Grado en Ingeniería Informática

Escuela Politécnica Superior

Universidad Carlos III de Madrid

Trabajo Fin de Grado

Análisis, diseño e implementación de un sistema de control de encendido, apagado e instalación de un aula departamental

Alumno: *Carlos Medina Sánchez*

Tutor: *Alejandro Calderón Mateos*

Índice de contenidos

Índice de contenidos

Índice de contenidos

Índice de figuras

Índice de figuras

Índice de tablas

Índice de tablas

Índice de tablas

Agradecimientos

Podría pasarme horas citando los nombres de todas las personas a las que daría las gracias por haberme cuidado, apoyado o ayudado durante todos estos años, incluso a aquellas que se marcharon tiempo atrás pero aún siguen guiándome con el fulgor de las ideas que sembraron en mí. De modo que, independientemente de lo mucho que todas ellas signifiquen en mi vida, sólo mencionaré a aquellas que puedan tener la ocasión de leer estas líneas alguna vez. Si os preguntáis por qué no digo vuestro nombre, sabed que no me perdonaría olvidarme de alguien; esa es la razón por la cual prefiero no decir ninguno.

En primer lugar, agradecerle inmensamente a mi familia haberme criado y educado, sobre todo porque si algún día yo también he de formar la mía propia, la utilizaré de buen ejemplo. Si existe alguien o algo más responsable de mi forma de ser, de los valores en los que se basa mi conducta, de las ideas que pueda tener acerca de lo que es correcto o incorrecto; esa es mi familia. No puedo estar más orgulloso de tener un padre, una madre y un hermano como los que tengo.

También he conocido a mucha gente durante todos estos años: creo que a todos los considero mis amigos. Pero si he de darle las gracias con toda mi alma a alguno de ellos, sólo puedo referirme a los que considero mis mejores amigos: aquellos que han compartido conmigo la infancia, la adolescencia y que aún siguen compartiendo momentos conmigo, y a aquellos que a pesar de haberlos conocido más tarde, me demostraron tener un gran corazón y también se han convertido en mis mejores amigos.

Tampoco podría olvidarme del lugar en el que se gestó este proyecto: el Laboratorio del Departamento de Informática. Ese mágico lugar que recordaré siempre y que me apena dejar atrás: por lo que allí me enseñaron y por lo que yo aprendí; pero sobre todo por esas personas que empezaron siendo mis jefes, pasaron por ser mis compañeros, y acabaron siendo también mis amigos.

Por último me gustaría mostrar igualmente mi más profunda gratitud hacia ese *algo* que provocó mi existencia; a esa infinita sucesión de causalidades que, remontándose hasta el origen de los tiempos en una interminable cascada de consecuencias, alcanzan la primera chispa que dio luz a nuestro universo. A esa primera causa, a ese primer motor inmóvil, a ese invisible demiurgo… gracias. Gracias porque después de considerarlo todo desde una perspectiva tan amplia como la que me permite hacerlo mi propio juicio, he de decir que me siento inmensamente afortunado. Creo que he tenido una suerte increíble por haber nacido, por haber tenido la oportunidad de crecer, por haber conocido a personas tan maravillosas como las que he conocido, y por haber podido llegar hasta el lugar en el que hoy me encuentro.

A todas, a todos, y a todo; gracias.

Resumen

La motivación más importante para realizar este trabajo derivó de la necesidad de eliminar muchas de las limitaciones que poseen las utilidades que se utilizan a diario en el Laboratorio del Departamento de Informática. Muchas de estas utilidades son simples scripts cuyo número aumenta a medida que se añaden nuevas funcionalidades y que hace cada vez más difícil su uso.

La herramienta desarrollada en este proyecto unifica y mejora estas utilidades, permitiendo administrar equipos de distintas aulas al mismo tiempo: encenderlos, reiniciarlos en un sistema operativo concreto, apagarlos, o realizar instalaciones de forma remota. También permite llevar a cabo otras tareas que se realizan con frecuencia en el Laboratorio, como la copia de máquinas virtuales o la modificación de permisos de directorios.

Abstract

The most important motivation for this work was the need to eliminate many of the limitations of the tools that are used daily in the Laboratory of Computer Science and Engineering Department. Many of these utilities are just scripts whose number increases as new features are added and that makes their use more difficult.

The tool developed in this project unifies and enhances these utilities, allowing you to manage computers in different classrooms at the same time: turn them on, restart them on a particular operating system, turn them off or perform installations remotely. Also it allows you to perform other tasks that are often performed in the laboratory, such as copying virtual machines or changing directory permissions.

1. Introducción

En este apartado se realiza una aproximación general al proyecto, detallando el contexto en el que surgió la necesidad de desarrollarlo y los objetivos que se pretenden alcanzar tras hacerlo. Por último se describe la estructura escogida para organizar el documento y se definen los acrónimos que se han utilizado.

1.1. Motivación

Durante los dos años en los que he sido becario del **Laboratorio del Departamento de Informática** de la Universidad Carlos III de Madrid, me he adaptado a la dinámica de trabajo de dicho Laboratorio y he aprendido a manejar muchas de las herramientas que se utilizan para mantener a punto los equipos de las aulas. Esta es la razón por la que conozco las limitaciones que poseen algunas de estas utilidades.

La mayoría de las limitaciones se relacionan con la **mala usabilidad** de las herramientas, ya que muchos de estos programas de gestión no son más que un grupo cada vez más diversificado de scripts cuyo uso (recordar el comando concreto, parámetros que acepta, etc.), a medida que su número aumenta, se hace progresivamente más difícil.

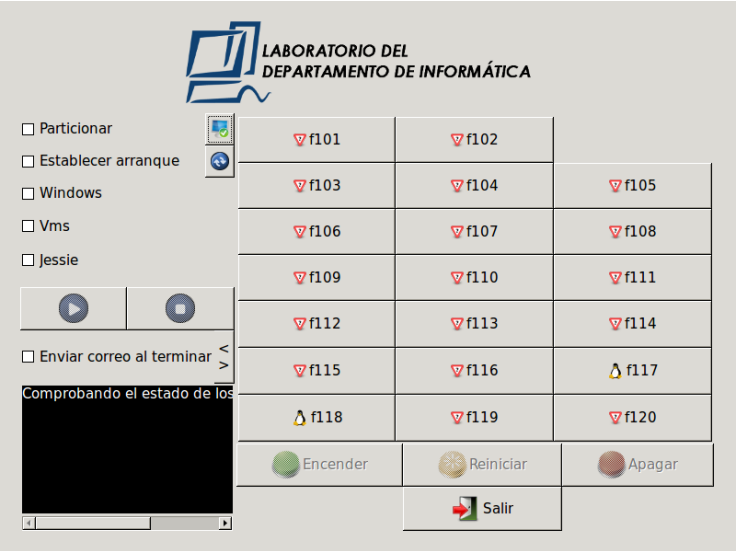


Ilustración 1: Antigua aplicación de instalación de equipos

Hace unos años se desarrolló una herramienta que intentaba agrupar algunas de las tareas de gestión más comunes que se llevan a cabo con los equipos. Esta herramienta permitía el encendido, apagado e instalación de los ordenadores de forma remota mediante una interfaz gráfica. Uno de los propósitos del presente proyecto es sustituir esta vieja herramienta por otra mucho más potente y que complete algunas de las tareas que no se habían incorporado en aquella. Como ejemplo de algunos de los aspectos que se pueden mejorar está la de explotar la capacidad de realizar determinadas tareas de forma concurrente.

Otra de las motivaciones para desarrollar el proyecto fue el hecho de que la herramienta iba a ser utilizada en un **entorno real**, como es el Laboratorio del Departamento de Informática. No todos los alumnos que llevan a cabo un trabajo de fin de grado tienen la oportunidad de evaluar el rendimiento de sus proyectos en los escenarios para los que están diseñados.

1.2. Objetivos

Después de plantear las limitaciones de las herramientas de las que disponemos en el Laboratorio, se enumeraron algunos objetivos que debía cumplir la solución deseada. Estos eran:

- La solución debe ser capaz de realizar las **mismas tareas que la anterior herramienta**, es decir, **encendido**, **reiniciado** y **apagado** de los equipos así como la **instalación remota** de imágenes previamente generadas de dos sistemas operativos distintos (*Debian* y *Windows*).
- La solución debe ser capaz de llevar a cabo **otras tareas** que se realizan con frecuencia en el Laboratorio, como es el **envío de máquinas virtuales** a los equipos y la **modificación de los permisos** (lectura, escritura y ejecución) asociados a los ficheros de estas máquinas virtuales.
- La solución debe ser capaz de aplicar las tareas anteriormente descritas en **diferentes equipos de distintas aulas** de forma simultánea.
- La solución debe implementar una **interfaz gráfica de usuario** que facilite la configuración de las tareas a realizar así como su aplicación en los equipos de las aulas.
- La solución debe poder ejecutarse en los equipos de los técnicos del Laboratorio y en los equipos de las aulas, es decir, deberá ser **multiplataforma**.

