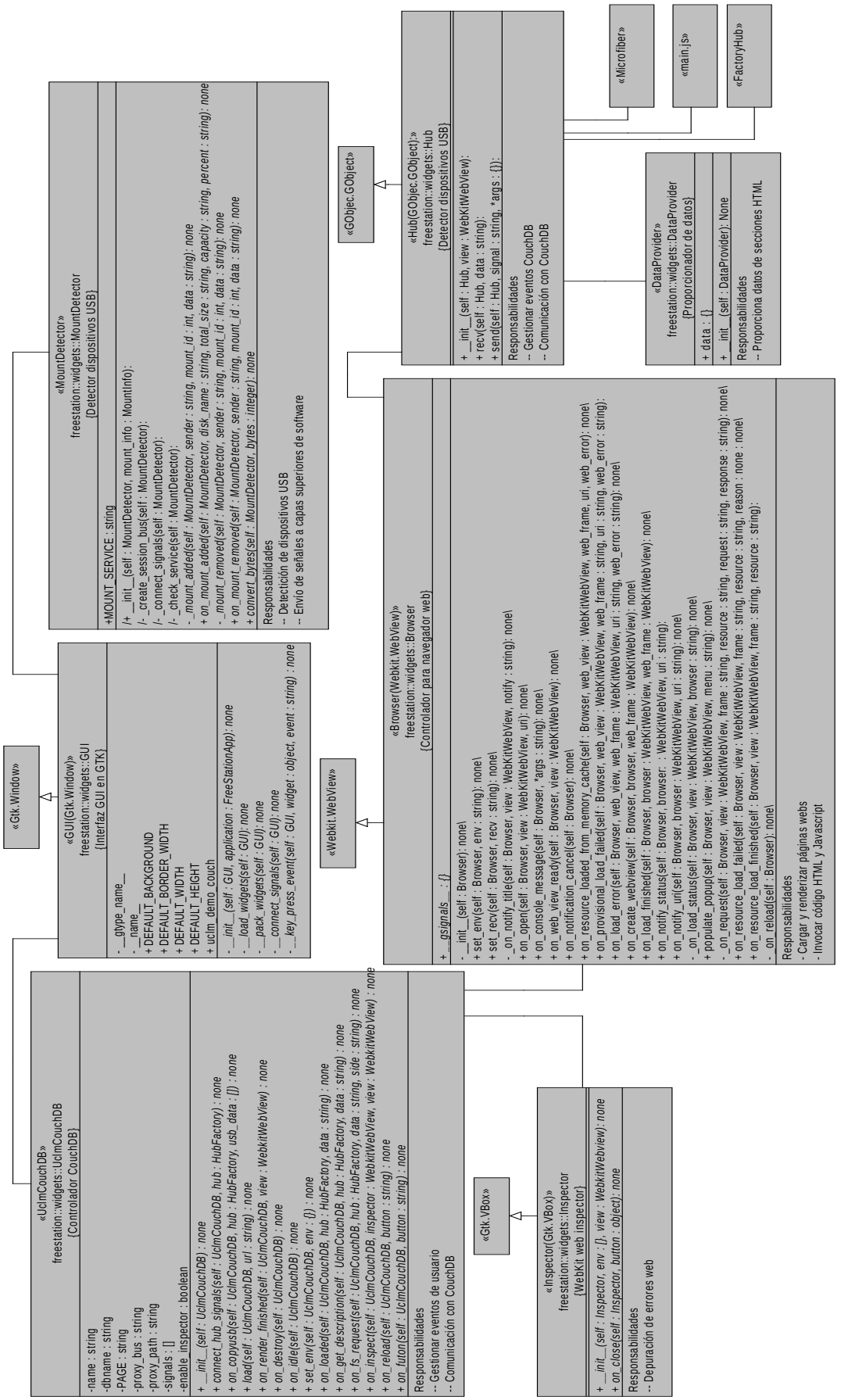


Figura 5.30: Diagrama de clases para CouchDB

Las ventajas de la utilización de CouchDB es la realización y maquetación de aplicaciones basadas en HTML5. El enriquecimiento de la interfaz abre más posibilidades de interacción con el usuario y el acercamiento a interfaces más naturales (NUI) (véase sección 3.4). Al contrario que los elementos GTK, permite una mayor personalización en la apariencia de la aplicación. El prototipado por tanto es mucho más rápido y permite esbozar la aplicación con poco esfuerzo para el desarrollador.

El enfoque orientado a eventos y señales permite abstraer por capas la aplicación y reutilizar código entre diferentes módulos o widgets.

El problema de este enfoque es conocer la latencia entre un cambio realizado por la interfaz, el procesado de CouchDB y su recepción por parte del cliente. En el peor escenario, el cliente también podría emitir una señal de vuelta a la interfaz a través de CouchDB, lo que completaría el ciclo de interacción.

Según se comentó en las sección 3.5 era crítico que la respuesta ante la interacción del usuario esté totalmente acotada, y en un tiempo mínimo para que la percepción del usuario fuera óptima.

Según la ejecución del fichero benchmark.py basado en el módulo cProfile de Python, el arranque en frío de aplicación conlleva 57106 llamadas de funcione primitivas. Estas son en su mayoría importaciones del módulo gnome instrospection de bindings. Esta ejecución toma un tiempo total de 1,267 segundos. Las funciones primitivas implicadas en freestation son 1756, tomando únicamente un tiempo de 0,145 segundos.

Cada evento y señal procesado de CouchDB conlleva un tiempo de procesado de 0,019 segundos.

Por tanto, para reducir la latencia del sistema, CouchDB se focaliza en la concurrencia. Para que el sistema sea lo suficientemente escalable, se deben tratar con tres principales tipos de problemas:

- Peticiones de lectura
- Peticiones de escritura
- Volumen de datos

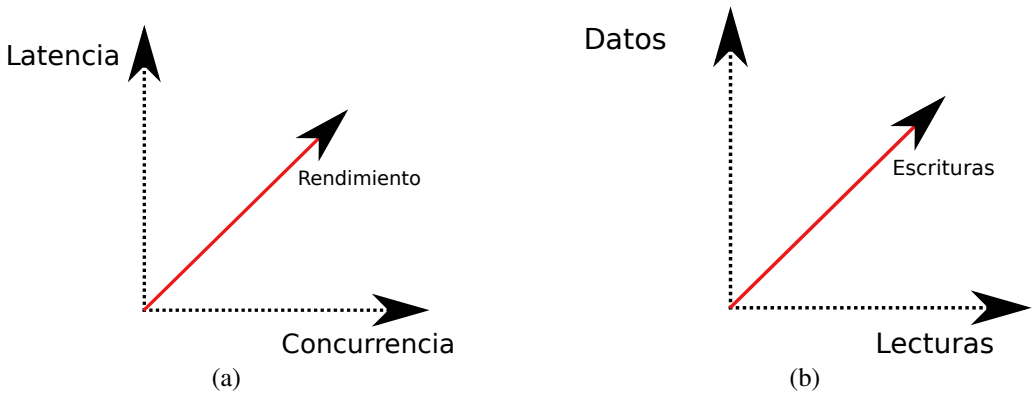


Figura 5.31: a) Relación entre latencia y concurrencia b) Relación entre datos y lecturas

La figura 5.31 muestra la relación explicada en CouchDB. En general el sistema será más fiable con un menor número de atributos entre las señales y cuando estas puedan paralerizarse. CouchDB es flexible en ese aspecto y permite construir bloques de datos para resolver un problema exacto.

De esta forma, las señales permiten acciones que interactuaran en flujo HTML emitido desde Javascript al motor de WebKit. El bucle de ejecución sera procesado con CouchDB y será recibido en una aplicación cliente basada en GTK como FreeStation.

Así, gracias a esta solución, se obtienen las siguientes ventajas frente a un desarrollo basado en aplicaciones de escritorio puras:

- Mayor facilidad en el diseño del interfaz.
- Total flexibilidad a la hora de modificar el interfaz “al vuelo en tiempo de ejecución”
- Posibilidad de reutilizar la amplia gama de componentes diseñados en HTML.
- La información es fácilmente organizable estructurando contenidos en JSON como formato estándar.
- Desarrollo totalmente multiplataforma, sin necesidad de desarrollar bindings propios
- Combinación de presentación y navegación de la aplicación en un mismo estándar basado en HTML 5.
- Simplicidad y coherencia con principios fundamentales para el desarrollo de una interfaz efectiva (véase sección 3.2.1).

5.8.2.1. API REST en CouchDB

La API REST de CouchDB es gestionada mediante el protocolo HTTP. Para ello utiliza un método de almacenamiento de información basado en documentos, es decir, una gestión documental. El atributo DocID identifica de forma única un documento.

Algunos ejemplos de acceso a documentos únicos:

Listado de código fuente 5.5: Ejemplo 1 CouchDB

```
1 http://localhost:5984/prueba/algun_id
```

Listado de código fuente 5.6: Ejemplo 2 CouchDB

```
1 http://localhost:5984/prueba/otro_id
```

Listado de código fuente 5.7: Ejemplo 3 CouchDB

```
1 http://localhost:5984/prueba/BA1F48C5418E4E68E5183D5BD1F06476
```

Las anteriores url mostradas apuntan al identificador del documento. De esta forma pueden accederse a documentos con una simple URL y mediante HTTP[11]..

Listado de código fuente 5.8: Petición HTTP

```
1 curl -X GET 'http://localhost:5984/prueba/algun_id?rev=3243'
```

Un documento CouchDB es simplemente un objeto JSON, con cualquier estructura o anidamiento posible. Algunos atributos como el número de revisión se guardan de forma adicional.

Listado de código fuente 5.9: Ejemplo Documento CouchDB

```
1  {
2   "_id":"freestation_widgets",
3   "_rev":"D1C946B7",
4   "load":true,
5   "maximize":false,
6   "allocation":[0, 0, 800, 600],
7   "wigets": [
8     {"Name":"MountInfo", "Enabled":True, "Depends":"MountUSBStorage"},
9     {"Name":"VideoArea", "Disabled":True} ]
10 }
```

[11] CouchDB HTTP document API: http://wiki.apache.org/couchdb/HTTP_Document_API

Los atributos que preceden por "'_'son reservados para uso de CouchDB.

5.8.3. Cliente ligero microfiber

Microfiber[12] es una biblioteca con el propósito de adaptador genérico para hacer peticiones HTTP con datos JSON en la API REST de CouchDB[13].

Es una API orientada a una gran cantidad de métodos excepcionales que permite realizar llamadas de forma sencilla, abstrayendo la complejidad de CouchDB.

Según la ejecución del fichero benchmark_microfiber.py la cantidad de lecturas/escrituras/borrados/guardados es aproximadamente de 140 por segundo, lo que garantiza una gestión eficiente de los recursos:

Listado de código fuente 5.10: Benchmark Microfiber

```
Benchmarking microfiber *** Python: 3.2.3, i686, Linux
  Saving 2000 documents in db 'test_benchmark_microfiber'...
    Seconds: 16.21
    Saves per second: 123.4
  Getting 2000 documents from db 'test_benchmark_microfiber'...
    Seconds: 7.91
    Gets per second: 253.0
  Deleting 2000 documents from db 'test_benchmark_microfiber'...
    Seconds: 17.73
    Deletes per second: 112.8
Total seconds: 41.84
Total ops per second: 143.4
```

Su principal ventaja y motivo de uso es la abstracción del capturar el entorno (enviroment) de CouchDB. El entorno se compone de la url de acceso, con los credenciales en base64 para autenticación básica en OAuth de HTTP. Por tanto microfiber lidia con los pequeños detalles de la conexión con CouchDB.

Otro punto a favor es que permite instancias por usuario (*per user single based*) o como sistema completo (*per system based*).

Por otro lado, Microfiber también permite llevar a cabo acciones a nivel de base de datos

[12]Proyecto Microfiber: `https://launchpad.net/microfiber`
[13]Documentación de Microfiber: `http://docs.novacut.com/microfiber/index.html`

para CouchDB usando la instancia del servidor CouchDB.

5.9. CASO DE EXPLOTACIÓN POI UCLM

Con el fin de ilustrar un ejemplo funcional, se desarrolló un caso de explotación basado en un POI para la UCLM siguiendo las guías de diseño de interfaces según la sección 3.2.1. En este caso de explotación personalizado, se pretende que los alumnos dispongan de un POI con fácil acceso para descargas de software libre, distribuciones de sistemas operativos, apuntes, transparencias u otros programas de utilidad según el proceso de distribución de POIs de la sección 3.2.

A través de un Widget llamado UclmDemoCouch se implementa una interfaz completamente basada en HTML5, CSS 3 y javascript que es renderizada por el motor web WebKit[14].

A diferencia de los widgets basados en GTK permite utilizar degradados css en la interfaz, border redondeados en elementos y potenciar un diseño líquido para distintas resoluciones. Todos los recursos utilizados pueden ser cargados localmente a través de CouchDB y mediante Microfiber comunicar la interfaz con CouchDB para obtener y recibir señales y datos en el cliente FreeStation, de esta forma se cumple con un patrón MVC como fue explicado en la sección 3.5.

En el caso de explotación, se permite la copia de software seleccionado a una unidad USB. Para su detección la interfaz se comunica con CouchDB para que el cliente FreeStation se ponga a la escucha y monitorice eventos de hardware mediante DBus.