1.3. Estructura del documento

A continuación se enumeran los distintos capítulos en los que está dividido el documento, incluyendo un breve resumen de sus respectivos contenidos:

- **Capítulo 1: Introducción**. En este apartado se realiza una aproximación general al proyecto, detallando el contexto en el que surgió, los objetivos que se persiguen al desarrollarlo y la estructura escogida para organizar el documento. También se definen los acrónimos utilizados.
- **Capítulo 2: Estado del arte**. En este apartado se analizarán algunas herramientas que suponen una aproximación al problema que se plantea, además de llevar a cabo una comparativa de las tecnologías existentes en las que apoyarse para construir la solución final.
- **Capítulo 3: Análisis**. En este apartado se describe y modela el problema a resolver, llevando a cabo una definición detallada de los requisitos que debe cumplir el sistema.
- **Capítulo 4: Diseño**. A lo largo de este apartado se muestran todas las especificaciones de construcción y decisiones de diseño relativas al sistema que se desea construir con el fin de resolver el problema modelado en la sección anterior.
- **Capítulo 5: Implementación**. En este apartado se describe el proceso de instalación y configuración del entorno de operación.
- **Capítulo 6: Evaluación**. Este apartado engloba las diversas pruebas que permiten valorar la calidad del sistema y verificar si el sistema cumple las necesidades establecidas por el cliente, así como estimar la satisfacción de los usuarios en relación a la usabilidad de la aplicación.
- **Capítulo 7: Implantación**. En este apartado se detalla la estrategia de implantación del sistema, incluyendo el proceso de despliegue, instalación y configuración de la aplicación desarrollada. También se incluye una referencia al manual de usuario.
- **Capítulo 8: Planificación y presupuesto**. En este apartado se expone un diagrama con la programación estimada de las fases del proyecto y el cálculo detallado de los costes y beneficios obtenidos.
- **Capítulo 9: Conclusiones y trabajos futuros**. En este apartado se exponen algunas consideraciones finales relativas a la elaboración del proyecto y algunas ideas y ampliaciones surgidas durante su desarrollo y que mejorarían las características de la aplicación.
- **Apéndice I: Descripción del prototipo inicial**. Este apéndice incluye una descripción detallada de la maqueta básica que sirvió de base para la programación de la aplicación final.
- **Apéndice II: Manual de administración de equipos con Kosmos**. El manual que se incluye en este apéndice está dirigido a los usuarios finales de la aplicación, con el objetivo de que les sirva de guía de aprendizaje sobre la administración de equipos con la herramienta Kosmos.

- **Apéndice III: Resumen en inglés**. Breve resumen necesario para demostrar las competencias del alumno en inglés.
- **Bibliografía**. En este apartado se enumeran las fuentes y referencias consultadas durante la realización del proyecto.

1.4. Definiciones, acrónimos y abreviaciones

A continuación se definen algunos términos, acrónimos y abreviaciones que aparecen a lo largo de todo el documento. Cuando un concepto resulte demasiado específico, aparecerá en letra cursiva para indicar que se encuentra definido en este apartado:

- **Ad hoc**: es una locución latina que hace referencia a una solución diseñada específicamente para resolver un problema concreto.
- **Administrador**: el rol de administrador se corresponde con el usuario o usuarios que llevarán a cabo un uso más avanzado de la aplicación.
- **Autotools**: es un kit de herramientas de instalación de *GNU* diseñado para generar paquetes portables entre sistemas *UNIX*.
- **Bash**: es una consola de comandos de *UNIX* que ha acabado siendo la terminal por defecto en *Linux* y *OS X*.
- **Bcedit**: es una herramienta para sistemas *Windows* análoga a *efibootmgr* que permite modificar la tabla de particiones en sistemas EFI.
- **Breakpoint**: se trata de un "punto de ruptura" en el código que define una línea dónde debe detenerse el flujo de ejecución de un programa que se está depurando.
- **Checkbox**: una casilla de verificación es una pequeña caja que permite pintar un *tick* en su interior al pinchar con el ratón sobre ella y que se utiliza para seleccionar objetos.
- **Click:** hacer "*click*" con el ratón significa pinchar con alguno de sus botones sobre un lugar de la pantalla.
- **Consola:** una consola o terminal de comandos es un programa que permite interactuar con un sistema operativo sin necesidad de utilizar una interfaz de usuario.
- **Debian:** es un sistema operativo basado en *Linux* que se distribuye con licencias de software libre.
- **Debian Live:** una versión "*Live*" de un sistema operativo permite su uso sin necesidad de instalarlo en el equipo, por lo que no realiza cambios en el disco duro al ejecutarse.
- **EFI**: el sistema *EFI* (o su versión unificada *UEFI*) es una especificación que sustituye a la antigua BIOS.
- **Efibootmgr**: es una herramienta para sistemas *Linux* análoga a *bcedit* que permite modificar la tabla de particiones en sistemas EFI.
- **ESA**: son las siglas correspondientes a la Agencia Espacial Europea.
- **Ext4**: es un sistema de ficheros transaccional que mejora algunas características de ext3.
- **Framework web**: es una estructura tecnológica con diferentes artefactos o módulos concretos destinados al desarrollo de sitios, aplicaciones y servicios web.
- **GNOME**: es un entorno de escritorio para sistemas operativos *GNU/Linux* que se distribuye como software libre.

- **GNU**: es un sistema operativo basado en *UNIX* y formado de manera íntegra por software libre. Fue desarrollado y apoyado por el *Proyecto GNU* y la *Free Software Foundation*.
- **GPL**: se trata de un tipo de licencia de software libre que asegura a los usuarios la libertad de utilizar, estudiar, compartir y modificar el software distribuido bajo esta licencia. Fue creada por *Richard Stallman*, fundador de la *Free Software Foundation*.
- **GRUB**: se trata de un gestor de arranque que permite al usuario escoger el sistema operativo con el que se iniciará el equipo.
- **IDE**: siglas correspondientes a *Integrated Development Environment* (Entorno de Desarrollo Integrado).
- **Iptables**: es un cortafuegos integrado en el *kernel* de *Linux* que permite filtrar los paquetes que salen o entran de la red en función de las condiciones que se especifican en las tablas IP.
- **ISO/IEC**: Organización Internacional de Normalización/Comisión Electrotécnica Internacional.
- **Kernel**: el *kernel* o núcleo de un sistema operativo es una de las partes más importantes, ya que se encarga de gestionar algunos recursos críticos como el acceso seguro al hardware del ordenador a través de llamadas al sistema.
- **Linux**: comúnmente hace referencia a un conjunto de sistemas operativos con un núcleo o *kernel* común basado en *UNIX*. Por ejemplo, *Debian* es una distribución de *Linux*.
- **Multicast**: es un tipo de comunicación (uno-a-muchos o muchos-a-muchos) que permite enviar datagramas IP a distintos receptores en una misma transmisión.
- **Mutex**: es un objeto que permite a varios flujos de ejecución concurrentes acceder al mismo recurso.
- **NFS**: es un sistema de ficheros distribuido que permite trabajar con ficheros a través de la red como si se encontrasen en un dispositivo de almacenamiento local.
- **NMAP**: es una herramienta que analiza los puertos abiertos de un equipo para obtener información sobre los servicios o el sistema operativo que esté ejecutando.
- **NTFS**: es un sistema de ficheros desarrollado por la empresa Microsoft y que se viene instalado por defecto en muchas versiones de *Windows*.
- **OS X**: es un sistema operativo basado en *UNIX* desarrollado por la empresa Apple.
- **Paquete mágico**: se trata de una trama de difusión con una cadena de bytes concreta que permite encender equipos de manera remota. Para ello es necesario configurar el modo *Wake-On-LAN* en los equipos correspondientes para que la tarjeta de red se mantenga a la escucha de paquetes aunque el ordenador se encuentre apagado.
- **PXE:** es un entorno de ejecución que permite arrancar un sistema operativo a través de la red.
- **RAID:** es una tecnología de virtualización que añade redundancia a los datos mediante distintas técnicas como *striping* (división de volúmenes), *mirroring* (espejo) o *parity* (paridad).

- **S.O.**: o su plural SS.OO., hace referencia a las siglas Sistema Operativo.
- **SSH**: es un protocolo criptográfico que se utiliza para acceder de manera segura a máquinas remotas a través de una red.
- **SVG**: es un formato para almacenar imágenes escalables mediante un lenguaje de marcado similar a *XML*.
- **Swap**: el área de intercambio es un espacio del disco duro que se utiliza para almacenar las imágenes de los procesos que han sido expulsados de la memoria principal.
- **Terminal:** una consola o terminal de comandos es un programa que permite interactuar con un sistema operativo sin necesidad de utilizar una interfaz de usuario.
- **TFTP:** es un protocolo de transferencia de ficheros muy básico que permite a un cliente enviar o recibir ficheros desde un equipo remoto.
- **Thread**: un *thread* o hilo es un conjunto de instrucciones que pueden ejecutarse de forma concurrente al flujo principal del programa.
- **Tick**: un *tick* es una pequeña marca en forma de uve que indica la selección de un elemento.
- **UDP**: es un protocolo de red correspondiente a la capa de transporte que permite el envío de datagramas a través de una red sin que se haya establecido una conexión previa entre el emisor y el receptor.
- **Udp-sender/Udp-receiver:** son dos aplicaciones muy simples que permiten enviar y recibir respectivamente paquetes *UDP* a través de una red.
- **UML**: hace referencia a un lenguaje de modelado que permite describir y documentar la estructura de los sistemas.
- **UNIX**: es una familia de sistemas operativos en la cual se han basado otros conocidos sistemas como *Linux* o *OS X*.
- **Wake-On-LAN**: es un estándar que permite configurar las tarjetas de red para encender de manera remota un equipo que se encuentra apagado.
- **Windows:** es una familia de sistemas operativos desarrollados por la empresa Microsoft.
- **Winexe:** es un pequeño software que permite ejecutar comandos en sistemas *Windows* de forma remota.
- **XML:** es un lenguaje de marcas que permite almacenar y tratar datos de manera legible y sencilla.

2. Estado del arte

En este apartado se analizarán algunas herramientas que suponen una aproximación al problema que se plantea, además de llevar a cabo una comparativa de las tecnologías existentes en las que apoyarse para la construcción de la solución final.

2.1. Herramientas de administración remota

Tras llevar a cabo un proceso de investigación se descubrió la existencia en el mercado de varias herramientas de **administración remota de equipos** con diversas características. Uno de los requisitos que debe cumplir la solución a implementar, quizás uno de los más importantes, es que ofrezca la posibilidad de aplicar las tareas de administración tanto a sistemas *Windows* como *Linux*, ya que el objetivo principal del proyecto es gestionar los equipos de las aulas del Departamento de Informática (los cuales permiten trabajar en ambos sistemas operativos).

Otro requisito de gran relevancia, el cual se ha descubierto difícil de cumplir por parte de las utilidades existentes en el mercado, es que la herramienta a implementar no proporcione la opción de conexión por escritorio remoto. Esto se debe al deseo de preservar la privacidad de los usuarios (profesores y alumnos) que hacen uso de los ordenadores de las aulas, además de la falta de necesidad por parte del Laboratorio de una aplicación que ofrezca esta característica.

De entre todas las aplicaciones analizadas, aquí se detallan sólo tres de ellas. Esto se debe a la gran similitud que existe entre las herramientas destinadas a la administración remota de equipos. Para la elección de dichas herramientas se ha optado por escoger aquellas que se aproximan más a la solución que se pretende alcanzar.

A pesar de que muchas de las restricciones impuestas por el cliente son satisfechas por estas herramientas, quedan otras que, por ser demasiado específicas y concretas para el problema que se desea resolver, son incapaces de cumplirlas. Para ello se propone finalmente una solución alternativa que satisfaga todas y cada una de estas funcionalidades.

2.1.1. LogMeIn

LogMeIn[1] es un programa orientado a la administración de equipos y servidores a distancia. Entre sus principales características destacan la capacidad de realizar conexiones de **escritorio remoto**, **apagado** y **encendido remoto** mediante la tecnología *Wake-on-LAN*, o la **transferencia de archivos**.

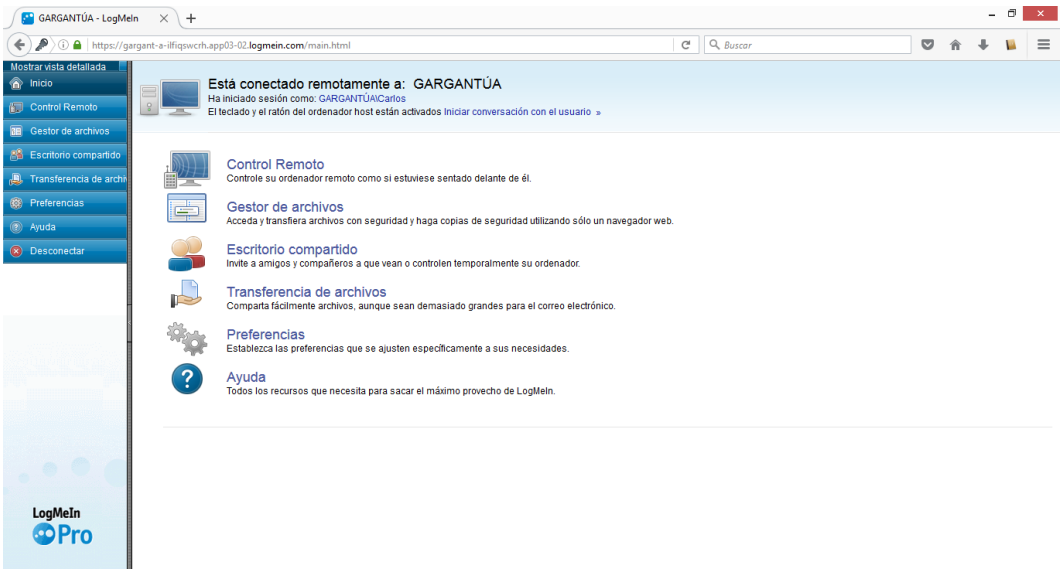


Ilustración 2: Interfaz del programa LogMeIn

También es posible llevar a cabo **reinicios en remoto**, sin embargo esta utilidad no puede realizarlos en un sistema operativo concreto. Esto quiere decir que si el administrador desea que tras un reinicio del sistema éste cargue específicamente *Linux* o *Windows* no podrá hacerlo. Como se verá, esta es una característica que incumplen todas las herramientas analizadas y que supone, por tanto, que la solución final no pueda implementarse con ninguna de ellas.

LogMeIn[2] tampoco ofrece la posibilidad de **instalar imágenes de sistemas operativos** generadas previamente, **modificar permisos** de archivos o directorios de *Windows* (sin recurrir a la conexión por escritorio remoto) o realizar algunas de las **tareas de gestión** más comunes (como son el encendido, reiniciado o apagado) **de forma simultánea** en varios equipos.

Otro de los requisitos que incumple esta aplicación es que a pesar de poder ejecutarse en ordenadores con *Windows* o *OS X*, **no tiene soporte para ordenadores con Linux**, tanto en la ejecución del programa que se encarga de la administración de los equipos como en el control remoto si estos equipos tienen instalado dicho sistema operativo.

2.1.2. TeamViewer

TeamViewer[3] es una aplicación que ofrece acceso remoto para equipos con *Windows*, *OS X* o *Linux*. Además de poseer una interfaz gráfica fácil de usar proporciona las mismas funcionalidades que ofrece LogMeIn, con la ventaja de que TeamViewer sí puede ejecutarse en equipos que tengan *Linux* instalado.

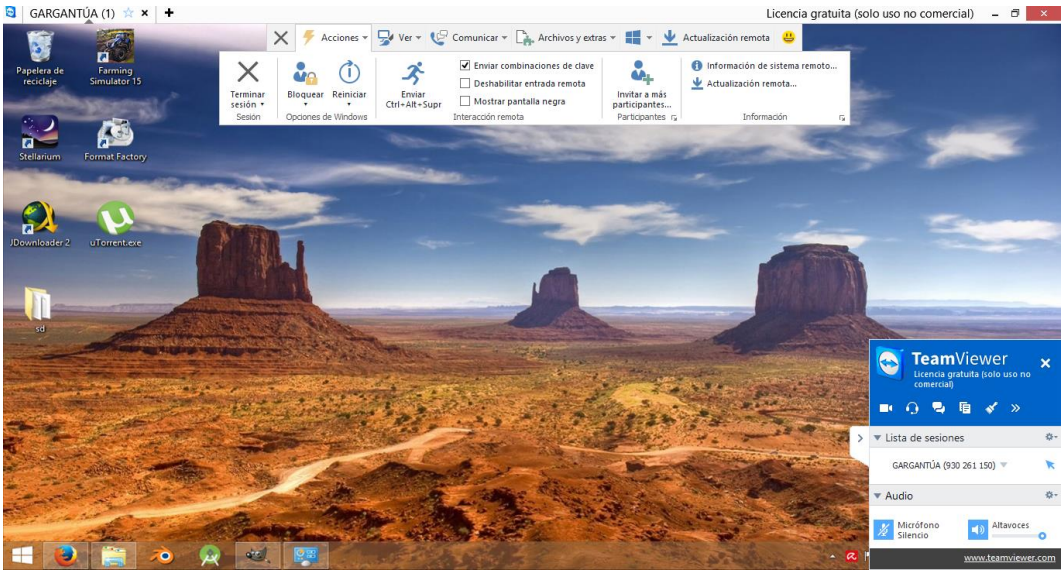


Ilustración 3: Interfaz del programa TeamViewer

Sin embargo, uno de los principales inconvenientes que presenta esta herramienta es que las tareas de administración se llevan a cabo a través de una **interfaz de escritorio remoto** cuyo menú incluye dichas tareas, por lo que es obligatorio acceder al equipo concreto para reiniciarlo o apagarlo.

Otra desventaja es que no se pueden llevar a cabo las **tareas en varios equipos** a la vez, sino que hay que acceder uno por uno a cada equipo. Esto supone una limitación enorme, por ejemplo, en el caso de tener que encender, apagar o reiniciar un aula entera, escenario bastante común en el día a día del Laboratorio.

Además de estas restricciones hay que destacar que, igual que LogMeIn, TeamViewer es incapaz de realizar otras tareas de gestión más específicas como son el reiniciado en un sistema operativo concreto, la instalación de imágenes de sistemas operativos o la modificación de permisos en ficheros y directorios de *Windows*.

2.1.3. Remote Utilities

Remote Utilities[4] es una de las herramientas más potentes que hay en el mercado relacionadas con la administración y el control remoto de equipos. Ofrece prácticamente las mismas funcionalidades que las dos herramientas anteriores, aunque con algunas diferencias.

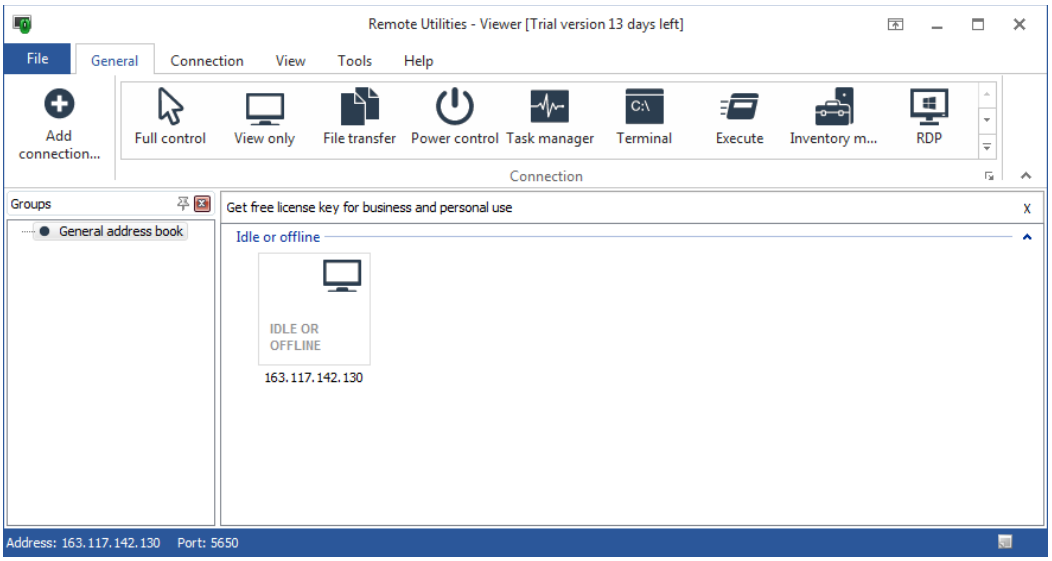


Ilustración 4: Interfaz del programa Remote Utilities

Una de las ventajas que posee es que, al contrario que LogMeIn y TeamViewer, esta utilidad sí permite aplicar las tareas de gestión a varios **equipos de forma simultánea**. Esto significa que permitiría el encendido, apagado o reinicio de un aula completa sin tener que aplicar dichas tareas de equipo en equipo.

Sin embargo, el resto de características que posee no difieren en gran medida de las que ofrecen las otras herramientas analizadas, lo que implica que tampoco es capaz de permitir **reinicios en un sistema operativo concreto**, **instalación de imágenes** o **modificación de permisos**. A esto hay que añadir que no ofrece soporte para equipos que tengan *Linux* instalado.