[14]Biblioteca UserWebkit: `https://launchpad.net/userwebkit`

Todo el desarrollo se ejecuta en Python 3 mediante bindings a cada biblioteca necesitada.

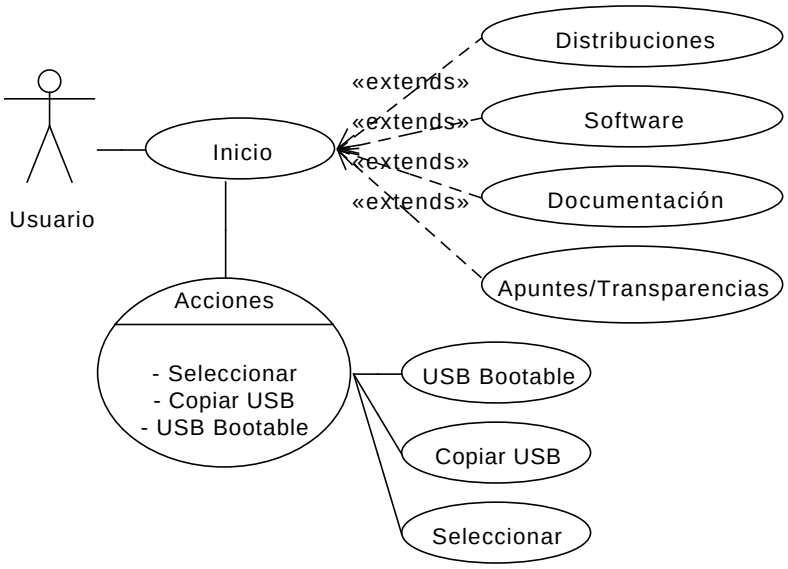


Figura 5.32: Diagrama de casos de uso para caso de explotación

En la figura 5.32 se detalla el diagrama de casos de usos para el caso de explotación con sus diferentes acciones implicadas.

5.9.1. Caso de explotación basado en GTK

El caso de explotación basado en GTK fue desarrollado inicialmente como un prototipo funcional de aplicación. Este era completo y ofrecía buen rendimiento y desempeño, pero la apariencia no resultaba demasiado óptima.

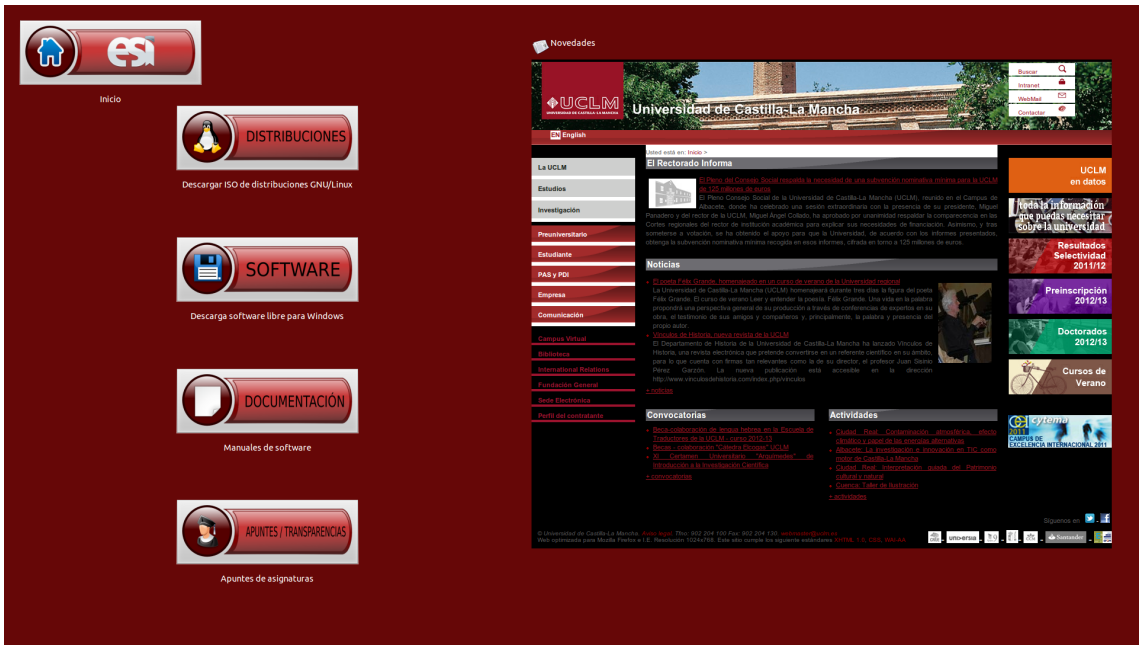


Figura 5.33: Caso de explotación basado en GTK - Vista inicial

La figura 5.33 muestra la interfaz de bienvenida del POI desarrollado como ejemplo para la UCLM. La interfaz no seguía algunos de los patrones NUI definidos en la sección 3.5. El theming en GTK era complicado y no permitía características avanzadas como bordes redondeados de cajas, degradados lineares en fondos o animaciones de interacción.

El fondo de los botones no admitía efectos de hover con opacidad y no podía establecerse como transparente. Si podía optarse por establecer el mismo color del fondo de la aplicación, pero el efecto botón seguía vigente. El diseño por su parte aunque utilizaba el sistema de cajas de GTK verticales y horizontales sin posicionamientos absolutos, no era muy adaptable a resoluciones diferentes, por lo que no disponía de un diseño líquido teniendo el programador la responsabilidad de adaptar los widgets a la resolución presente.

La documentación de GTK es bastante completa pero sufre un vacío en el theming siendo bastante complicado su desarrollo sin ejemplos avanzados. Por otro lado los bindings a python estan poco probados en dicha competencia y a menudo tienen errores de programación que no permiten realizar los cambios o no funcionan correctamente.

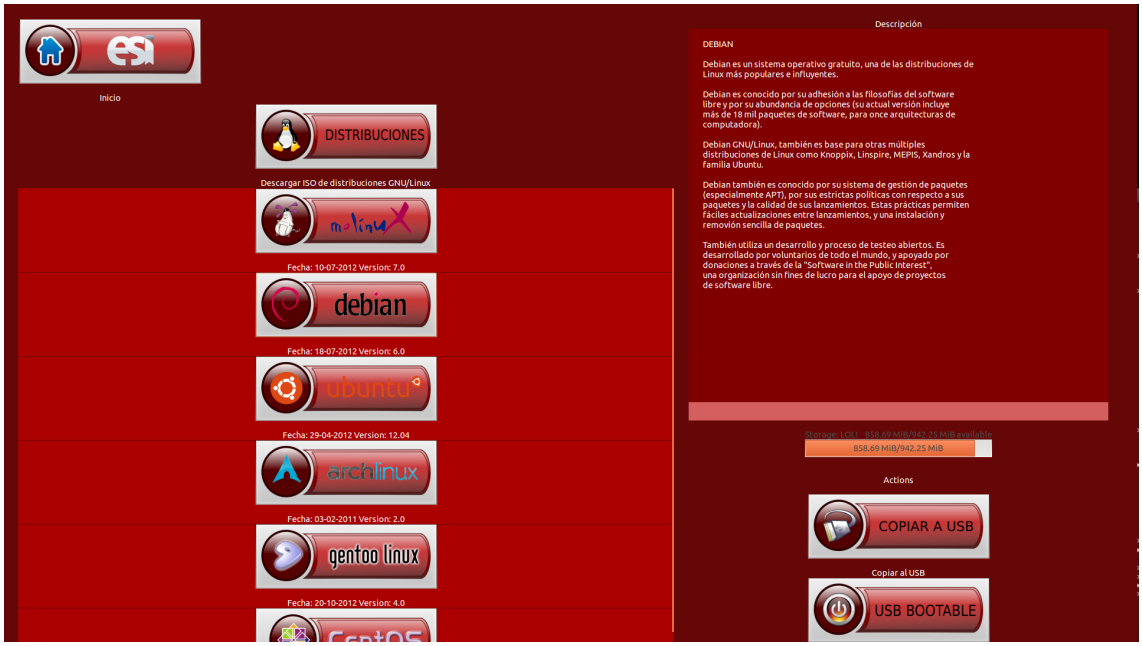


Figura 5.34: Caso de explotación basado en GTK - Vista de distribuciones

La vista de distribuciones de la figura 5.34 tenía problemas similares con las vistas de despegables en scroll y los botones no eran adaptables en el contenido. La posibilidad de incorporar estados a botones seleccionados era complicada y requería la inserción dinámica de widgets, destruyendo anteriores. La programación de la interfaz era complicada ya que el programador debe estar pendiente del modelo de widgets a mostrar y ocultar en cada interacción de la aplicación.

Por estas razones, se investigaron nuevas tecnologías para la generación dinámica de interfaces como fue explicado en la sección de antecedentes 3.5 y se intento aplicar un enfoque basado en CouchDB en combinación de WebKit. Este enfoque requería más investigación y esfuerzo que GTK pero prometía un gran cambio de resultados en la apariencia y mayor versatilidad en programación del mismo.

Los anteriores widgets basados en GTK se mantuvieron como presentación del esfuerzo realizado y como futuro aprovechamiento para otras implementaciones de widgets que todavía basaran su diseño en GTK.

5.9.2. Caso de explotación basado en CouchDB

El caso de explotación basado en CouchDB tuvo inicialmente una curva de aprendizaje bastante elevada. Se basaba en una tecnología reciente y poco usada por empresas o programadores independientes.

Gracias al contacto de varios desarrolladores, conversaciones en IRC y consultas directas al código fuente, la comprensión de la tecnología fue más rápida adquiriendo un mayor nivel de la misma.

El resultado, fue la realización de una aplicación notablemente diseñada con mejor apariencia y más adaptada a las reglas NUI.

Figura 5.35: Caso de explotación basado en CouchDB - Vista inicial

La figura 5.35 muestra la pantalla de bienvenida basada en estas nuevas tecnologías, que en comparación con la figura 5.33 es notablemente superior.

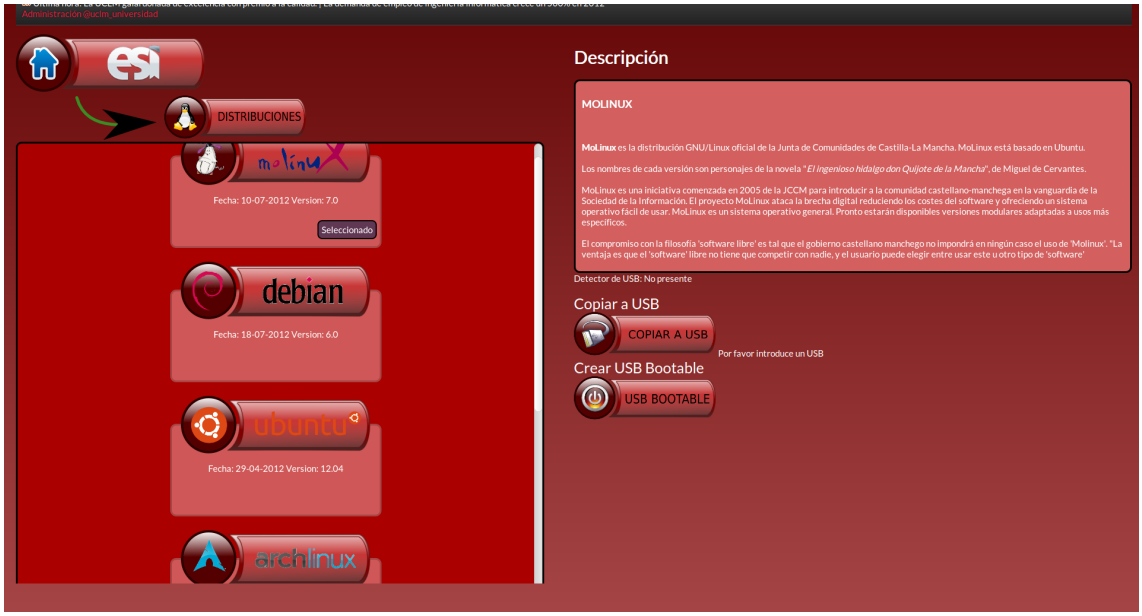


Figura 5.36: Caso de explotación basado en CouchDB - Vista de distribuciones

De la misma forma, la figura 5.36 muestra que la vista de distribuciones mejoró su apariencia y fue posible añadir efectos de fade *in/fade out* que no son apreciables en las capturas de pantalla, pero si en ejecución de la aplicación. Los degradados lineales, border redondeados, opacidades, etc mejoran ampliamente la visualización del efecto resultado final.

En el desarrollo se han tenido en cuenta los principios de diseño NUI se la sección 3.5 como por ejemplo:

- **Contexto**: aprovechar el contexto como principio fundamental de diseño, en este caso, educativo y con entorno similar a la paleta de diseño de la UCLM.
- **Interacciones**: aprovechar interacciones NUI donde el usuario tendría un beneficio de interfaz. Los efectos *in/fade out*, opacidad, notificación de acciones y remarcados de interacción contribuyen en este punto probando mejoras en las mecánicas fundamentales de las interacciones primarias.
- **Menos es más**: proponer un diseño simple sin pérdida de información y con casos de uso muy concretos.

La incorporación de nuevas características como la internalización (que no ha sido completada por falta de tiempo) deberían resultar triviales de añadir sobre el caso de explotación actual. En definitiva, la acción de añadir nuevas características a FreeStation resulta sencilla así como adaptar el interfaz a las necesidades particulares de cada organización. Este enfoque abre multitud de puertas abiertas al desarrollo y mejoras del proyecto.