2.2. Tecnologías para el desarrollo de una herramienta de administración remota

En este apartado se analizarán algunas de las tecnologías existentes con las que se puede construir una aplicación ad hoc de administración remota de equipos. En primer lugar se compararán algunos **lenguajes de programación** existentes, después se estudiarán diversas **bibliotecas gráficas** disponibles y por último se analizarán algunos de los **entornos de desarrollo** que permiten desarrollar una aplicación con las anteriores herramientas.

La razón principal por la que no se ha estudiado la posibilidad de desarrollar una aplicación basada en tecnología web es que esta opción fue descartada a petición expresa del cliente por motivos de seguridad.

2.2.1. Lenguajes de programación

Antes de examinar los lenguajes de programación que se plantearon para el desarrollo del proyecto se definirán algunos conceptos concretos que aparecerán a lo largo de este análisis. Toda la información ha sido obtenida en páginas web relacionadas con los lenguajes de programación analizados[5] [6] [7].

Uno de estos términos es el que se utiliza para designar a un lenguaje como **lenguaje compacto**, que hace referencia a un lenguaje de programación que es capaz de codificar una mayor cantidad de complejidad en un número reducido de caracteres. Esta idea está muy relacionada con el concepto de entropía y con la teoría matemática de la información de Claude E. Shannon[8].

También hay que aclarar el concepto de **lenguaje compilado** frente al de **lenguaje interpretado**. El primero de ellos denomina los lenguajes que transforman el código en lenguaje máquina para que sea entendible por una determinada arquitectura, mientras que el segundo transforma el código en un lenguaje que un intérprete va traduciendo paso a paso en tiempo de ejecución y que es independiente de la plataforma dónde se transformó.

Otro de los términos a definir es el ***tipado* estático** y **dinámico**. Este concepto hace referencia al momento en el que se realiza la comprobación de los tipos de datos con los que opera un lenguaje. En el caso de tener un *tipado* estático, esta comprobación se lleva a cabo durante la compilación. Esto implica que no se puede cambiar el tipo de una variable después de haber sido declarada por primera vez. Por otra parte, con un *tipado* dinámico la comprobación se realiza durante la ejecución del programa.

Las ventajas y desventajas de estos conceptos se detallarán durante el análisis de los tres lenguajes escogidos: **C++**, **Java** y **Python**.

C++

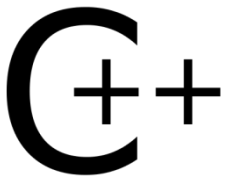


Ilustración 5: Logo de C++

C++[9] es un lenguaje de programación multiparadigma orientado a objetos, ampliamente utilizado para el desarrollo de aplicaciones. En términos de **compactación de código**, C++ es aproximadamente una tercera parte de compacto que Java y una sexta parte que Python, lo que significa, explicado en términos aproximados, que una línea de código en Python equivale a tres líneas en C++.

Una de las ventajas más útiles de este lenguaje es que es un **lenguaje compilado**, lo que hace que posea una velocidad de ejecución mucho más rápida que la de lenguajes interpretados como Python, además de tener una gestión de memoria mucho más eficiente que este tipo de lenguajes. Como se verá más adelante Java posee una condición especial en relación con este aspecto, ya que no puede considerarse ni un lenguaje compilado ni uno interpretado al uso.

Al contrario que Python, C++ y Java comparten la característica de tener un *tipado* estático, lo que permite evitar muchos errores en tiempo de compilación sin tener que esperar a la ejecución del programa. Esto es una ventaja que, como se explicará más adelante, puede paliarse con la facilidad de depuración que poseen los lenguajes interpretados.

Java

Java[10] también es un lenguaje de programación orientado a objetos. Sin embargo, como ya se ha comentado, se encuentra a medio camino entre considerarse un lenguaje compilado y uno interpretado. Esto se debe a que el código Java se traduce a un lenguaje intermedio (*bytecodes*) que es el que posteriormente interpreta en tiempo de ejecución la máquina virtual de Java.

Ilustración 6: Logo de Java

Esta característica brinda una ventaja importante que no poseen los lenguajes compilados, y es que puede ejecutarse el código en cualquier plataforma (se entiende que poseen la máquina virtual de Java instalada). No obstante también supone una desventaja, puesto que al ejecutarse en una máquina virtual los programas poseen una velocidad más lenta de ejecución.

Como se ha comentado antes, el *tipado* de datos que posee Java es estático. Otro incentivo importante para utilizar este lenguaje es el amplio conocimiento en el mismo ya que se ha utilizado en repetidas ocasiones durante la carrera.

Python

Python[11] es un lenguaje de programación *multiparadigma* orientado a objetos. Entre sus características más importantes se encuentra que es un lenguaje interpretado, multiplataforma y de *tipado* dinámico.

Ilustración 7: Logo de Python

Como ya se ha expuesto con anterioridad, Python es un lenguaje muy compacto en comparación con Java o con C++. Esto le permite condensar en pocas líneas de código lo que en otros lenguajes tendría que hacerse en muchas más.

Otra de las ventajas más destacadas es su flexibilidad. Al ser un lenguaje interpretado se pueden llevar a cabo cambios en el código de forma más rápida que en otros lenguajes, ya que a la hora de realizar pruebas no es necesario compilar todo el programa de nuevo (sobre todo si se trata de una aplicación muy grande). Esto lo convierte en un lenguaje muy útil para la realización de prototipos y desarrollos ágiles. Además, el *tipado* dinámico le aporta todavía más flexibilidad a este lenguaje.

Python es un lenguaje con una curva de aprendizaje de inicio rápido, pero que a su vez posee un potencial muy grande (debido a la cantidad de módulos y librerías disponibles). Por otra parte posee una desventaja que es común a todos los lenguajes interpretados, que es la reducida velocidad de ejecución en comparación con los compilados. Sin embargo, en nuestro caso particular, esta diferencia no supone una condición crítica.

Lo que sí puede suponer una condición crítica es la elección entre las dos versiones existentes de Python[12], **Python 2.x** y **Python 3.x**, puesto que existen algunas diferencias importantes entre las que se encuentra, por ejemplo, cambios en la sintaxis (como la llamada a funciones *print* o *xrange*) o que determinadas librerías todavía no ofrezcan soporte para la última de las dos versiones. No obstante, la mayor parte de los módulos de Python actualmente trabajan con ambas versiones y como se comprobará más adelante las bibliotecas utilizadas en el proyecto se incluyen dentro de este grupo de bibliotecas compatibles.

Por último, es importante destacar que Python no es un lenguaje que se enseña durante la carrera. A pesar de esto, tener la oportunidad de aprender un nuevo lenguaje tan versátil como Python también puede suponer una motivación extra en su elección para el desarrollo de Kosmos.

2.2.2. Bibliotecas gráficas

Las bibliotecas gráficas ofrecen un conjunto de objetos y funciones que permiten generar interfaces gráficas de manera más sencilla y rápida para el programador. Estas bibliotecas proporcionan la posibilidad de crear y manejar ventanas, botones, menús, pestañas, etc. Aquí se analizan dos de las que más se ajustan a los requisitos de la aplicación a desarrollar: **Qt** y **Gtk**.

Qt

Ilustración 8: Logo de Qt

Qt[13] es una biblioteca que permite desarrollar interfaces gráficas para aplicaciones que ejecuten en sistemas *Windows*, *Linux*, *OS X* y otros sistemas *UNIX*. Al ser una biblioteca de código abierto se distribuye bajo licencia *LGPL*.

A pesar de que fue escrita en C++, ofrece soporte para multitud de otros lenguajes en forma de *bindings* o adaptaciones de la biblioteca. Entre los lenguajes que ofrecen la posibilidad de trabajar con Qt se encuentran C++, Java, Python, Ruby, Pascal o PHP.

Algunas de las ventajas que posee esta biblioteca es que tiene un entorno de desarrollo visual bastante completo y potente (*Qt Creator*). También posee una amplia comunidad en Internet, que unida la antigüedad de la biblioteca le permite ofrecer una gran cantidad de documentación. Además de ser compatible con las versiones de Python 2.x y 3.x, se distribuye gratuitamente.

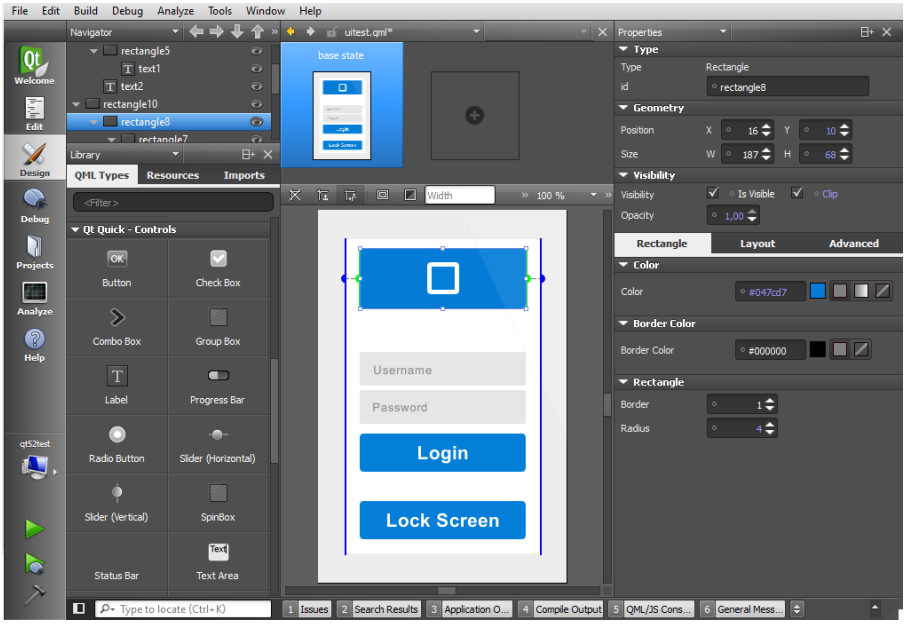


Ilustración 9: Captura extraída de la página oficial de Qt

GTK+

GIMP Toolkit[14] es un conjunto de bibliotecas gráficas para la programación de interfaces gráficas de usuario en sistemas *Windows*, *Linux* y *OS X*. Al igual que Qt, se distribuye bajo licencia *LGPL*.