Capítulo 6

6 EVOLUCIÓN Y COSTES

En el presente capítulo se describirán las diferentes fases empleadas en el desarrollo de *FreeStation*, especificando los hitos característicos de cada una, y aportando datos relacionados con la complejidad y el coste temporal de cada una.

Igualmente se aportará información del rendimiento (profiling) del sistema en diferentes situaciones, así como algunas comparativas con aplicaciones demostrativas que no han sido desarrolladas utilizando *FreeStation*.

6.1. FASES E ITERACIONES

En este apartado se describe la aplicación del método ha conllevado una serie de prácticas descritas por la XP para llegar a la solución.

Las primeras fases tenían como objetivo maximizar el valor del software desarrollado como prototipo y pruebas de concepto iniciales para siguientes iteraciones. Según los resultados producidos por iteración se han retroalimentado las siguientes iteraciones. De esta forma, la valoración de costes y riesgos por cada iteración ha podido evaluarse según las tareas completas y las planificadas por completar.

A continuación se describen las tareas realizadas y planificadas por fase.

6.1.1. Fase 1 - Implementación prototipo

En la primera fase se desarrollo una versión inicial como prototipo de cliente y servidor usando sockets en Python. La implementación tiene una duración aproximada de semana y media, donde se implementa una clase servidor que es iniciada como primera instancia, siendo los parámetros por defecto el puerto y nodo donde es iniciado. El puerto por defecto es el 2626 y el nodo o host es localhost.

El método run permite ejecutar el servidor, que inicializa las estructuras poniéndose a la escucha de las peticiones de los clientes. Para ello utiliza el método $accept\_new\_connection()$. Cada petición será tratada como una consulta o query por el servidor procesándose según el tipo establecido.

Dichas consultas son simples mensajes de información o descarga de archivos de texto o binarios. Por ejemplo el método del servidor $get\_info()$ muestra un mensaje al cliente con el nombre del nodo y el puerto en el que esta ejecutándose. Internamente el mensaje enviado por el cliente es «info» a través de un socket.

El cliente queda a la escucha del mensaje que enviara el servidor y lo muestra en consola o bien si se ejecuta en modo interfaz gráfica. En la clase cliente se observan dos atributos similares como el puerto y el nodo.

Mediante el método $request$ se pasara la información o tipo de mensaje que debe atender el servidor para actuar en consecuencia. El método run, ejecutara el cliente en consola, que también puede ser integrado en una interfaz gráfica.

Se realiza un pequeño diagrama de clases que ilustre el estado actual del proyecto:

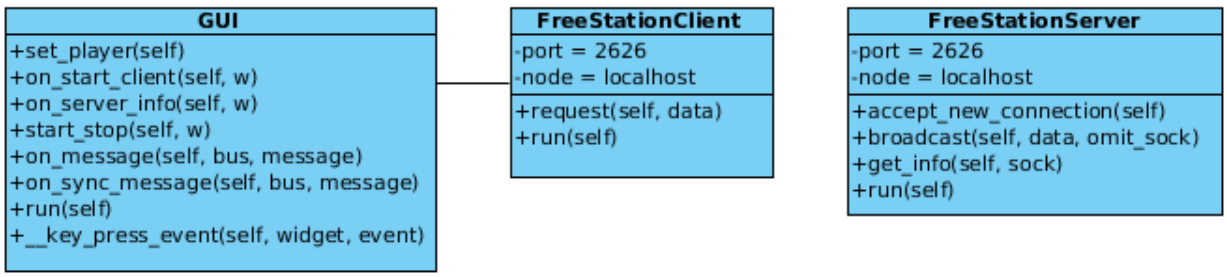


Figura 6.1: Diagrama de clases

El primer prototipo es funcional y pretende demostrar la intención de comunicación de datos entre cliente y servidor.

6.1.2. Fase 2 - Migración a ICE

Debido a la complejidad incremental de usar sockets para un desarrollo más avanzado, se desecho la idea en una tutoría de proyecto en favor de utilizar una tecnología de distribución de objetos distribuidos. Tras varias comparativas, se decidió usar ICE por su versatilidad y comodidades en el desarrollo con Python y otras ventajas explicadas anteriormente (véase apartado 4.5.5.2).

Se dedica tiempo de estudio a ICE leyendo la documentación oficial[Spr12] y formas de implementación en Python. Se realizan pequeñas pruebas y prototipos de ejemplos con el manual de David Vallejo sobre ICE[Val06]. Se acuerda establecer un listado de componentes necesarios para la aplicación.

6.1.3. Fase 3 - Implementación widgets

Se implementa una lista de componentes (widgets) tentativa para el cliente de *FreeStation*. Esta lista es inicialmente un prototipo de widgets que podrían resultar útiles en siguientes iteraciones de proyecto. Se planifica en posteriores iteraciones estudiar la viabilidad de implementar los widgets más detalladamente o con características más avanzadas.

Algunos de los widgets:

- *1 - **MountDetector**: Detecta la presencia de unidades USB (montaje/desmontaje) y recopila la información del nombre de la unidad, espacio disponible y total.
- *2 - **MountInfo**: Configura una interfaz a partir de los datos de MountDetector con una barra de progreso para mostrar el espacio disponible/total.
- *3 - **VideoArea**: Carga un vídeo en reproducción continua y se activa en pantalla completa cuando FreeStation pasa a modo idle.

- 4 - **SliderArea**: Paneles informativos deslizantes con información para banners, transparencias, etc.
- *5 - **Browser**: habilita la navegación de una pagina web externa o documento html local.
- *6 - **BrowserView**: renderiza templates según la vista (MVC: Modelo Vista Controlador) y trata con la lógica de la información proporcionada por el componente Browser.
- *7 - **GUI**: Inicializa la aplicación y lee los componentes habilitados para su carga, lanzándolos posteriormente.
- 8 - **CategoryArea**: Zona de información de listas verticales de categorías de información.
- *9 - **MenuActionsArea**: menú formado por flechas de anterior/siguiente/inicio para interactuar con las posibles zonas
- 10 - **ItemArea**: Muestra la información sobre algún elemento (puede ser software, documento, etc) con elementos como logo, descripción, dependencia, título.
- *11 - **LogoArea**: Carga un logo por defecto para la institución, universidad, escuela, oficina de turismo, ayuntamiento, etc.
- *12 - **TitleDisplay**: Carga un título central por defecto para la institución, universidad, escuela, oficina de turismo, ayuntamiento, etc
- 13 -**ClockArea**: Renderiza la hora actual.

Nota: Los componentes marcados con * delante son funcionales e implementados en dicha fase.

La implementación en esta fase conlleva alrededor de tres semanas al tener un gran componente de desarrollo e investigación por cada widget planteado.

6.1.4. Fase 4 - Clases abstractas

Puesto que la implementación esta hecha en Python, se intentan utilizar clases abstractas como sugerencia, aunque no tiene mucha aplicación ya que cada componente tiene su lógica por separado y estos heredan (especializaciones) de las clases necesarias, o tienen asociaciones y dependencias con otros componentes (por ejemplo *MountDetector<->MountInfo<->Gui<->FreeStationApp*).

Aparte se investiga que las clases abstractas en python, son pseudo-implementadas con *RaiseError* o *Exception*, pero no tienen una sintaxis definida propia (como Java o PHP), sino como metaclases. Se desecha la idea de implementar un patrón de herencia basado en clases abtractas, aunque no se descarta la posibilidad de usar alguna cuando sea más necesario.

6.1.5. Fase 5 - Migración de GTK2 a GTK3

El desarrollo del código para interfaces empezó en las primeras iteraciones basado en la biblioteca GTK2. En el proceso, la mayoría del software relacionado con la plataforma GTK empezó a basarse en la nueva biblioteca GTK3 (véase 4.5.5.1).

A corto y medio plazo la estimación era que podrían producirse problemas de compatibilidades y abandono de la plataforma GTK2 o pequeños fallos que no fueran corregidos. La ventaja de utilizar la nueva biblioteca en GTK3 era una mayor potencia y versatilidad de desarrollo, pero podrían producirse fallos no detectados.

Se opta por migrar todo el código anterior de widgets de GTK2 a GTK3, ya que para el binding de Python existe un gran aumento de rendimiento y por lo general es más fácil de escribir el código en la nueva versión.

Este proceso dura aproximadamente una semana y media y requiere de consultas en listas de correo de desarrollo de GTK y preguntas a desarrolladores en IRC para las opciones más avanzadas en la migración. En la migración se utilizado el tutorial de migración de GTK (véase sección [Gno12]).

6.1.6. Fase 6 - Detección USB

En esta iteración se trata de completar, optimizar y añadir mas funcionalidad a los widgets encargados de la detección de USB. Se estudia el posible uso de DBus (véase 4.5.5.3) como principal elección para comunicación de eventos por mensajes de nuevos dispositivos hardware USB.

Para la implementación se desarrollan dos widgets. El primero llamado *MountDetector* detecta nuevos dispositivos de almacenamiento USB y recopila los datos en crudo. Una vez dispone de los datos, los envía a un segundo widget llamado *MountInfo* donde se procesan los datos para impresión y tratamiento de la interfaz. Este widget sera el encargado de tratar con la interfaz principal de la aplicación cliente.

La implementación resulta tediosa al no disponer de una documentación lo suficientemente clara y ejemplos basados en python. A través de ejemplos en C y visualización de implementaciones de otros programas se realiza la implementación. La migración del código GTK2 a GTK3, resulta satisfactoria, aunque algunas primitivas nativas de la biblioteca cambian radicalmente.

6.1.7. Fase 7 - Problemas con Gstreamer

Para la realización e implementación de un widget de vídeo se empezó el desarrollo del widget *VideoArea*. Este widget sería capaz de mostrar un vídeo, reproducirlo en bucle o pasar a modo idle en pantalla completa cuando no existiera interacción con la aplicación principal.

Se decidió usar la biblioteca *Gstreamer* (véase 4.5.5.4) en su versión 0.10. Se implemento la reproducción sin bucle y sin modo idle en GTK2. Posteriormente en la migración del código a GTK3, se detectó que los mensajes emitidos en la finalización del flujo de vídeo no eran emitidos, lo que imposibilitaba implementar la característica de integrar el widget de la ventana de vídeo con otros widgets en la misma ventana.

Esto resultaba en la aparición de dos ventanas por separado y como consecuencia el modo completo de pantalla pasaba a modo normal.

Se intento llegar a una solución intermedia donde mezclar el widget de GTK2 funcional con el resto de widgets portados a GTK3, pero seguían produciéndose conflictos con la biblioteca interna GObject.

Se reporto un bug en proyecto Gnome (véase Bug #631901[1]) con el fin de que los desarrolladores tuvieran consciencia del problema y se presentara una arreglo o solución próxima. La iteración se finaliza a la espera del arreglo para una próxima versión 0.11 de Gstreamer (considerada 1.0) que pueda solucionarlo. El problema principal esta en la migración interna de gst-python (binding de python para Gstreamer) al nuevo Gnome Instrospection (GI) de GTK3.

6.1.8. Fase 8 - POC de ICE

Tras la investigación y formación en la documentación de ICE se decide realizar un desarrollo commo prueba de concepto (POC) para probar las llamadas y la ejecución de ICE.

Se desarrolla una clase cliente llamada *FreeStationClient* y otra backend llamada *FreeStationServer* como fue explicado en la sección de arquitectura 5.6.2.1. Se realiza un vídeo ilustrativo como ejemplo de funcionamiento de las conexiones y pruebas de rendimiento.

En el vídeo se trata la aplicación en ejecución en modo pantalla completa para el cliente con algunos widgets se hace la prueba de interacción insertando un USB de 1 GB para su detección. Se acuerda una nueva reunión para visualizar la demo en directo, aumentar/modificar la lista de widgets, enfocar las siguientes tareas a realizar y comentar cualquier otra duda.

Se propone realizar pequeños artículos en el blog personal para ir comentando el progreso y tener una memoria de las iteraciones.

[1] Bug #631901 Gnome: `https://bugzilla.gnome.org/show_bug.cgi?id=631901`

6.1.9. Fase 9 - Reemplazar Webkit por Gecko

Para la visualización de páginas webs embebidas como widget en la aplicación cliente se debería elegir un motor de renderizado de páginas webs.

Las alternativas disponibles bajo un binding de python eran:

- **Gecko**: el motor de renderizado usado por Firefox. Usa la biblioteca gtkmozembed y estaba basado para funcionamiento con GTK.
- **WebKit**: el motor de renderizado de Google Chrome, Safari, etc. Estaba adaptado a GTK mediante la biblioteca WebKitGTK+.

Respecto a gtkmozembed el proyecto no tenía un ritmo de actualización activo, ya que no disponía de actualizaciones desde 2005. Desde las versiones de GTK 2.2X no tenía buen soporte y no era perfectamente funcional. Para GTK3 no existía ninguna actualización que permitiera ejecutar su código sin fallos.

Debido a ello, se descarto usar gtkmozembed por su bajo soporte. Por lo tanto, WebKit basado para Gnome con el nombre de biblioteca *WebKitGTK*[2] tenia un soporte bastante decente con actualizaciones frecuentes. El soporte ofrecido por Ubuntu parece consolidarse al utilizar la biblioteca en varias aplicaciones de la distribución.