Ilustración 10: Logo de GTK+

También existen *bindings* de la biblioteca que permiten trabajar en diversos lenguajes de programación, aunque no en todos está disponible en su última versión. Entre ellos se encuentran C, C++, C#, Java, PHP o Python.

De la misma forma que Qt posee una herramienta visual de creación de interfaces gráficas, se puede hacer uso de *Glade* como editor visual que permite generar archivos *XML* con toda la información de interfaces *GTK+/GNOME*. Aunque se puede utilizar como una aplicación independiente existen algunos entornos de desarrollo que la integran en su propia interfaz.

Algunas ventajas de GTK+ sobre Qt es su sencillez, característica que a su vez no le resta versatilidad. Un ejemplo de ello es su utilización en programas como GIMP, Inkscape o Chromium (hasta la versión 34). Esta biblioteca también es compatible con Python 2.x y Python 3.x, además de distribuirse de manera gratuita.

2.2.3. Entornos de desarrollo

Los entornos de desarrollo integrados (comúnmente conocidos como IDE's) son herramientas de programación que integran diversos componentes útiles para la programación de aplicaciones, como editores de texto, compiladores, depuradores y en algunos casos incluso diseñadores de interfaces gráficas.

Todos los IDE's analizados aquí ofrecen la posibilidad de trabajar con Python o están orientados exclusivamente a la programación de aplicaciones con este lenguaje[15].

Wing IDE

Wing IDE[16] es un entorno de desarrollo diseñado específicamente para la programación con Python, que integra entre otras cosas un editor de código con múltiples características y un depurador bastante potente. Ofrece soporte tanto para la versión de Python 2.x como para Python 3.x, además de ser una aplicación multiplataforma (puede ejecutar en sistemas *Windows*, *Linux* y *OS X*).

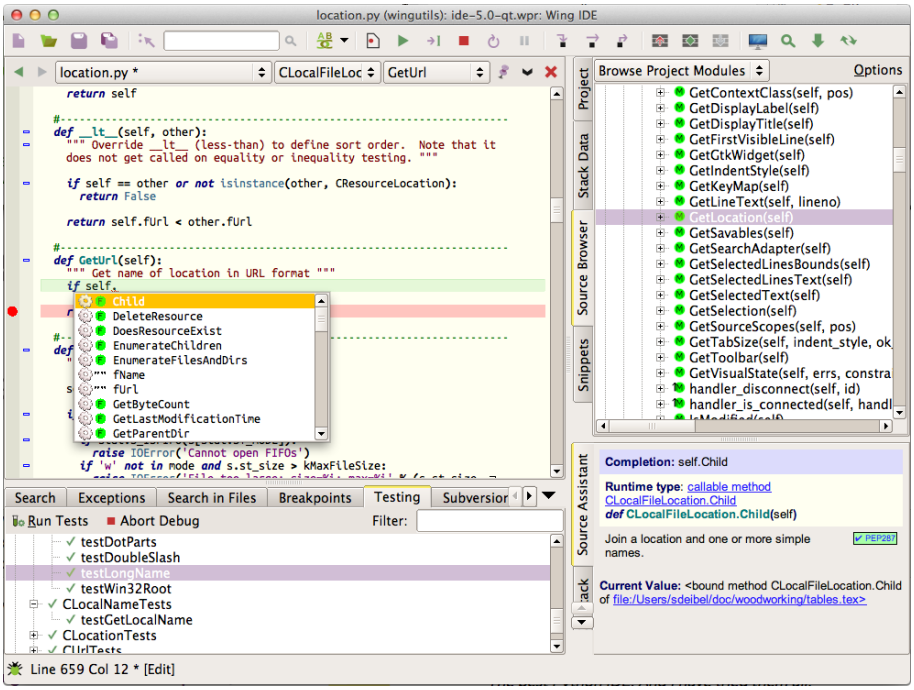


Ilustración 11: Captura extraída de la página oficial de Wing IDE

Wing IDE está disponible en tres versiones: **Wing IDE Professional**, **Wing IDE Personal** y **Wing IDE 101**. Como las características del producto varían entre las tres versiones, aquí se hablará de la versión más completa, que se corresponde con la *Professional*.

El editor de código de Wing IDE presenta la opción de auto-completado, ofrece ayuda en la llamada a funciones obtenida de las fuentes de documentación, resaltado de código así como de errores en la sintaxis, un explorador de archivos integrado que permite navegar entre ficheros y entre módulos y clases, renombrado de símbolos (actualizándose de forma automática en aquellos lugares donde se hace uso de los mismos) o sangrado automático de código.

Por otra parte, el depurador ofrece la posibilidad de establecer *breakpoints*, depuración paso a paso, análisis en tiempo real del valor de variables o depuración remota (lo que permite depurar código que esté ejecutando, por ejemplo, en un servidor web).

Una de las principales desventajas que presenta es que no integra en la propia aplicación ninguna herramienta que permita la creación de interfaces gráficas. Sí incorpora control de versiones, con multitud de programas como Git, Subversion o CVS.

Otra de las desventajas que posee es que el programa posee una licencia propietaria, en contraposición con otros entornos de desarrollo que se distribuyen gratuitamente.

PyCharm

PyCharm[17] es un editor multiplataforma orientado principalmente a la programación de aplicaciones con Python, aunque también proporciona soporte para algunos lenguajes como Javascript, HTML/CSS o Node.js, dirigidos a *frameworks* de desarrollo web con Python. Permite trabajar en sistemas *Windows*, *Linux* y *OS X*, así como con las versiones de Python 2.x y Python 3.x.

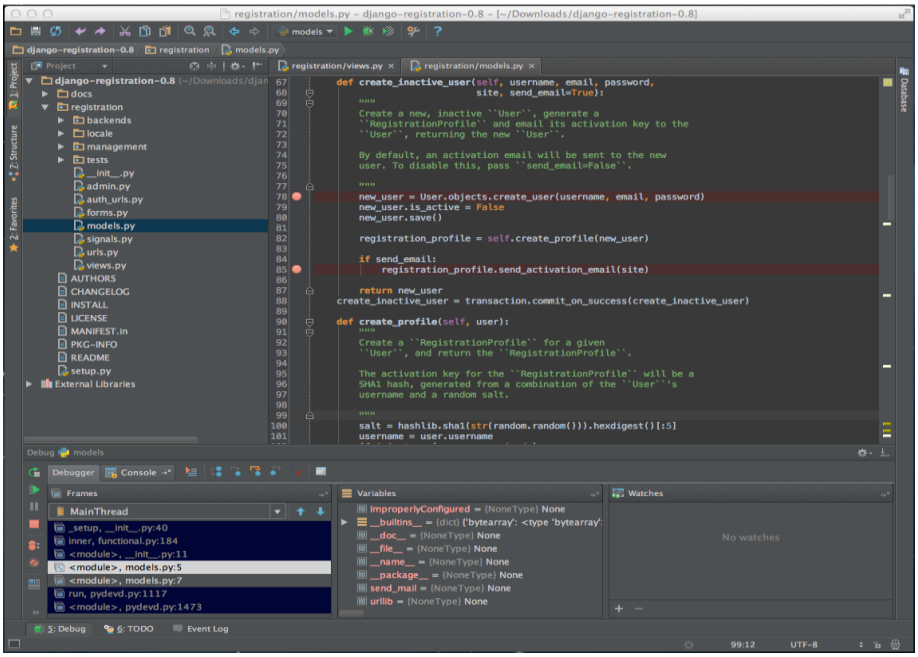


Ilustración 12: Captura extraída de la página oficial de PyCharm

Las características que ofrece el editor de código son muy similares a las de Wing IDE; entre ellas podemos encontrar auto-completado, resaltado de sintaxis/errores, asistencia y análisis de código, exploración del proyecto y navegación entre ficheros, clases y métodos, renombrado automático de símbolos o soporte para entornos de desarrollo web.

El depurador que incorpora PyCharm permite establecer *breakpoints*, la depuración paso a paso del código, examinar el valor de las variables en tiempo real o la depuración remota. PyCharm también integra algunas herramientas muy útiles para el control de versiones, como Git, Mercurial, Subversion o CVS.

Sin embargo, este entorno no incorpora ninguna herramienta de desarrollo de interfaces gráficas integrada, característica deseable para un proyecto como el que se pretende desarrollar. Otra desventaja de esta herramienta es que a pesar de que se distribuye con una licencia Apache en la *Community Edition*, la versión del producto denominada *Professional Edition* que es un poco más completa que la anterior, posee varias opciones de licencia propietaria en función del precio y los términos de uso.

Anjuta

Anjuta[18] es uno de los entornos de desarrollo más completos que existen en el mercado. Entre los lenguajes de programación que soporta se encuentran C, C++, Java, Python (versión 2.x y 3.x), Javascript o Vala. Una de las desventajas que posee es que sólo permite trabajar en sistemas basados en *UNIX*.