Se propone la investigación del funcionamiento de algunas aplicaciones como el Centro de Software de Ubuntu para ver el funcionamiento de la biblioteca y principales usos.

6.1.10. Fase 10 - Watchdog y Webkit

En la mejora del *Watchdog* se implementa finalmente siendo totalmente funcional (véase sección 5.5.6). Se observa un problema en el reinicio de la aplicación que parece interferir en el widget de Webkit.

[2]WebKitGTK: `https://live.gnome.org/WebKitGtk`

El problema reside en la biblioteca ya que realiza un lock (bloqueo) interno para renderizar en el hilo principal. Esto provoca un mal renderizado del widget de Webkit exclusivamente al utilizar el GDK lock

Se observan referencias al problema en listas de correo[3] y fuentes oficiales de webkit[4]. Como conclusión se deduce que Webkit no libera correctamente el lock establecido y provoca que el objeto del widget del Browser se quede congelado.

Al ser previsto para solucionar en próximas versiones se decide seguir con la implementación actual ya que es la única alternativa para renderizar páginas webs como widget con threads. Este enfoque es necesario por el watchdog que lanza las aplicaciones como hilos.

El watchdog en las primeras iteraciones se implemento con el módulo *multiprocessing* y la clase *Process*. De esta forma se realizaban nuevos procesos hijos a partir de forks del watchdog. Era funcional esta implementación, pero al reiniciar la aplicación, existían problemas con punteros de GTK que no eran liberados desde el binding de Python. Esto provocaba errores como que las X eran un recurso temporalmente en uso.

Se intento investigar el problema bajando el código de *XLib* y se detecto que la función que procesaba el binding era la función *XOpenDisplay* y devolvía un XIO error. Debido a la complicación del binding en GTK escrito en C++ y tener que realizar compilaciones con demasiadas dependencias del entorno GNOME se desecho al salirse de los objetivos de la iteración y extensión del proyecto.

De esta forma se optó por el módulo de threadings e implementarlo como hilos con un funcionamiento correcto, excepto por el objeto de Webkit comentado anteriormente. También se intento previamente bajar el repositorio actual de código de Webkit (3.5 GB) y se encontró la parte de desarrollo de GTK para WebKit[5]

[3]The GDK Lock: `http://pvanhoof.be/blog/index.php/2007/08/04/the-gdk-lock`

[4]What supposed to hold the GDK in calls from WebKit/gtk/WebCoreSupport: `https://lists.webkit.org/pipermail/webkit-gtk/2011-August/000651.html`

[5]Webkit source para GTK:
`http://svn.webkit.org/repository/webkit/releases/\WebKitGTK/webkit-1.6.1/Source/WebKit2/`

La extensión era bastante grande para detectar el XIO error y se abandonó aún a pesar de consultar algunos manuales para modificaciones de GTK[6].

6.1.11. Fase 11 - Punto de entrada con GTKApplication

GtkApplication[7] es una clase para administrar todos los aspectos importantes de una aplicación GTK. Esta ideada para reforzar el concepto de encajar en la idea de aplicación modelo.

GtkApplication realiza la inicialización GTK+ por defecto, asegura la unicidad de ejecución de la aplicación, administración de sesiones y provee integración con scripts y escritorio. Permite exportar acciones y menú a ventanas de alto nivel en el ciclo de desarrollo de la aplicación. La versión inicial desarrollada usaba *GtkApplication* para aprovechar las ventajas expuestas.

Debido a su novedad e implementación reciente en GTK 3 surgieron problemas en el desarrollo. En concreto, la aplicación mediante *GTKApplication* creaba un objeto que recibía señales. Una de las señales permite interactuar con la aplicación una vez que finaliza. En este caso la señal invocada por el método $release()$ debería terminar la aplicación, pero dejaba el thread como zombie. Esto es debido a que la clase *GTKApplication* actúa como un constructor/destructor, pero no liberaba correctamente la última instancia.

Tras investigar el problema, parece un problema conocido por los desarrolladores de GTK[8]. En el momento de la iteración, desde GTK no estaban seguros de la especificación final de la implementación y con previsiones de avanzarla o terminarla en Gnome 3.2[9]. Debido a estos problemas, se aplaza y desecha la implementación actual con *GtkApplication* y se implementa como un thread lanzado por la clase watchdog.

[6] HackingGtk: `http://trac.webkit.org/wiki/HackingGtk`

[7] GtkApplication:
`http://developer.gnome.org/glibmm/unstable/\classGio_1_1Application.html`

[8] Bug 637445 - Finish Gtk::Application:
`https://bugzilla.gnome.org/show_bug.cgi?id=637445#c17`

[9] Gtk::Application punted to gtkmm 3.2:
`http://old.nabble.com/Gtk::Application-punted-\to-gtkmm-3.2-td31220771.html`

6.1.12. Fase 12 - Widgets posicionados relative/absolute

Se investiga la forma de crear widgets mediante posicionamientos relativos y absolutos. Se mira el funcionamiento de Ogre3D[10] y su implementación como referencia. Se observa las características ofrecidas en Glade[11] y por GTK para realizar los posicionamientos mediante ficheros XML.

6.1.13. Fase 13 - Carga por XML

En esta iteración se concreta y modela la especificación de carga para los widgets. Se define una especificación basada en XML. Este enfoque permite utilizar las definiciones para manejar eficientemente tipos de datos complejos, etiquetados o ocurrencias de widgets.

Se establece una primera definición con las propiedades y elementos que pueden disponer como base los widgets. Para la carga de widgets mediante XML se implementa una clase llamada *WidgetLoader* vista en la sección de arquitectura 5.6.1.

Esta delega la tareas de carga un parseador XML (XMLParser) y detecta los widgets con sus respectivas configuraciones. Una vez que los datos son procesados, se cargan en la estructura del WidgetLoader permitiendo controlar los fallos de análisis o duplicidades.

6.1.14. Fase 14 - Arranque del backend

Se prueba un posible arranque del backend por medio del un servicio en el arranque por init.d. Se desecha la idea ya que requiere intervención mediante consola de comandos y se piensa una mejor solución.

Se implementa un servicio para arrancar en el servidor los servicios de ICE y que respondan a un frontend en PHP. Esto evita la interacción por consola y una gran comodidad y facilidad de gestión desde el frontend web.

[10] Ogre3D: `http://www.ogre3d.org`
[11] Glade `http://glade.gnome.org`

La interfaz web permite conectar al cliente y crear el servidor en ejecución. La interfaz web del servidor se planifica para mejor desarrollo, con el fin de que el caso de explotación pueda hacer un ejemplo simple de los widgets y utilice distintos datos cargados como posicionamientos relativos o absolutos, ancho y alto y configuraciones personalizadas.

6.1.15. Fase 15 - PyUnit testing

Para las pruebas del proyecto se empezó realizando un pequeño conjunto de pruebas de test unitarios. Dado que la mayor parte del proyecto estaba escrito en Python se eligió el framework PyUnit. En su inicio se cubrieron los test para secciones más críticas y consolidadas como *Watchdog*, *FreeStationApp* y *BrowserView*. El posterior objetivo no era hacer un completo code coverage, pero si bastante avanzado.

En posteriores iteraciones se valoró cubrir las clases *WidgetLoader* y *XMLParser* por su repercusión de errores en la detección de archivos XML y asegurar un buen funcionamiento de la aplicación en la parte cliente.

Debido a la falta de tiempo se pospone la escritura de artículos técnicos en el blog personal prioritizando el desarrollo y documentación del proyecto. Asimismo los problemas surgidos con GTK y derivados se retrasan algunas partes del desarrollo investigando soluciones.

6.1.16. Fase 16 - Mejora del POC de ICE

Se mejora el diseño de la aplicación cliente y servidor con la conexión ICE. Se hacen llamadas mediante especificaciones de ficheros .ice (slice), mejora del tratamiento de excepciones y reescritura del código para mejorar la encapsulación.

La configuración del cliente y servidor se hace en tiempo de ejecución y se dedica la gran parte del tiempo de iteración a comprender mejor el funcionamiento interno de las funciones de ICE.

6.1.17. Fase 17 - Mejora widget VideoArea en GTK3

En anteriores iteraciones el widget de *VideoArea* fue desarrollado pero se necesitaban nuevas funcionalidades como reproducción en bucle y empotrado del widget en ventanas GTK.

En esta iteración se consiguió hacer funcionar la reproducción en bucle en GTK3 debido a que parte del bug que fue reportado funcionaba mejor con la versión 0.11 de GStreamer. También se modificó la disposición del widget y permitía el empotrado en una ventana independiente que podía asociarse con otros widgets. La reproducción de ficheros locales e incluso url externas era funcional. Se realizan pruebas con el trailer de Sintel[12][13] en formato WebM[14].

Como información para la reproducción mejorada en Gstreamer se usan documentación y tutoriales procedentes de la Guía de Novacut[15] para migraciones Gstreamer 1.0[16]

Por otro lado, se confirma la liberación de GStreamer 1.0 para principios de Octubre como fecha tentativa. Esta nueva versión sera empaquetada por defecto en todas las versiones de Ubuntu 12.10 y posteriores lo que asegura un buen funcionamiento de la aplicación siguiendo la especificación actual.

6.1.18. Fase 18 - Intento de solución para WebKit

En esta iteración se propone solucionar el problema con los threads de WebKit. Tras varios días de intentos infructuosos se obtienen un resultado como conclusión. Webkit no permite el uso con threads fuera del hilo principal.

El motivo es que su implementación no es "thread safeż solo puede usarse desde el hilo principal de la aplicación. El problema afecta en el desarrollo del watchdog desde que es lanzado como aplicación principal en un thread para *FreeStationApp* y esta a su vez lanza los hilos correspondientes en la aplicación, uno de ellos el correspondiente a WebKit.

[12] Sintel: `http://www.sintel.org`
[13] Sintel trailer 480p: `http://docs.gstreamer.com/media/sintel_trailer-480p.webm`
[14] WebM Project: `http://www.webmproject.org`
[15] Novacut: `http://novacut.com`
[16] Guía Novacut para GStreamer:
https://wiki.ubuntu.com/Novacut/GStreamer1.0

Puesto que no esta dentro de uso del thread principal, la primera vez que es iniciado WebKit funciona sin problemas, pero posteriores reinicios con el Watchdog no habilitarán el renderizado de Webkit. Para ello es necesario reiniciar por completo el watchdog. El problema de Webkit viene de la cancelación del render en uno de sus cuatro estados posibles al renderizar una aplicación web: *provisional->commited->finished->rendered*. En el momento del render, el GDK lock hace imposible utilizarlo y se hacer un rollback del estado del render pintando un widget en negro vacío.

La confirmación de esta situación viene del mensaje:
`http://markmail.org/message/4dwft6s6g6ptavj6`

En concreto, se establece el problema con las líneas:

> Webkit is not thread-safe. External processing on a secondary thread is OK, but any calls into the DOM will have to happen on the main thread.

De esta forma los contenidos se llegan a cargar, pero no se renderiza el DOM ya que debe realizarse en el thread principal[17]

6.1.19. Fase 19 - Refactorizado de widget para propiedades

Para la carga del *WidgetLoader* se debían detectar propiedades que el widget incorporará en el formato de la etiqueta $< properties >$ del fichero XML.

Se refactoriza el código del mismo para que se admitan propiedades del alto, borde y algunas propiedades más en algunos widgets básicos como el *LogoArea*, *TitleArea* y *FeedArea*.

[17] GDK_Threads:
`http://www.yolinux.com/TUTORIALS/GDK_Threads.html`

6.1.20. Fase 20 - Implementación de widget lector RSS

Como consecuencia de la anterior iteración se hace necesaria la implementación de un widget que realice lecturas de suscripciones RSS. En el widget se utiliza la biblioteca feedparser de python y se añade una imagen de feed como icono.

Se configura con lector de título RSS, personalización de cada entrada y una ventana con scroll. El lector tiene en cuenta las entidades HTML para limpiar caracteres no adecuados en su representación.

6.1.21. Fase 21 - Theming de GTK

Tras la realización de un conjunto bastante amplio de widgets para *FreeStation* se decide investigar el theming con GTK para mejorar la apariencia de la aplicación y crear un caso de explotación con una paleta de colores y diseño similar a la UCLM.

Los principales problemas que surgen con el theming es que no existe mucha información al respecto ni ejemplos avanzados. Se decide investigar en el código fuente de GTK para realizar algunas propiedades que no estaban completamente claras y por otro lado en el binding de GTK generado para python.

El theming es posible realizar a partir de ficheros de estilos CSS básicos. Pero la mayoría de los modelos renderizados con GTK son cuadrados y no se puede aplicar bordes redondeados de forma sencilla. Se prueba con la propiedad border-radius pero no es efectiva. Buscando ejemplos similares en código fuente del centro de software Ubuntu se realizarán algunos cambios significativos, pero no resulta en la apariencia esperada.

La apariencia final tiene como resultados los expuestos en la sección de arquitectura 5.9.1.

6.1.22. Fase 22 - Investigación de tecnologías basadas en CouchDB

Puesto que en la anterior iteración el resultado no era el esperado se investigan nuevas tecnologías punteras que permitan una mejor apariencia de la aplicación y efectos de animación. En una sesión presencial se comenta la posibilidad de usar CouchDB con una interfaz web en Webkit que comunique con GTK. Se dedica una semana y media para comprender la tecnología y realizar pequeños ejemplos funcionales.