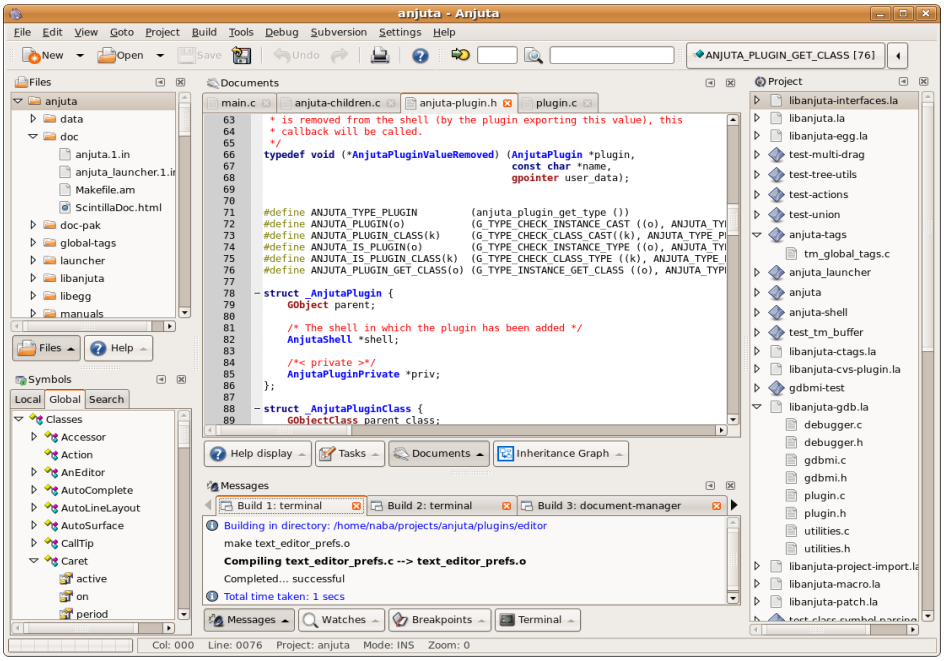


Ilustración 13: Captura extraída de la página oficial de Anjuta

Anjuta integra un editor de código que presenta auto-completado de código, resaltado de sintaxis para todos los lenguajes soportados, sangrado inteligente, bloques de código plegables (se pueden ocultar bucles, funciones, etc.), muestra consejos o ayuda sobre los parámetros de las funciones, posee un explorador que permite navegar entre los archivos del proyecto, búsqueda y reemplazado de símbolos o establecer marcadores para navegar más rápido por el código.

El depurador de código que incorpora Anjuta está basado en el conocido depurador GDB. Este incluye *breakpoints*, análisis de variables en tiempo real, depuración paso a paso y otras opciones comunes a la mayoría de los depuradores que existen. No obstante, una de las características que más versatilidad le proporciona a este entorno es la capacidad de integrar multitud de *plugins* externos, entre los que se encuentra el depurador *Valgrind*, muy útil para detectar fugas y otros problemas relacionados con un mal uso de la memoria.

Pero una de las opciones que más se valoran de este entorno de desarrollo es la integración de un editor de interfaces gráficas de *GTK+/GNOME* en la propia aplicación. Además de esto y como ya se ha comentado, ofrece una gran cantidad de complementos externos, entre los que se encuentra un *plugin* de *Git* para el control de versiones del proyecto.

También ofrece la posibilidad de generar un código inicial ya asociado a la interfaz gráfica, la importación automática de paquetes al configurar por primera vez el proyecto (*Gtk*, *Pixbuf*, etc.) o poder compilar y ejecutar el proyecto completo desde la propia aplicación. Otra de las ventajas que tiene es la distribución bajo una licencia *GPL* y de manera totalmente gratuita.

3. Análisis

Una de las principales razones por las que ninguna de las herramientas descritas en el apartado *2.1 Herramientas de administración remota* resuelven el problema planteado es que éste posee características demasiado específicas para poder satisfacerlas con programas de administración tan genéricos.

Por ello la solución ideal pasaría por el desarrollo de una aplicación *ad hoc*, es decir, especialmente orientada a cubrir cada una de las necesidades específicas que posee el cliente. De aquí en adelante nos referiremos a esta solución con el nombre de Kosmos.

En este apartado se llevará a cabo la definición de los **requisitos del sistema** a implementar. Estos requisitos se han obtenido a través de entrevistas con el cliente, que en este caso era el Laboratorio del Departamento de Informática, en las cuales los técnicos exponían las necesidades que la aplicación debía satisfacer. Tras la revisión de estos requisitos y la elaboración de un primer prototipo se llevaron a cabo algunas entrevistas adicionales con el fin de identificar las necesidades no contempladas previamente y corregir aquellas que presentaban ambigüedades en su definición.

La metodología usada para el análisis del proyecto está inspirada en el estándar para proyectos de software que define la **ESA** y **Métrica v3**, la cual se basa a su vez en los estándares internacionales **ISO/IEC 12207** y **ISO/IEC 15504**.

Aunque la metodología Métrica v3 sitúa la identificación de **requisitos de usuario** en el estudio de viabilidad del sistema, se ha decidido incluir esta sección en la fase de análisis por resultar más adecuada a la estructura del presente documento.

3.1. Características de la solución deseada

Con Kosmos, además de solventar los problemas que se plantean y cubrir las funcionalidades básicas que ya incorporan las herramientas analizadas, se pretende incorporar algunas características adicionales útiles aprovechando la coyuntura que supone el desarrollo de una aplicación específica. A lo largo de esta sección se analizarán algunas de estas propuestas.

Las funcionalidades básicas que debe cubrir Kosmos ya se han planteado anteriormente de forma general. Una de ellas es que ofrezca la posibilidad de aplicar las tareas a los dos sistemas operativos con los que operan los ordenadores de las aulas (*Windows* y *Debian*). Como característica adicional útil, se propuso además que fuese **multiplataforma**, es decir, que pudiese ejecutarse tanto en el servidor que alojará la aplicación como en cualquiera de los equipos que se desean administrar (para así tener la posibilidad de hacer uso de ella conectándose al servidor desde las propias aulas).

Análisis

Como ya se ha comentado, la conexión por escritorio remoto (esto es la visualización del escritorio de los equipos en tiempo real, independientemente del usuario que esté conectado en ese momento) no es una característica deseable para la aplicación a desarrollar. Por esa razón, Kosmos no incluye esta característica entre sus funcionalidades. Sí incorpora, como opción adicional, la capacidad de conectarse de forma remota a la **terminal** de los equipos en *Linux* mediante *SSH*.

Kosmos también permite el **encendido remoto** de equipos (mediante la tecnología *Wake-on-LAN*), el **apagado** o incluso el **reinicio en un sistema operativo** concreto que puede elegirse desde la propia interfaz de la aplicación. Las opciones disponibles son *Windows*, *Debian* o *PXE* (arranque del sistema operativo por red), opción que permite a los técnicos del Laboratorio cargar una *Debian Live* modificada para hacer instalaciones rápidas en los equipos copiando imágenes de los sistemas operativos generadas previamente.

En cuanto a esta última característica, la aplicación debe permitir el **envío de estas imágenes** a los equipos. Como funcionalidad adicional, también debe permitir el particionado de los discos duros así como establecer la configuración de arranque o el copiado de todas las máquinas virtuales de forma conjunta.

Además, Kosmos debe permitir la **transferencia de máquinas virtuales** de forma individual, así como la **modificación de permisos** de sus ficheros y directorios. Esto permite a los técnicos del Laboratorio el copiado de una máquina virtual concreta a mediados de un cuatrimestre, sin tener que enviar también las demás. La modificación de permisos es una tarea muy común cuando se copia una máquina virtual, ya que la mayoría necesitan otorgar permisos de escritura para que los usuarios puedan crear y modificar ficheros cuando las usan.

Por último, también se requiere que las tareas de administración puedan ejecutarse **de forma simultánea en varios equipos**. Esto facilita mucho la aplicación de las tareas a aulas enteras, además de incrementar la velocidad de ejecución del programa si se utilizan hilos o *threads* para llevarlas a cabo de manera concurrente.

En la siguiente tabla se expone una comparación sinóptica de las principales características que poseen las herramientas analizadas junto a la propuesta que se plantea:

Característica \ Herramienta	LogMeIn	TeamViewer v11	Remote Utilities	Kosmos (solución deseada)
La aplicación puede aplicar las tareas en equipos con Windows y Linux	✘	✔	✘	✔
Escritorio remoto	✔	✔	✔	✘
Encendido Wake-on-LAN	✔	✔	✔	✔
Reiniciado en un S.O. concreto	✘	✘	✘	✔
Apagado	✔	✔	✔	✔
Instalación de imágenes de SS.OO.	✘	✘	✘	✔
Transferencia de ficheros (máquinas virtuales)	✔	✔	✔	✔
Modificación de permisos de ficheros y directorios en Windows	✘	✘	✘	✔
Las tareas de gestión se pueden aplicar de forma simultánea en varios equipos	✘	✘	✔	✔

Tabla 1: Comparación de herramientas de administración remota con la solución deseada

3.1. Elección de las tecnologías para el desarrollo de la solución deseada

En el apartado *2.2 Tecnologías para el desarrollo de una herramienta de administración remota* se han estudiado algunas de las tecnologías que permiten el desarrollo de una aplicación de administración remota de equipos. Debemos elegir cuáles de estas herramientas son las más adecuadas para aplicarlas en el sistema, comparando sus ventajas y desventajas en relación a las demás.

Lenguajes de programación

Los parámetros utilizados para comparar de manera objetiva los lenguajes de programación han sido los siguientes:

- **Lenguaje multiplataforma**. Un lenguaje que cumpla esta característica permite desarrollar aplicaciones que puedan ejecutarse en los sistemas operativos de los que se dispone en los ordenadores del Laboratorio (*Debian* y *Windows*).
- **Velocidad de ejecución**. Mide la rapidez con la que se ejecutan las aplicaciones desarrolladas con el lenguaje.
- **Flexibilidad**. Un lenguaje flexible permite realizar cambios y pruebas en el código con mayor rapidez, posibilitando de esta manera desarrollos más ágiles.
- **Compactación**. Este parámetro mide la capacidad de codificar una mayor cantidad de operaciones en un menor número de líneas de código.

Pese a que ninguna de las alternativas cumple con todas las características deseables, en la siguiente tabla se puede observar cuál de las opciones se ajusta más a los requisitos de la aplicación. Cada parámetro se mide en una escala de tres valores (bajo, medio o alto) o con dos respuestas posibles (sí o no):

Característica \ Lenguaje	C++	Java	Python
Lenguaje multiplataforma	Sí	Sí	Sí
Velocidad de ejecución	Alta	Media	Baja
Flexibilidad	Baja	Media	Alta
Compactación	Baja	Media	Alta

Tabla 2: Comparación de lenguajes de programación

La elección final del lenguaje de programación fue **Python**.

Bibliotecas gráficas

Para la elección de la biblioteca gráfica se estudiaron dos alternativas: *Qt* y *GTK+*. Cabe destacar que la selección resultó más difícil, pues las características de ambas bibliotecas son muy similares entre sí.