6.1.23. Fase 23 - Migración del proyecto completo a CouchDB

Los resultados en los ejemplos funcionales son muy prometedores y aunque resta poco tiempo de desarrollo para finalización del proyecto se decide reimplementarlo completamente (aprovechando los módulos posibles) para que estos sean renderizados con CouchDB. Esto conlleva varias semanas de trabajo y reescribir parte de la documentación y test existentes.

6.1.24. Fase 24 - Mejoras en la implementación

Se finaliza la implementación de la parte de copia de USB en CouchDB y cambios en la presentación de los contenidos (flechas de botones en jerarquía) y mejorado el aspecto visual de la parte de noticias.

Se termina la implementación de la parte de copia a usb y realizan algunos cambios de presentación (añadida una flecha en los botones de jerarquía) y mejorada la sección de noticias. Se consigue acoplar perfectamente todas las bibliotecas antiguas en freestation y ya no es necesario ejecutarlo como proyecto aparte del basado en GTK. Se integra la funcionalidad completa en un nuevo widget llamado *UclmDemoCouch*

6.1.25. Fase 25 - Reescritura a Python 3

El cambio a CouchDB también implica la reescritura del código de python 2.7 a python 2.7, salvo el cliente en Ice que no dispone de soporte para Python 3 y se ejecuta de forma independiente en Python 2.7

6.1.26. Fase 26 - Creación de USB Bootable

La característica a implementar del USB Bootable da algunos problemas al no conocer una documentación existente por línea de comandos y durante la migración de CouchDB queda relegada hasta poder investigar más la tecnología. Se valora la posibilidad de no incluirlo en la implementación por falta de tiempo.

Se mejoran los mensajes de error en la copia de USB si no existe espacio en disco suficiente y se propone como implementación futura que previamente se realice un cálculo de los ficheros para valorar el tamaño final a copiar. Se habilita un flag o bandera en el código para habilitar el caso de explotación en GTK3 o CouchDB.

6.1.27. Fase 27 - Contribuciones a bibliotecas de CouchDB

Tras la implementación completa del proyecto en *CouchDB* y el uso de una biblioteca de ejemplo llamada *userwebkit*[18] se decide incorporar la biblioteca integrada y adaptarla a las necesidades específicas de *FreeStation*. Esta biblioteca es una variedad de framework en *Webkit* para la aplicación Novacut[19].

Puesto que la biblioteca esta orientada a la aplicación *Novacut* incorpora por defecto un recolector de imágenes del sistema llamado *dmedia*[20]. Por este motivo se refactoriza la biblioteca userwebkit para desacoplar las funcionalidades e integrar sólo las partes necesarias en FreeStation. En concreto se incluyen las mejoras en el widget *BrowserView* que se desarrollo para el uso de *WebKit*.

En el proceso de refactorización se consigue crear una funcionalidad que permite interceptar la consola de mensajes de WebKit. Aunque no era objetivo de FreeStation si complementaba su funcionalidad. Capturando los mensajes de la consola se podían tratar bien los errores y mensajes lanzados por WebKit.

Como resultado se realizó un parche de la funcionalidad para que el autor de las bibliotecas novacut, dmedia y userwebkit pudiera incorporarlo en su proyecto.

[18] userwebkit: `https://launchpad.net/userwebkit`
[19] Novacut: `https://launchpad.net/novacut`
[20] Dmedia: `https://launchpad.net/dmedia`

El bug #1023770[21] fue reportado y adjuntado el parche[22]. Su autor agradecio enormemente la contribución en una mención en Google+[23]

6.1.28. Fase 28 - Prueba de funcionalidades en CouchDB

Como prueba de la funcionalidad se comprobó que los eventos de CouchDB se emitían y capturaban bien, aunque en el proceso de detecto que no permitía reevaluar contenido javascript en las peticiones, por lo que todo el código javascript debería ser previamente incluido y activar funciones con eventos.

Las peticiones Ajax no eran *crossdomain* por lo que no se permitía obtener datos de otros servidores que no fueran localhost. Tampoco se podían cargan imágenes en local, sólo permitía imágenes externas. Se solucionó este problema añadiendo una directiva en webkit para carga de archivos locales.

Se procede a rellenar el contenido de las restantes secciones del ejemplo del caso de explotación y se realiza un vídeo para la evaluación del tutor.

6.1.29. Fase 29 - Desarrollo USB Bootable con UnetBootIn

Tras analizar la funcionalidad de *unetbootin* se valora la posibilidad de hacer un wrapper o adaptador del código C++ a python.

Posteriormente en la documentación del proyecto (wiki)[24] se descubren funcionalidades por línea de comandos que permiten generar USB bootables a partir de ficheros .iso.

[21] Bug #1023770 Launchpad `https://bugs.launchpad.net/userwebkit/+bug/1023770`

[22] Merge proposal(parche): `https://code.launchpad.net/~shakaran/userwebkit/webkit-console/+merge/118470`

[23] Mención Google+: `https://plus.google.com/u/0/114471118004229223857/posts/7287aAdYNup`

[24] unetbootin: `http://sourceforge.net/apps/trac/unetbootin/wiki/commands`

Un ejemplo de comando sería:

Listado de código fuente 6.1: Copia de fichero .iso con unetbootin

```
unetbootin method=diskimage isofile="/path/file.iso"
```

Debido a la falta de tiempo, se pospone y se decide dedicar el resto de tiempo disponible a terminar y mejorar la documentación del proyecto.

6.2. RECURSOS Y COSTES

En este apartado e enumeran y describen el conjunto de diferentes recursos principales y la especificación de los mismos, tanto temporales como económicos, en conjunto a herramientas de apoyo y análisis para el proyecto de fin de carrera.

Se evalúan sus características principales y se detalla la planificación y presupuesto inicial y final (estimados) del mismo.

6.2.1. Presupuesto y planificación iniciales

6.2.1.1. Presupuesto inicial

Se parte de una planificación inicial desde Septiembre de 2011, elaborada al comienzo del proyecto basada en modelos, conceptos teóricos y la experiencia en el desarrollo de proyectos similares. Se estima inicialmente el proyecto para una duración de 4 meses para entrega en Febrero de 2012.

Durante el desarrollo la incorporación de nuevas características y ampliación de requisitos llevan a prolongar la duración del proyecto durante 4 meses más hasta Junio de 2012. Como resultado de problemas en el desarrollo e incorporación de nuevas características se amplía como fecha definitiva y no prorrogable hasta Septiembre de 2012.

Por tanto la duración del mismo conlleva un año de trabajo.

6.2.1.2. Coste de personal

En este apartado, se detalla el proceso operacional seguido para la obtención del salario por hora de trabajo de los diferentes roles participantes en el proyecto. Los resultados obtenidos se presentan, posteriormente, en forma de tabla.

Estimación horas laborables	
Días laborables / mes	20
Horas trabajadas / día	6 horas
Horas trabajadas / mes	120 horas

Cuadro 6.1: Estimación horas laborables por día y mes

En el presupuesto inicial se estimo un salario neto por persona al mes (el mismo para todos los roles) de 1200 € como ingeniero junior.

Aplicando los costes de recursos humanos:

Costes por recurso humano	
IRPF (20 % sobre el salario neto)	240,00
Seguridad Social (40 % sobre salario neto + IRPF)	576,00
Salario bruto por persona / mes: Salario neto + IRPF + Seguridad Social	2016
Meses laborables	11 meses
Horas en un año	1760 horas

Cuadro 6.2: Estimación horas laborables por día y mes

Esta tabla refleja por tanto un coste bruto por persona y hora igual al salario bruto por persona y año / horas en un año. Total 16,03 €/hora.

Suponiendo un salario bruto por persona y año (suponemos 14 pagas): 28224 €

6.2.1.3. Equipos informáticos

Como fue explicado en la sección fue necesario un coste Hardware en equipos y servidores.

Las siguientes tablas detallas una estimación del coste de los mismos:

Costes por equipos	
Acer Aspire Ethos 5943G (laptop)	1299,00 €
Acer Aspire One (netbook)	299,00 €
Compaq Presario C700	469,00 €
HP 620	699,00 €
Sobremesa	999,00 €

Cuadro 6.3: Costes por equipos

El total de la tabla 6.3 asciende a 3765 €.

Costes servidores	
Servidor dedicado (OpenVZ, 512 MB RAM)	27 € / mes
Servidor dedicado (VMWare, 8 GB RAM)	150 € / mes

Cuadro 6.4: Costes por servidores

El total de la tabla 6.4 para un año asciende a 2124 €.

6.2.1.4. Licencias

Costes por licencias	
GTK	0,00 € / año
VMWare	225,63€ / año
OpenVZ	0 € / año
Ubuntu 12.10	0 € /año
CentOS 6.8	0 € /año
GStreamer	0 € /año
Zero C ICe	0 € /año
Gimp	0 € /año
Otros	0 € /año

Cuadro 6.5: Costes por licencias

El total de la tabla 6.5 asciende a 225,63 €.

6.2.1.5. Material fungible

Descripción	Coste (Euros)
Material de impresión	150,00
Material de oficina	55,00

Cuadro 6.6: Coste material fungible

El total de la tabla 6.6 asciende a 205,00 €.

6.2.1.6. Resumen de costes del proyecto

El coste final del presupuesto inicial del proyecto, partiendo de los datos obtenidos en los apartados anteriores serían.

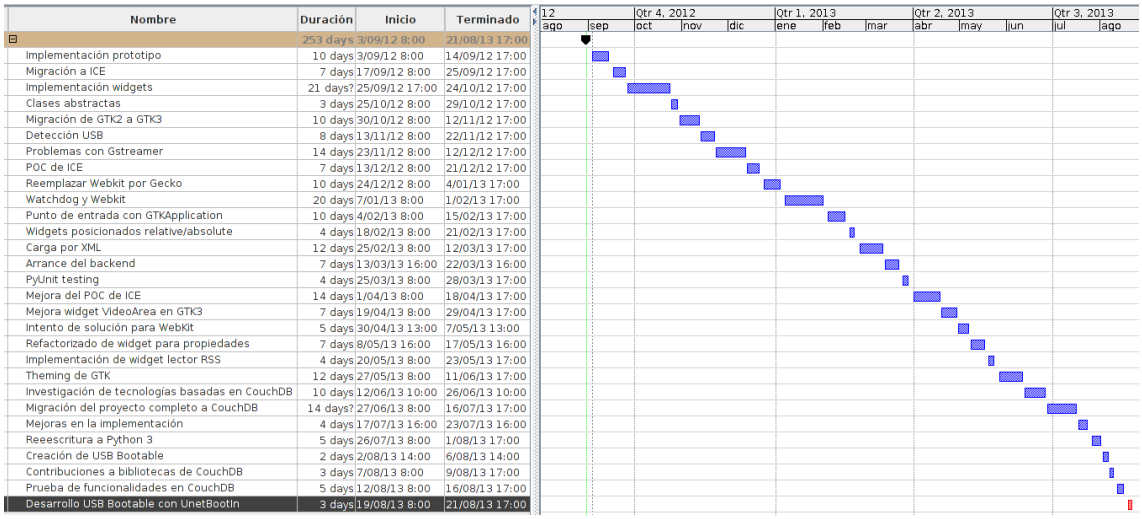
Descripción	Coste (Euros)
Recursos humanos	28224,00 €
Coste hardware	3765
Coste servidores	2124 €
Licencias	225,63 €
Material fungible	205,00 €

Cuadro 6.7: Resumen costes del proyecto

El Total de presupuesto inicial mostrado por la tabla 6.7 asciende a 34453,63 €.

6.2.2. Planificación final del proyecto

Para contabilizar y representar las iteraciones y el tiempo de desarrollo. Se creo un diagrama Gantt correspondiente a la planificación inicial y final que se realizó para el desarrollo del sistema *FreeStation*.



Nombre	Duración	Inicio	Terminado
	253 days	3/09/12 8:00	21/08/13 17:00
Implementación prototipo	10 days	3/09/12 8:00	14/09/12 17:00
Migración a ICE	7 days	17/09/12 8:00	25/09/12 17:00
Implementación widgets	21 days?	25/09/12 17:00	24/10/12 17:00
Clases abstractas	3 days	25/10/12 8:00	29/10/12 17:00
Migración de GTK2 a GTK3	10 days	30/10/12 8:00	12/11/12 17:00
Detección USB	8 days	13/11/12 8:00	22/11/12 17:00
Problemas con Gstreamer	14 days	23/11/12 8:00	12/12/12 17:00
POC de ICE	7 days	13/12/12 8:00	21/12/12 17:00
Reemplazar Webkit por Gecko	10 days	24/12/12 8:00	4/01/13 17:00
Watchdog y Webkit	20 days	7/01/13 8:00	1/02/13 17:00
Punto de entrada con GTKApplication	10 days	4/02/13 8:00	15/02/13 17:00
Widgets posicionados relative/absolute	4 days	18/02/13 8:00	21/02/13 17:00
Carga por XML	12 days	25/02/13 8:00	12/03/13 17:00
Arrance del backend	7 days	13/03/13 16:00	22/03/13 16:00
PyUnit testing	4 days	25/03/13 8:00	28/03/13 17:00
Mejora del POC de ICE	14 days	1/04/13 8:00	18/04/13 17:00
Mejora widget VideoArea en GTK3	7 days	19/04/13 8:00	29/04/13 17:00
Intento de solución para WebKit	5 days	30/04/13 13:00	7/05/13 13:00
Refactorizado de widget para propiedades	7 days	8/05/13 16:00	17/05/13 16:00
Implementación de widget lector RSS	4 days	20/05/13 8:00	23/05/13 17:00
Theming de GTK	12 days	27/05/13 8:00	11/06/13 17:00
Investigación de tecnologías basadas en CouchDB	10 days	12/06/13 10:00	26/06/13 10:00
Migración del proyecto completo a CouchDB	14 days?	27/06/13 8:00	16/07/13 17:00
Mejoras en la implementación	4 days	17/07/13 16:00	23/07/13 16:00
Reeescritura a Python 3	5 days	26/07/13 8:00	1/08/13 17:00
Creación de USB Bootable	2 days	2/08/13 14:00	6/08/13 14:00
Contribuciones a bibliotecas de CouchDB	3 days	7/08/13 8:00	9/08/13 17:00
Prueba de funcionalidades en CouchDB	5 days	12/08/13 8:00	16/08/13 17:00
Desarrollo USB Bootable con UnetBootIn	3 days	19/08/13 8:00	21/08/13 17:00

Figura 6.2: Diagrama de Gantt

La figura 6.2 muestra la planificación desde principios de Septiembre de 2011 a finales de Agosto de 2012.