Las dos bibliotecas son multiplataforma, que era uno de los requisitos esenciales que debía cumplir la aplicación. Las dos bibliotecas son semejantes en cuanto a versatilidad, además de poseer una comunidad muy activa en internet y una amplia documentación. También se distribuyen con licencias *LGPL*.

Sin embargo, pese a la gran similitud entre las dos alternativas, existe una pequeña diferencia que aúpa a *GTK+* por encima de *Qt*, pues se adapta mejor al lenguaje Python. La elección quizás pasó también por una cuestión de gustos o preferencias personales en cuanto a la estética de las interfaces construidas con *GTK+*. Por estas razones la biblioteca gráfica elegida fue **GTK+**.

Entornos de desarrollo

La elección del entorno de desarrollo resultó más sencilla al limitarse en número a aquellos *IDE's* enfocados al desarrollo de aplicaciones con Python. Entre los principales aspectos que se tuvieron en cuenta para la selección de una alternativa se encontraban:

- **Editor de código**. Funcionalidades y versatilidad del editor de código; mayor versatilidad implica reunir más características como auto-completado, resaltado de sintaxis/errores, asistencia y análisis de código, etc.
- **Editor de interfaces integrado**. Una herramienta de creación de interfaces de usuario incluida en el mismo entorno de desarrollo.
- **Control de versiones**. Un sistema de control de versiones como *Git*, *Mercurial*, *Subversion* o *CVS* integrado en la herramienta.
- **Depurador**. Mide la potencia del depurador de código.
- **Licencia**. Software libre o propietario.

Característica \ IDE	Wing IDE	PyCharm	Anjuta
Editor de código	Alta	Alta	Alta
Editor de interfaces integrado	No	No	Sí
Control de versiones	Sí	Sí	Sí
Depurador	Alta	Alta	Alta
Licencia	Propietaria	Apache/Propietaria	GPL

Tabla 3: Comparación de entornos de desarrollo

La elección final del entorno de desarrollo fue **Anjuta**.

3.2. Determinación del entorno operacional

A continuación se definirá el entorno operacional del proyecto. Ante todo se han tenido en cuenta las limitaciones en cuanto a presupuesto y recursos disponibles en el Laboratorio.

Hay que destacar que hasta el momento se disponía únicamente de un equipo para realizar instalaciones en los ordenadores de las aulas. Con el diseño que se plantea en el presente proyecto el número de equipos destinados a esta tarea aumentará a tres y se organizarán de la siguiente forma:

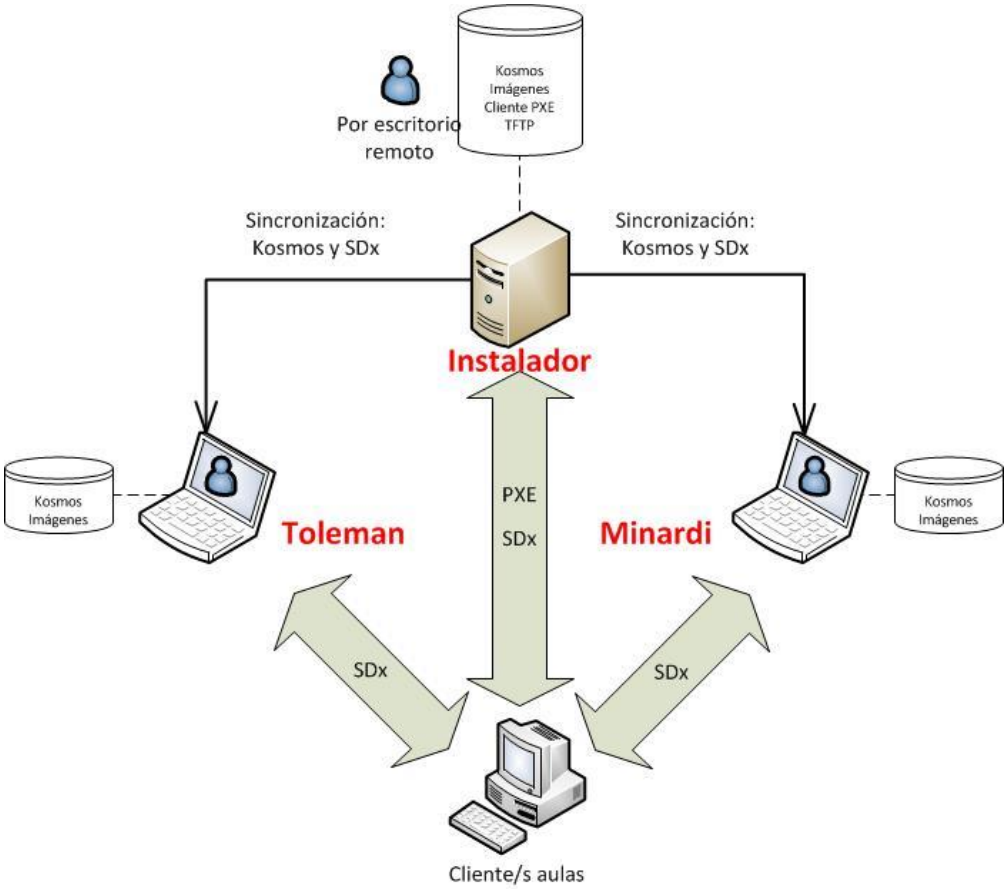


Ilustración 14: Diagrama de la arquitectura de componentes

3.2.1. Servidor Instalador

Instalador es un servidor encargado de almacenar los scripts de gestión que a su vez permite ejecutarlos sobre los ordenadores de las aulas. También contiene la imagen del cliente *PXE* que permite arrancar los equipos por red para llevar a cabo las instalaciones pertinentes. Esta será la única imagen que no se sincronizará con los equipos Minardi y Toleman, ya que el resto de imágenes relativas a los sistemas operativos se almacenarán y sincronizarán en los tres equipos mostrados en el gráfico mediante un script.

Este servidor se encuentra alojado en un armario rack del Centro de Procesamiento de Datos del Laboratorio. En un principio, Instalador se implementó como una máquina virtual que corría bajo un *hipervisor Xen* en el mismo servidor en la que se encuentra alojado. Sin embargo, a pesar de las ventajas que proporciona *virtualizar* una máquina (como la capacidad de guardar fácilmente copias de seguridad y restaurarlas con rapidez), finalmente se decidió no hacerlo de esta manera porque el rendimiento se veía afectado en gran medida.

Las características físicas de la máquina son las siguientes:

CARACTERÍSTICAS DE INSTALADOR	
Procesador	- AMD Opteron™ Processor 850 de 64 bits. - Quad-core a 2400 MHz. - Caché L1 de 128 Kb. - Caché L2 de 1024 Kb.
Memoria RAM	- 16 Gb.
Tarjetas de red	- Broadcom Corporation NetXtreme BCM5703X Gigabit Ethernet (x2).
Discos duros	- 5 discos duros de 136,8 Gb cada uno.

Tabla 4: Características del servidor Instalador

3.2.2. Equipos Toleman y Minardi

Estos dos ordenadores portátiles (idénticos en cuanto al software instalado aunque distintos en relación a sus características físicas) servirán de interfaz gráfica para la instalación de los equipos. Tendrán alojada una copia de la aplicación Kosmos, por lo que podrán transferir las imágenes de los sistemas operativos a los ordenadores de las aulas ya que como se ha comentado contendrán una copia de dichas imágenes sincronizada con el servidor central Instalador.

Las características físicas de estas máquinas son:

CARACTERÍSTICAS DE TOLEMAN	
Procesador	- Intel® Core™ i5-4210U de 64 bits. - Dual-core a 1700 MHz. - Caché L1 de 128 Kb. - Caché L2 de 512 Kb. - Caché L3 de 3072 Kb.
Memoria RAM	- 8 Gb.
Tarjetas de red	- Intel Corporation Ethernet Connection I218-V. - Intel Corporation Wireless 3160.
Discos duros	- 1 disco duro de 248,8 Gb.

Tabla 5: Características del equipo Toleman

CARACTERÍSTICAS DE MINARDI	
Procesador	- AMD Athlon™ X2 Dual Core Processor L310. - Dual-core a 1200 MHz. - Caché L1 de 256 Kb. - Caché L2 de 1024 Kb.
Memoria RAM	- 4 Gb.
Tarjetas de red	- Atheros Communications Inc. AR8131 Gigabit Ethernet. - Atheros Communications Inc. AR928X Wireless Network Adapter.
Discos duros	- 1 disco duro de 242,2 Gb.

Tabla 6: Características del equipo Minardi

3.3. Requisitos de usuario

Con el objetivo de identificar correctamente los **requisitos de usuario** los hemos agrupado en dos tipos básicos:

- **Requisitos de capacidad**: describen las funciones o capacidades que debe ofrecer el sistema.
- **Requisitos de restricción**: describen las condiciones o restricciones impuestas al sistema en la forma en que se alcanzan los objetivos.

Cada requisito de usuario estará definido en una tabla como la que se presenta a continuación:

Identificador			
Título:			
Fuente:	☐ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☐ Baja
Necesidad:	☐ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☐ Baja
Estabilidad:			
Descripción:			

Tabla 7: Plantilla para requisitos de usuario

En la cual cada campo especificará la siguiente información:

- **Identificador**: será un código asociado a cada requisito que lo diferenciará unívocamente. Este identificador tendrá el formato **UX-NN** donde:
  - **U**: indica que el requisito es de usuario.
  - **X**: indica el tipo de requisito, que puede ser:
    - **C**: requisito de capacidad.
    - **R**: requisito de restricción.
  - **NN**: es un número consecutivo del 01 al 99.
- **Título**: será una breve definición general del requisito.
- **Fuente**: indicará el origen del requisito, pudiendo ser el cliente o el desarrollador.
- **Prioridad**: especificará la celeridad con la que deberá implementarse la funcionalidad o restricción definida con respecto a otros requisitos. Puede tomar los valores: **alta, media o baja**.
- **Necesidad**: determinará la importancia que tiene el requisito para el cliente, pudiendo tomar los valores:
  - **Esencial**: el requisito es esencial para un correcto funcionamiento del sistema. Salvo causa justificada no podrán modificarse ni eliminarse requisitos con una definición de necesidad esencial.
  - **Deseable**: el requisito especifica una funcionalidad objetivo pero se incluirá en el sistema siempre que no entre en conflicto con otros requisitos de necesidad esencial.
  - **Opcional**: el requisito no es crítico y podría prescindirse de él en el producto final.
- **Verificabilidad**: indicará el grado en que puede constatarse que el requisito está incluido en el sistema y cumple el objetivo por el que fue definido. Puede tomar los valores: **alta**, **media o baja**.
- **Estabilidad**: determinará la probabilidad que posee un requisito de sufrir cambios a lo largo de todo el desarrollo del proyecto (podría darse el caso si se encontrasen conflictos durante la definición de los requisitos de software o en la fase de diseño).
- **Descripción**: será una explicación más detallada de las características del requisito.