Nota: se ha utilizado la herramienta OpenProj[25] para realizar la estimación de costes y diagrama de Gantt.

La estimación total se ha hecho siguiendo horarios laborales de 8:00 a 17:00 de Lunes a Viernes con una duración total de 253 días laborables en aproximadamente unas 50 semanas de trabajo.

6.2.2.1. Coste final del proyecto

Siguiendo la figura 6.2 y haciendo un recuento del tiempo empleado en el desarrollo del proyecto se han necesitado 253 días laborables, a un precio estimado de 40 € la hora de ingeniero durante 6 horas al día, 5 días a la semana.

Total presupuesto: 60720 € de recursos humanos. Con una media mensual de 5060 €/mes. Esto supone la contratación de recursos humanos para un año de trabajo de un ingeniero cualificado incluyendo costes de IRPF y derivados[26].

[25] OpenProj: `http:\openproj.org`

[26] En estos cálculos no se ha tenido en cuenta el tiempo dedicado a la documentación del PFC, de unas 300 horas aproximadamente.

6.2.2.2. Estadísticas de actividad en el repositorio

Para el control de versiones se ha utilizado un repositorio GIT en local.

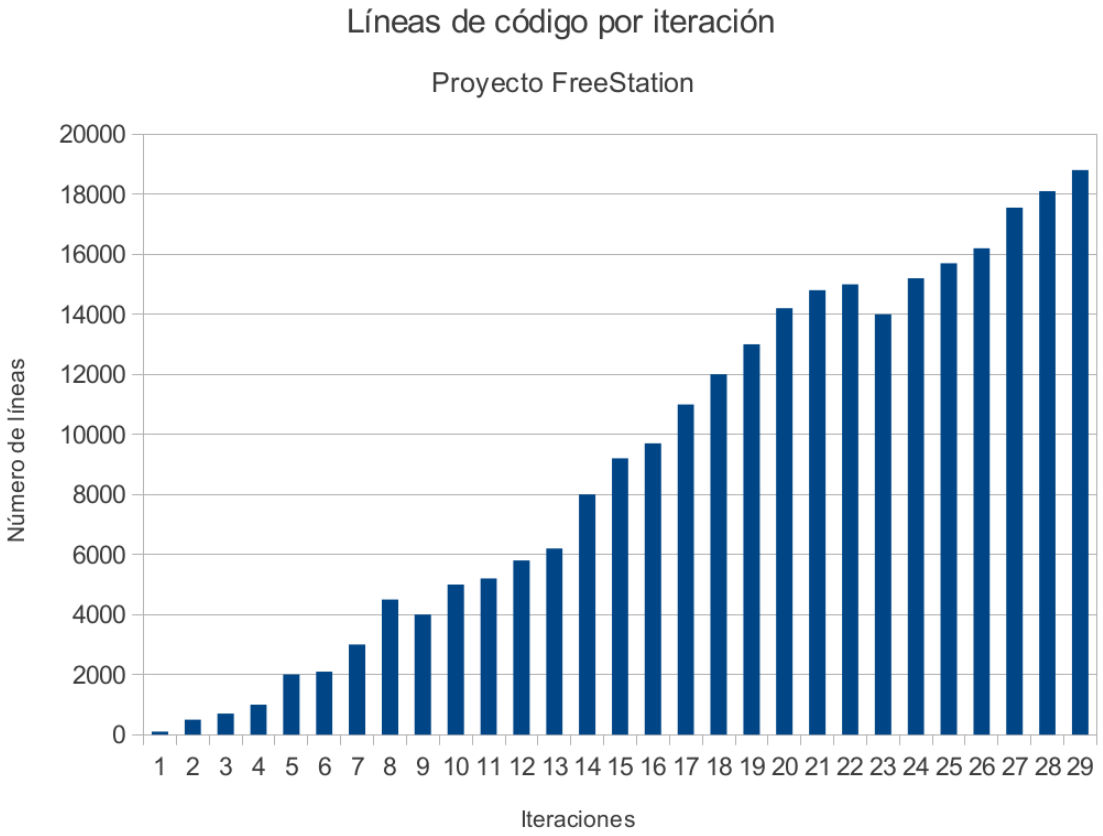


Figura 6.3: Líneas de código por iteración

En la figura 6.3 se muestra un gráfico de la evolución de las líneas de código fuente durante toda la etapa de desarrollo.

Se ha utilizado la herramienta cloc y sloccount para contabilizar las líneas de código fuente, asociado a los lenguajes en los que está programado *FreeStation*. Hay que tener en cuenta que dicha herramienta contabiliza todas las líneas de código del proyecto, también las autogeneradas.

Proyecto FreeStation web (Frontend y Backend Servidor)

Idiomas agrupados por lenguaje dominante primero				
Lenguaje	Archivos	Espacios en blanco	Comentarios	Líneas de código
Javascript	19	2901	2177	10495
HTML	18	505	5	2797
CSS	14	507	141	2405
PHP	27	402	930	1999
Python	8	367	348	876
make	1	24	5	124
SQL	2	37	57	82
Bourne Shell	2	22	6	48
XML	1	0	0	10
Total	92	4765	3709	18836

Cuadro 6.8: Tabla líneas de código FreeStation web

Las estimaciones de SLOC son:

Development Effort Estimate, Person-Years (Person-Months) = 0.43 (5.19)
(Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months) = 0.39 (4.67)
(Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule) = 1.11
Total Estimated Cost to Develop = $58,410$
$(averagesalary =$56,286/year.

Proyecto FreeStation client (Frontend y Backend cliente)

Idiomas agrupados por lenguaje dominante primero				
Lenguaje	Archivos	Espacios en blanco	Comentarios	Líneas de código
XML	7	44	30416	158519
Python	79	1971	3584	4664
Javascript	5	261	400	1249
HTML	2	16	1	141
CSS	2	27	13	123
Bourne Shell	2	7	5	29
Total	97	2326	34419	164725

Cuadro 6.9: Tabla líneas de código FreeStation client

Las estimaciones de SLOC son:

Development Effort Estimate, Person-Years (Person-Months) = 41.46 (497.46)
(Basic COCOMO model, Person-Months = 2.4 * (KSLOC**1.05))
Schedule Estimate, Years (Months) = 2.21 (26.47)
(Basic COCOMO model, Months = 2.5 * (person-months**0.38))
Estimated Average Number of Developers (Effort/Schedule) = 18.80
Total Estimated Cost to Develop = $5,600,037$
($averagesalary$ =56,286/year, overhead = 2.40).

6.3. RESULTADOS

En esta sección se describen los resultados. Se muestran los elementos más relevantes obtenidos al realizar pruebas con entornos reales. También se realiza un estudio del rendimiento de la aplicación cuando a ésta se le conectan diferente número de clientes.

6.3.1. Resultados con test unitarios

Una comparación de los resultados obtenidos de los test unitarios son plenamente satisfactorios. Todos los test pasan correctamente y han contribuido a realizar un mejor desarrollo detectando fallos en la implementación. La batería de pruebas no es completa pero si prueba las partes críticas de la aplicación.

6.3.2. Profiling

Profiling es como se conoce en ingeniería del software a la medida y análisis de rendimiento de una aplicación de forma dinámica, es decir, utilizando una ejecución concreta de dicha aplicación. El objetivo es determinar qué partes del código son cuellos de botella (consumen más recursos), y por tanto, son más susceptibles de ser optimizadas.

En el profiling de *FreeStation*, se ha utilizado el módulo de python cProfile, herramienta libre para GNU/Linux.

Para realizar el análisis es necesario ejecutar el programa benchmark.py y este ofrecerá los resultados al igual que otros test de rendimiento como el programa benchmark_microfiber.py para rendimiento en microfiber.

El profiler genera un documento a modo de informe del porcentaje de uso de CPU de cada una de las funciones individuales de la ejecución de la aplicación. En la sección de arquitectura 5.8.2 se procedió a explicar un análisis del mismo.

Capítulo 7

7 CONCLUSIONES Y PROPUESTAS

7.1. OBJETIVOS ALCANZADOS

Partiendo de los requerimientos iniciales, el enfoque aplicado propuesto en este proyecto de fin de carrera alcanza los resultados previstos.

La aplicación tiene una funcionalidad completa en base a la especificación de objetivos inicial. Se ha desarrollado por completo el concepto origen del proyecto de fin de carrera y se han utilizado todas las tecnologías inicialmente previstas. Esto ha sido posible gracias a la independencia de un modelo específico y la reusabilidad de bibliotecas y aplicaciones existentes.

En el lado cliente se han finalizado los principales módulos esenciales como Watchdog, tratamiento de un conjunto suficientemente amplio de widgets y el módulo de comunicación con el servidor.

El sistema permite la generación dinámica de interfaces complejas según los requisitos especificados por el cliente de forma precisa. Los datos son fácilmente exportables con configuraciones basadas en un esquema XML.

Gracias a este enfoque se consigue un alto nivel de adaptabilidad, permitiendo la futura realización de mejoras internas, adaptándose al desarrollo de bibliotecas y aplicaciones de terceros. Por otro lado, el componente servidor funciona perfectamente con la conexión de backend a clientes mediante ICE y utiliza el frontend web basado en PHP facilitando la administración web.

Otro de los aspectos a destacar es la incorporación de nuevas tecnologías como CouchDB (ver sección 5.8), microfiber y widgets basados en HTML 5.

El desarrollo del proyecto no sólo supondrá un beneficio directo para otros desarrolladores con similares propuestas, sino para usuarios que usen la aplicación a diario. En cualquier caso, es preciso tener en cuenta que en este proyecto de fin de carrera se han realizado casos funcionales de explotación a modo de demostración del mismo. *FreeStation* se ha desarrollado teniendo en cuenta su posición como herramienta útil, que permita el manejo y la gestión de grandes volúmenes de información.

Como parte de su objetivo se han tomado medidas de simplificación en tareas complejas para el usuario facilitando el proceso de interacción del mismo a través del diseño de interfaces naturales (NUI) (ver sección 3.4). Gran parte de estas interfaces son generadas dinámicamente siguiendo una especificación definida por la aplicación.

Todo los objetivos y subobjetivos han sido cumplidos adecuadamente. Una de las partes clave en *FreeStation* es la adaptabilidad de para los requisitos de una organización. La robustez del sistema permite la recuperación de errores mediante el subsistema de *Watchdog* visto en la sección 5.5.6.

Las tecnologías que se utilizan en el proyecto de fin de carrera ayudan a homogeneizar el software a través del diseño modular del sistema. Su versatilidad permite la reutilización de componentes. El código de *FreeStation* se basa en buenos principios de diseño basados en el uso de patrones implementando varios de ellos (ver sección 5.6). El sistema basado en programación distribuida permite cumplir los objetivos de un software actualizable y distribuido usando el framework ICE (ver sección 4.5.5.2).

Por último, *FreeStation* basa su desarrollo en software y estándares libres lo que permite posible continuación a otras personas interesadas.

7.2. PROPUESTAS DE TRABAJO FUTURO

El desarrollo de FreeStation ha completado una arquitectura completa para la definición dinámica de POIs específicos para la distribución de software. Sin embargo, el proyecto tiene grandes ambiciones futuras y se proponen futuras propuestas que permitan mejorar y adaptar las herramientas a nuevos desarrollos o incluso adaptarlas a las características específicas de un equipo de trabajo.

A continuación se enumeran algunas de las líneas de trabajo futuras propuestas:

- **Ampliación de widgets**:
  El desarrollo de widgets actuales puede ser ampliado y ofrecer un gran catálogo de widgets adicionales. Explotar la posibilidad de funcionalidades basadas en 3D con widgets basados en OpenGL e investigar widgets más útiles para el público general. Los widgets basados en sonidos de música ambientes o en combinación con efectos visuales pueden ser útiles para el modo idle de un POI. En definitiva, se pretende buscar formas de mejorar los widgets actuales y añadir widgets novedosos. Como ayuda se pueden realizar encuestas a organizaciones para conocer sus necesidades prioritarias en sus POIS particulares. Estimación de tiempo: 2-3 meses

- **Sustitución del almacenamiento de datos para widgets**:
  El modelo de almacenamiento de datos utilizado es XML. Éste podría ser sustituido por una base de datos. El patrón MVC permite modificar el modelo de datos sin afectar al resto de componentes. La información podría ser guardada en una base de datos sqlite debido a la poca necesidad de usuarios concurrentes en un mismo POI. Es de suma importancia la gestión que realiza con los datos y por tanto debe asegurarse las mismas condiciones de fiabilidad y tiempo de procesado. Estimación de tiempo: 1-2 meses

- **Incorporar un caso de explotación con localización de idiomas**:
  Como se comentó en la sección 3.3 la incorporación de un selector de idiomas en un POI beneficia mucho su adaptación y permite llegar a un mayor número de personas. La realización de un widget o una adaptación sobre HTML4 y CouchDB (véase sección 5.8) sería modular para reutilización en otros POIs. Es necesario buscar un equilibrio en el número de idiomas ofrecidos al público del usuario. La estimación de 3 a 5 idiomas en la mayor parte de los casos podría ser util. Estimación: 1-2 meses.

7.3. CONCLUSIÓN PERSONAL

La realización del proyecto de fin de carrera me ha permitido llevar a cabo un proyecto real e investigación de nuevas tecnologías y diferentes enfoques de resolución de problemas.

El tiempo destinado ha sobrepasado el inicialmente requerido, pero ello me ha permitido desarrollar mejores conceptos y soluciones a través de iteraciones progresivas en el mismo.

De la realización del proyecto se desprende el gran potencial que disponen algunas nuevas tecnologías emergentes, en particular basadas en el ámbito web y en combinación con las aplicaciones de escritorio regulares.

Gracias a que la gran mayoría del software utilizado este proyecto ha sido basado en software libre se ha podido estudiar profundamente su funcionamiento y incorporar mejoras y contribuciones cuando ha sido posible a la comunidad.

El coste económico del proyecto es viable incluso para pequeñas instituciones o empresas que requieran en un futuro el uso de las aplicaciones desarrolladas. La labor de desarrollo afrontada compensa con la ganancia en productividad que puede llegar a retornar a los usuarios.

Las aproximaciones de prototipos y posteriores desarrollos fueron los más óptimos e inmediatos para comenzar a esbozar el proyecto, su ciclo posterior de desarrollo ha tenido un mayor esfuerzo, pero ha permitido llevar a cabo una labor proyecto completo de ingeniería resolviendo problemas.

Como conclusión final podemos afirmar que este proyecto satisface las especificaciones iniciales, pero deja abierto un amplio abanico de ampliaciones e implantaciones futuras.

7 Bibliografía y Referencias

[AnJ10] J. Chris Anderson, Jan Lehnardt, Noah Slater,
CouchDB: The Definitive Guide. Time to Relax
O'Reilly Media, Enero 2010.
`http://shop.oreilly.com/product/9780596155902.do` `https://github.com/oreilly/couchdb-guide` `http://guide.couchdb.org`

[Asl09] Matthew Aslett,
Red Hat's organic growth opportunities, 451 CAOS Theory, A blog for the enterprise open source community
`http://blogs.the451group.com/opensource/2009/08/26/red-hats-organic-growth-opportunities/`

[Bar09] Barandiarán Gordo, Laura,
Generación automática de interfaces gráficas de usuario para una herramienta CASE de toma de requisitos
Universidad Carlos III de Madrid. Departamento de Informática, 2009.
`http://e-archivo.uc3m.es/handle/10016/6133`

[Bau10] Eric Bauer,
Design for Reliability: Information and Computer-Based Systems
Wiley-IEEE, 2010.

[Bec00] Kent Beck,
Planning Extreme Programming
Addison-Wesley Professional, Octubre 2000.

[Bec01] Kent Beck,
Extreme Programming Explained: embrace change
Addison-Wesley, 2001.

[Dea98] Nathaniel Dean,
Robust communication networks: interconnection and survivability
November 18-20, 1998, DIMACS Center, Volumen 53.

[Fis00] Fischer, G,
User Modeling in Human-Computer Interaction. User Modeling and UserAdapted Interaction)
Kluwer Academic Publishers, 2000.

[Fou05] Benoit Foucher,
Grid Computing with IceGrid. Connections
Diciembre 2005.

[GHe06] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides - Addison Wesley (GoF- Gang of Four)
Design Patterns. Elements of Reusable Object-Oriented Software
Addison-Wesley Professional, Noviembre, 1994).

[GoB87] Gould, J. D., Boies, S. J., Levy, S., Richards, J. T. and Schoonard, J.
The Olympic Message System: a test of behavioural principles of system design
Communications of the ACM, páginas 758-769, 1987.

[Gno12] Gnome,
Migrating from GTK+ 2.x to GTK+ 3)
2012.
`http://developer.gnome.org/gtk3/3.5/gtk-migrating-2-to-3.html`

[Hel00] Gilbert Held,
Server Management (Best Practices)
Auerbach Publications; First edition, March 22, 2000.

[Hen06] Michi Henning,
Teach Yourself IceGrid in 10 Minutes. Connections
Noviembre 2006.

[Hud10] Jillian "JK" Hudson
The UX Progression of the American Airlines' Kiosk Interface
July 1, 2010.
`http://www.jkhudson.com/AACaseStudy_jkhudson.pdf`

[IEE05] IEEE Computer Society,
IEEE Standard for Learning Technology - Extensible Markup Language (XML) Schema Definition Language Binding for Learning Object Metadata)
Standard 1484.12.3, New York. 2005.

[Ind12] Indeed,
Windows, linux, mac Job Trends, Indeed
`http://www.indeed.com/jobtrends?q=windows,linux,mac&relative=1&relative=1`

[Lan10] Opsi, Lanrev,
Software Distribution: Shovelware, Software as a Service, Mobility-As-A-Service, Saas Platform, Aspectj, Software Deployment,
General Books LLC, 2010.

[LiB11] Linoff, G.S. and Berry, M.J.,
Data Mining Techniques: For Marketing, Sales, and Customer Relationship Management, 2011.

[LoM99] Víctor López Jaquero, Francisco Montero, José Pascual Molina, Pascual González,
Interfaces de Usuario Inteligentes: Pasado, Presente y Futuro
UCLM, 2006.
`http://www.dsi.uclm.es/personal/VictorManuelLopez/mipagina/archivos/Interaccion2006.pdf`

[Mag07] Maguire M.C
A Review of User-Interface Design Guidelines for Public Information Kiosk Systems
HUSAT Research Institute, 2007.
`http://ui4all.ics.forth.gr/UI4ALL-97/maguire.pdf`

[Mar11] David J. Marchette,
Computer intrusion detection and network monitoring: a statistical viewpoint 2011.

[McM12] Robert McMillan,
Red Hat Becomes Open Source's First $1 Billion Baby, Wired Enterprise
`http://www.wired.com/wiredenterprise/2012/03/red-hat/all/1`

[New10] Mark Newman,
Networks: An Introduction)
Oxford University Press, USA; First edition, May 20, 2010.

[ReJ12] Remo Suppi Boldrito, Josep Jorba Esteve,
GNU/Linux Advanced Administration
FTAcademy, 2012.
`http://ftacademy.org/materials/fsm/2`

[RoS04] Peter Van Roy & Seif Haridi,
Concepts, Techniques, and Models of Computer Programming
MIT Press, Marzo 2004.

[Shn89] Ben Shneiderman,
User-friendly computer interfaces (Critical technology report)
Chantico Pub. Co, 1989.

[Shn09] Ben Shneiderman,
Designing the User Interface: Strategies for Effective Human-Computer Interaction (5th Edition)
Addison Wesley; 5 edition, Marzo, 2009.

[Spr12] Mark Spruiell,
ZeroC Documentation
ZeroC, 2012.
`http://doc.zeroc.com/display/Doc/Home`
`http://www.zeroc.com/Ice-Manual.pdf`

[Val06] David Vallejo Fernández,
Documentacion de ZeroC ICE
Oreto, Diciembre 2006.

[Ver02] Dinesh C. Verma,
Content distribution networks: an engineering approach
John Wiley and Sons, 2002.

[WaF94] Stanley Wasserman & Katherine Faust,
Social Network Analysis: Methods and Applications (Structural Analysis in the Social Sciences)
Cambridge University Press; 1 edition , November 25, 1994.

[Wil99] Beau Williamson,
Developing IP Multicast Networks, Volume I)
Cisco Press, October 29, 1999.

[WiW11] Daniel Wigdor, Dennis Wixon,
Brave NUI World: Designing Natural User Interfaces for Touch and Gesture
Morgan Kaufmann; primera edición (27 Abril, 2011).

[W3C99] W3C Recommendation,
HTML 4.01 Specification
W3C Working Draft, Septiembre, 1999.

[W3C00] W3C Recommendation,
Document Object Model (DOM) Level 1 Specification (Second Edition)
W3C Working Draft, Septiembre, 2000.

[W3C06] W3C Recommendation,
Cascading Style Sheets (CSS) 2.1 Specification
W3C Working Draft, Noviembre 2006.

[ZaM06] Gonzalo Zarza, Hugo Minni
Generación dinámica de interfaces de usuario a partir de modelos representados

mediante esquemas XML
Grupo de Ingeniería Web, Facultad de Ingeniería y Ciencias Hídricas - Universidad Nacional del Litoral, Noviembre 2006.
`http://www.ing.unp.edu.ar/wicc2007/trabajos/TIAE/143.pdf`

Apéndices

Apéndice A

A APÉNDICE A: INSTALACIÓN Y EJECUCIÓN

La instalación de *FreeStation* esta compuesta por dos instalaciones diferentes. La instalación del entorno servidor con *Backend* PHP, Python y ZeroC ICE.

A.1. INSTALACIÓN DEL ENTORNO SERVIDOR

A.1.1. Instalación de Zero C ICE con repositorio

Esta opción de instalación es sencilla y cómoda, pero no esta aconsejada para desarrolladores que necesiten actualizaciones con bastante frecuencia o disponer de una versión determinada.

Para proceder es necesario descargar el repositorios (en estos momentos para la versión más actual 3.4) desde la página web oficial:

Listado de código fuente A.1: Añadir repositorio ICE

```
1       -O /etc/yum.repos.d/zeroc-ice-rhel6.repo
```

Posteriormente se debe activar el repositorio e instalar el paquete ZeroC Ice y biblioteca Ice para Python:

Listado de código fuente A.2: Instalación Zero C ICE desde repositorios

```
2       ice-python
```

A.1.2. Instalación desde fuentes

Si la necesidad de uso esta más orientada al enfoque desarrollador, se puede compilar la versión que se requiera para desarrollo. En el momento de escribir el proyecto se compiló para la versión 3.4.2 de ICE.

Como prerrequisito para la compilación es necesario instalar la biblioteca mcpp o "portable C preprocessor" desde el repo de ZeroC Ice (también se puede optar por bajar y compilar los fuentes de mcpp, pero no es necesario):

Listado de código fuente A.3: Instalación mcpp

```
1     sudo yum install -y mcpp-devel
```

A continuación, se descargan los ficheros fuente de Ice, se descomprimen y se compilan para la versión en C++ y su binding para Python (en este caso para la versión 3.4.2 de Ice). Para ello puede realizarse todo en una línea con:

Listado de código fuente A.4: Instalación Zero C ICE

```
3     xvzf Ice-*.tar.gz; cd Ice-*/cpp; make; make install; cd ../py/; make;make install
```

La instalación de Ice quedará bajo /opt/Ice-3.4.2/ y el binding python sobre /opt/Ice-3.4.2/python. Es importante recalcar que el binding python se asociara con la versión de defecto de Python en el sistema si se dispone de varias.

Por último, es necesario indicar el el PATH y el PYTHONPATH del sistema, donde se encuentra la instalación de Ice y su binding de Python:

Listado de código fuente A.5: Configuración path

```
4     sudo export PYTHONPATH=/opt/Ice-3.4.2/python:$PYTHONPATH
```

A.2. INSTALACIÓN DEL ENTORNO CLIENTE

La instalación del entorno cliente necesita de la biblioteca python-feedparser si se utiliza el widget *FeedParser*. Para su instalación en entornos basados en Debian:

Listado de código fuente A.6: Instalación python-feedparser

Para el widget VideoArea es necesaria la dependencia Gstreamer:

Listado de código fuente A.7: Instalación dependencias Gstreamer

```
5           gstreamer-tools gstreamer1.0-alsa gstreamer1.0-libav gstreamer1.0-plugins-bad
                gstreamer1.0-plugins-base gstreamer1.0-plugins-good
6       gstreamer1.0-plugins-ugly  gstreamer1.0-pulseaudio gstreamer1.0-tools
7       gstreamer1.0-x libgstreamer-plugins-bad1.0-0 libgstreamer-plugins-base1.0-0
8       libgstreamer1.0-0
```

Para widgets basados en CouchDB es necesario instalar couchdb y sus dependencias desde el PPA de Novacut:

Listado de código fuente A.8: Añadir PPA daily Novacut

```
9       update
```

Posteriormente se instalan los paquetes necesarios:

Listado de código fuente A.9: Instalar paquetes del PPA de Novacut

```
10      novacut dmedia userwebkit
```

Para la ejecución del cliente ICE es necesaria la instalación del paquete zeroc-ice34 para soporte Zero C ICE y el paquete python-zeroc-ice para obtener el soporte del binding de ICE para python.

En sistema basados en Debian, como Ubuntu, el comando de instalación es:

Listado de código fuente A.10: Instalar dependencias ICE

Desafortunadamente, el cliente de *FreeStation* para Ice no puede ejecutarse en Python 3 debido a que en el momento de escribir este proyecto de fin de carrera no existía una versión del binding adaptada[1].

En su lugar el código debe ejecutarse con un intérprete basado en Python 2.7.

A.3. FUNCIONAMIENTO Y EJECUCIÓN

A.3.1. Ejecución de la interfaz cliente

Acceder a la carpeta FreeStation/freestation:

Listado de código fuente A.11: Cambiar directorio

```
```

Posteriormente ejecutar el watchdog.py con un inteprete basado en Python 3:

Listado de código fuente A.12: Ejecutar cliente

```
1 python3 watchdog.py
```

Esto iniciara la interfaz del POI.

A.3.2. Ejecución del cliente basado en ICE

Usando un intérprete basado en Python 2.7, un ejemplo de ejecución cuando el servidor no esta activo:

Listado de código fuente A.13: Ejemplo de conexión rechazada

```
1 FreeStation Client 0.1
2 Ice Version 3.4.2
3 Base: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
4 Error: connection refused. Please, check if the server is running or you are
      connecting properly.
5 Client abort.
```

[1] Zero C ICE soporte Python 3: http://www.zeroc.com/forums/help-center/5138-support-python-3-x-scheduled.html

Nota: La IP ha sido modificada con propósitos de privacidad.

Cuando el servidor esta activo, pero el cliente no esta habilitado, producirá una salida diferente:

Listado de código fuente A.14: Ejemplo de conexión no autorizada

```
Base: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
Proxy: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
Checking if client autorized
Error: client no authorized or disabled
Client abort.
```

Esto provocará un mensaje de warning en el servidor, en la pestaña de logs “Warning” del tipo siguiente:

Listado de código fuente A.15: Mensaje de cliente no autorizado

```
111.111.111.111 not authorized
```

Si la IP del cliente es autorizada, pero no habilitada se mostrará un mensaje similar.

Si por el contrario el cliente cumple con los requisitos de estar autorizado y habilitado, la conexión procederá. Por ejemplo para un cliente configurado para descarga de widgets la salida será:

Listado de código fuente A.16: Conexión con éxito para extracción de widgets

```
Ice Version 3.4.2
Base: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
Proxy: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
Checking if client autorized
Client authorized.
Client ID: 7
Requesting widgets.xml configuration ...
<?xml version="1.0" encoding="UTF-8"?>
<interface>
    <widget>
        <name>TitleDisplay</name>
        <properties>
            <spacing>1</spacing>
            <width>300</width>
            <height>200</height>
            <data>UCLM - FreeStation3</data>
        </properties>
    </widget>
</interface>
```

```
20
21      Configuration saved.
22      Exit
```

Para una configuración de descarga de archivos del servidor, por ejemplo un archivo test.tar.gz el resultado sería:

Listado de código fuente A.17: Conexión con éxito para descarga de archivos

```
23      Ice Version 3.4.2
24      Base: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
25      Proxy: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000:udp -p 0 -z
26      Checking if client autorized
27      Client authorized.
28      Client ID: 7
29      Downloading files for client ...
30      Requesting test.tar.gz file ...
31      File size: 1613757 Bytes
32      Received (0.0 KB) Speed: 0.0 KB/s
33      Received (150.0 KB) Speed: 14.77 KB/s
34      Received (300.0 KB) Speed: 20.84 KB/s
35      Received (450.0 KB) Speed: 24.44 KB/s
36      Received (600.0 KB) Speed: 27.66 KB/s
37      Received (750.0 KB) Speed: 30.03 KB/s
38      Received (900.0 KB) Speed: 29.96 KB/s
39      Received (1.03 MB) Speed: 29.45 KB/s
40      Received (1.17 MB) Speed: 30.54 KB/s
41      Received (1.32 MB) Speed: 31.31 KB/s
42      Received (1.46 MB) Speed: 32.98 KB/s
43      Data saved in 45.4770698547 seconds
44      Exit
```

En el *Frontend*, en la pestaña de logs de la salida estándar, puede apreciarse la información de su inicio:

Listado de código fuente A.18: Ejemplo de pestaña de salida estándar

```
1   None
2   Creating adapter object
3   Adapter Name: ApiAdapter
4   Adapter Comunicator: object #0 (::Ice::Communicator)
5   {
6   }
7   ice thread started: <_DummyThread(Dummy-2, started daemon 1127827776)> thread locals:
        {'my_test_var': '_threading_local.local works'}
8   ice thread started: <_DummyThread(Dummy-3, started daemon 1138317632)> thread locals:
        {'my_test_var': '_threading_local.local works'}
9   Importing servant Api
10  Base servant adapter: Api -t:tcp -h 123.123.123.123 -p 10000 -t 60000 -z:tcp -h
11  123.123.123.124 -p 10000 -t 60000 -z:udp -h 123.123.123.123 -p 39447 -z:udp -h
12  123.123.123.124 -p 39447 -z Enabling adapter object ApiAdapter ready
```

En la pestaña de ejecución “Running”:

Listado de código fuente A.19: Ejemplod de pestaña Running

```
45 PID: 21463
46 Ice Version 3.4.2
47 Ice Version (integer) 30402
48 Module source /opt/Ice-3.4.2/python/Ice.pyc
49 Slice dir: /opt/Ice-3.4.2/slice
50 Loading properties
51 Loading .ice interface from ice/fs.ice
```

Apéndice B

B APÉNDICE B: MANUAL DE USUARIO

En este manual de usuario se describen las diferentes funcionalidades ofrecidas por el sistema, completando la utilización y desarrollo del proyecto de fin de carrera.

B.1. FRONTEND SERVIDOR

En el *Frontend* servidor se realizan las operaciones deseadas con clientes. En las siguientes secciones se explicarán los usos más comunes y necesarios.

B.1.1. Como autorizar y añadir un cliente

Figura B.1: Listado de clientes

Para añadir o autorizar un cliente se debe ingresar al sistema autentificando con los credenciales de usuario y contraseña. Posteriormente acceder a la sección “Clients“ como se muestra en la figura B.1 y pulsar sobre el botón “Add client“.

Después, rellenar con los datos del cliente (IP, hostname) y estado.

Figura B.2: Añadir cliente FreeStation

B.1.2. Como deshabilitar un cliente

La vista de estaciones freestation de la figura B.1, indica un listado de servidores y clientes activos/suspendidos.

Pulsando sobre el icono de circulo verde puede deshabilitarse un cliente o bien habilitarse si el icono del círculo es gris.

También puede editarse el estado pulsando sobre el icono de engranaje.

B.1.3. Como añadir un widget a un cliente

El *Frontend* dispone de una vista de widgets por servidor, donde pueden ser activados/-desactivados además de acceder a sus parametrizaciones correspondientes.

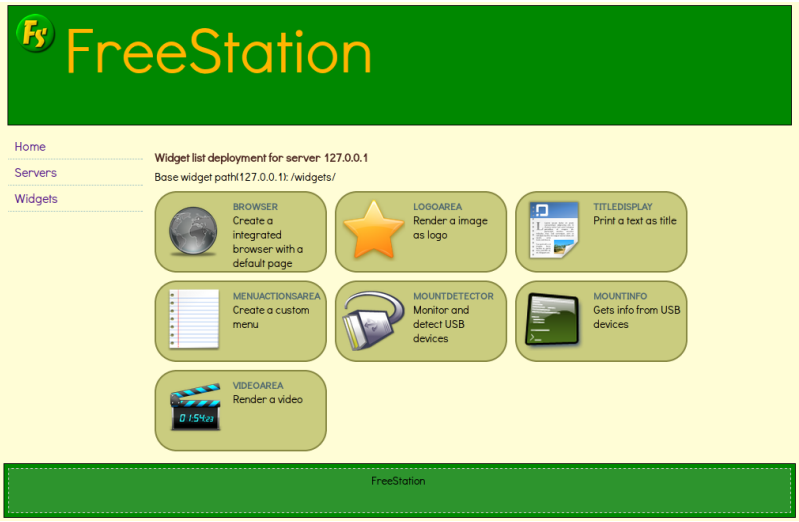


Figura B.3: Vista de Widgets

Pulsando sobre el icono de widgets en la vista cliente como se muestra en la figura B.1 se accede a un listado de los widgets disponibles para dicho cliente.

Posteriormente pueden asociarse nuevos widgets pulsando sobre el botón “Associate” y eligiendo en la lista de disponibles:

Figura B.4: Asociar un widget para un cliente

Dependiendo de la implementación del widget, puede ser editado para ese cliente ofreciendo la configuración personalizada.

Figura B.5: Configurar widget para un cliente

En este caso, el widget Browser muestra la personalización del título y url a cargar:

Figura B.6: Configurar widget Browser

B.1.4. Como añadir nuevos widgets al servidor

En la sección widgets del servidor aparece un listado de los disponibles.

Figura B.7: Lista de widgets para un cliente

Pulsando sobre el botón “Add widget” se procederá a una lectura del directorio de widgets para búsqueda de nuevos widgets. Para ello, el widget debe estar implementado en el *Frontend* para su uso.

B.1.5. Como administrar el servidor desde *Frontend*

Para iniciar o parar o incluso reiniciar el servidor desde *Frontend* se debe hacer click en la sección “Server”. Después el icono de círculo verde o gris mostrará el estado de encendido o apagado respectivamente.

Figura B.8: FreeStation Server Webserver GUI

Según sea el estado se podrá apagar o reiniciar o iniciar. En las pestañas de logs puede visualizarse la información relevante de conexiones y peticiones de los clientes o fallos si se producen.

Apéndice C

C APÉNDICE C: CÓDIGO FUENTE

Debido a la extensión del código fuente de *FreeStation*, éste se incluye únicamente en su versión electrónica en el CD adjunto a este documento.

C.1. INSTALACIÓN DEL ENTORNO SERVIDOR

La estructura de directorios con la que se ha organizado este código fuente es la siguiente:

Para el entorno cliente: **fs**

- freestation: el directorio que contiene la mayor parte del desarrollo del cliente, este contiene directorios útiles como:
  - files: espacio donde se almacenan los archivos descargados para copias de USB
  - ice: almacena los slices para Zero C ICE.
  - img: guarda los recursos de imágenes utilizados en el cliente.
  - ogv: guarda los vídeos utilizados en el cliente en formato ogv
  - templates: plantillas HTML para el widget Browser.
  - test: contiene los test unitarios realizados sobre el cliente.
  - themes: pruebas de themes para GTK.
  - ui: contiene la interfaz para el caso de explotación basado en CouchDB
  - widgets: almacena los widgets basados en GTK disponibles para el cliente
  - xml: guarda las configuraciones de widgets para el cliente.

- test: conteine los test generales realizados sobre pruebas de concepto.

Para el entorno servidor: **freestation-web**

- backend: almacena los ficheros de *Backend*, slices, api y módulo FS de Zero C ICe.
- css: almacena los archivos css del *Frontend*.
- docs: este directoriose ha generado autodocumentación con sphinx como prueba desde el código python. La versión de esta documentación no esta completa pero puede ser orientativa. Puede accederse online desde: `http://freestation.quijost.com/docs/_build/`
- img: contiene las imágenes del *Frontend*.
- inc: incluye los archivos de cabeceras y pie de página para el CMS.
- js: bibliotecas de javascript utilizadas.
- lib: contiene las clases principales del CMS.
- sql: guarda una copia del SQL generado en la primera instalación.
- test: test en PHP.
- tmp: directorio temporal.
- widgets: widgets configurados.

Apéndice D

D GNU FREE DOCUMENTATION LICENSE

Version 1.3, 3 November 2008

<http://fsf.org/>

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

The "**publisher**" means any person or entity that distributes copies of the Document to the public.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties—for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, or distribute it is void, and will automatically terminate your rights under this License.

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, receipt of a copy of some or all of the same material does not give you any rights to use it.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation. If the Document specifies that a proxy can decide which future versions of this License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Document.

11. RELICENSING

"Massive Multiauthor Collaboration Site" (or "MMC Site") means any World Wide Web server that publishes copyrightable works and also provides prominent facilities for anybody to edit those works. A public wiki that anybody can edit is an example of such a server. A "Massive Multiauthor Collaboration" (or "MMC") contained in the site means any set of copyrightable works thus published on the MMC site.

"CC-BY-SA" means the Creative Commons Attribution-Share Alike 3.0 license published by Creative Commons Corporation, a not-for-profit corporation with a principal place of business in San Francisco, California, as well as future copyleft versions of that license published by that same organization.

"Incorporate" means to publish or republish a Document, in whole or in part, as part of another Document.

An MMC is "eligible for relicensing" if it is licensed under this License, and if all works that were first published under this License somewhere other than this MMC, and subsequently incorporated in whole or in part into the MMC, (1) had no cover texts or invariant sections, and (2) were thus incorporated prior to November 1, 2008.

The operator of an MMC Site may republish an MMC contained in the site under CC-BY-SA on the same site at any time before August 1, 2009, provided the MMC is eligible for relicensing.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © 2012 Ángel Guzmán Maeso. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled “GNU Free Documentation License”.

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the “with … Texts.” line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.