3.3.1. Requisitos de capacidad

A continuación se enumeran los requisitos de capacidad, los cuales describen las funcionalidades esenciales del sistema:

UC-01			
Título:	Obtención de estado de equipos.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán examinar el estado en el que se encuentran los equipos de las aulas de forma remota. A saber: • Apagado. • Encendido en *PXE*. • Encendido en *Debian*. • Encendido en *Windows*.		

Tabla 8: Requisito de usuario UC-01

UC-02			
Título:	Obtención de información adicional de equipos.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán obtener información adicional de los equipos en tiempo real, como el nombre de los usuarios que tienen una sesión iniciada, la fecha y hora en que se encendió un equipo y la versión del *kernel* del sistema operativo que se está ejecutando.		

Tabla 9: Requisito de usuario UC-02

UC-03			
Título:	Selección de equipos.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☑ Media	☐ Baja
Necesidad:	☐ Esencial	☑ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La aplicación permitirá seleccionar los equipos sobre los que se aplicarán las diferentes tareas manualmente o en función de diferentes condiciones.		

Tabla 10: Requisito de usuario UC-03

UC-04			
Título:	Realización de tareas básicas.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán encender, apagar o reiniciar de forma remota los equipos seleccionados en el sistema operativo que escojan (*Debian*, *Windows* o *PXE*).		

Tabla 11: Requisito de usuario UC-04

UC-05			
Título:	Conexión a terminal remota.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☑ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán conectarse a la terminal de los equipos en *Debian* de forma remota.		

Tabla 12: Requisito de usuario UC-05

UC-06			
Título:	Conexión a terminal local.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☐ Deseable	☑ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán lanzar comandos en el servidor dónde se ejecuta la aplicación desde la propia aplicación.		

Tabla 13: Requisito de usuario UC-06

UC-07			
Título:	Transferencia de máquinas virtuales.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán copiar máquinas virtuales específicas a los equipos de las aulas de forma remota.		

Tabla 14: Requisito de usuario UC-07

UC-08			
Título:	Modificación remota de permisos.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán cambiar los permisos de un directorio de *Windows* de forma remota.		

Tabla 15: Requisito de usuario UC-08

UC-09			
Título:	Realización de instalaciones.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los administradores podrán realizar instalaciones en los equipos de las aulas de forma remota. Estas instalaciones pueden ser: • Crear particiones en el disco duro. • Establecer la configuración de arranque de los equipos. • Copiar todas las máquinas virtuales que se utilizan en las aulas. • Copiar la imagen de sistema operativo *Debian*. • Copiar la imagen de sistema operativo *Windows*.		

Tabla 16: Requisito de usuario UC-09

UC-10			
Título:	Concurrencia de tareas.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Las operaciones de encendido, reiniciado, apagado, instalación, copiado de máquinas virtuales y modificación de permisos se podrán llevar a cabo en diferentes equipos de distintas aulas de forma simultánea.		

Tabla 17: Requisito de usuario UC-10

UC-11			
Título:	Mapa de aulas.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☐ Deseable	☑ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La aplicación mostrará un mapa de cada aula que resalte el equipo sobre el que se ha hecho *click*.		

Tabla 18: Requisito de usuario UC-11

3.3.2. Requisitos de restricción

A continuación se enumeran los requisitos de restricción, los cuales describen las condiciones impuestas al sistema en la forma en que se alcanzan los objetivos:

UR-01			
Título:	Aplicación de escritorio multiplataforma.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La aplicación no podrá basarse en tecnologías web; será una aplicación de escritorio multiplataforma.		

Tabla 19: Requisito de usuario UR-01

UR-02			
Título:	Taxonomía de la interfaz.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☑ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La interfaz estará compuesta de dos partes: una mostrará la información de los equipos de las aulas y la otra permitirá seleccionar las tareas a realizar.		

Tabla 20: Requisito de usuario UR-02

UR-03			
Título:	Tabla de información de las aulas.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☑ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La información relativa a los equipos de las aulas se mostrará en forma de lista de aulas desplegables, las cuales serán a su vez listas de los equipos que las componen y que mostrará los datos correspondientes al estado, ocupación y el resto de información adicional en sucesivas columnas.		

Tabla 21: Requisito de usuario UR-03

UR-04			
Título:	Información de equipos apagados.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☐ Deseable	☑ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los equipos que se encuentren apagados ofrecerán la información relativa a su estado, ocupación y el resto de datos en un tono grisáceo.		

Tabla 22: Requisito de usuario UR-04

UR-05			
Título:	Menú desplegable con el botón derecho del ratón.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Las acciones de comprobación, encendido, reiniciado, apagado y apertura de terminal remota se mostrarán en forma de menú haciendo *click* con el botón derecho sobre cualquier lugar de la tabla de información de las aulas.		

Tabla 23: Requisito de usuario UR-05

UR-06			
Título:	Ventana de selección del sistema operativo.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Antes de reiniciar cualquier equipo o aula seleccionada se mostrará un mensaje que permitirá escoger el sistema operativo que cargará tras el reinicio.		

Tabla 24: Requisito de usuario UR-06

UR-07			
Título:	Mensaje de error al realizar una instalación.		
Fuente:	☑ Cliente	☐ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La aplicación mostrará un mensaje de error al intentar lanzar una instalación sobre un equipo que no se encuentre preparado para llevarla a cabo.		

Tabla 25: Requisito de usuario UR-07

3.4. Requisitos de software

A continuación se presentan los requisitos software del sistema a desarrollar. Estos requisitos se derivan de los requisitos de usuario detallados en el apartado anterior. Al igual que los requisitos de usuario, cada requisito de software estará definido en una tabla como la que se presenta a continuación:

Identificador			
Título:			
Fuente:	☐ Cliente	☐ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☐ Baja
Necesidad:	☐ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☐ Baja
Estabilidad:			
Descripción:			
Procedencia:			

Tabla 26: Plantilla para requisitos de software

En la cual cada campo especificará la siguiente información:

- **Identificador**: será un código asociado a cada requisito que lo diferenciará unívocamente. Este identificador tendrá el formato **SX-NN** donde:
  - **S**: indica que el requisito es de software.
  - **X**: indica el tipo de requisito, que puede ser:
    - **F**: requisito funcional.
    - **N**: requisito no funcional.
  - **NN**: es un número consecutivo del 01 al 99.
- **Título**: será una breve definición general del requisito.
- **Fuente**: indicará el origen del requisito, pudiendo ser el cliente o el desarrollador.
- **Prioridad**: especificará la celeridad con la que deberá implementarse la funcionalidad o restricción definida con respecto a otros requisitos. Puede tomar los valores: **alta, media o baja**.
- **Necesidad**: determinará la importancia que tiene el requisito para el cliente, pudiendo tomar los valores:
  - **Esencial**: el requisito es esencial para un correcto funcionamiento del sistema. Salvo causa justificada no podrán modificarse ni eliminarse requisitos con una definición de necesidad esencial.
  - **Deseable**: el requisito especifica una funcionalidad objetivo pero se incluirá en el sistema siempre que no entre en conflicto con otros requisitos de necesidad esencial.
  - **Opcional**: el requisito no es crítico y podría prescindirse de él en el producto final.
- **Verificabilidad**: indicará el grado en que puede constatarse que el requisito está incluido en el sistema y cumple el objetivo por el que fue definido. Puede tomar los valores: **alta, media o baja**.
- **Estabilidad**: determinará la probabilidad que posee un requisito de sufrir cambios a lo largo de todo el desarrollo del proyecto.
- **Descripción**: será una explicación más detallada de las características del requisito.
- **Procedencia**: requisito de usuario a partir del cual se ha derivado el requisito de software.

3.4.1. Requisitos funcionales

A continuación se enumeran los requisitos funcionales, los cuales definen las funciones del sistema de software así como sus componentes:

SF-01			
Título:	Obtención de estado de equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para comprobar el estado de los equipos la aplicación realizará un examen mediante la utilidad *NMAP* con el objetivo de verificar si el puerto 3389 se encuentra abierto (*Windows*) o cerrado (*Linux*). En caso de detectar un sistema *Linux* se accederá a un directorio oculto en el equipo para comprobar si se encuentra en *Debian* o por el contrario ha iniciado en *PXE*. En cualquier otro caso se analizará el puerto 24444 para confirmar que el equipo se encuentra apagado.		
Procedencia:	UC-01		

Tabla 27: Requisito de software SF-01

SF-02			
Título:	Obtención de información adicional de equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para obtener la ocupación, la fecha y hora en que se encendió el equipo o la versión del *kernel* de un equipo, se verificará en primer lugar el sistema operativo en el que se encuentra iniciado. Después se llevará a cabo una conexión mediante *Winexe* (en el caso de *Windows*) o *SSH* (en el caso de *Linux*) y se ejecutará en la terminal del sistema el correspondiente comando que devuelva la información requerida: • Ocupación: comandos *who* (*Linux*) o *quser* (*Windows*). • Fecha/hora de encendido: comandos *uptime* (*Linux*) o *wmic* (*Windows*). • Versión del *kernel*: comandos *uname* (*Linux*) o *ver* (*Windows*).		
Procedencia:	UC-02		

Tabla 28: Requisito de software SF-02

SF-03			
Título:	Selección de equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☐ Alta	☑ Media	☐ Baja
Necesidad:	☐ Esencial	☑ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La selección de los equipos sobre los que se aplicarán las tareas se podrá llevar a cabo de tres formas distintas: • Activando un *tick* situado junto al nombre del equipo. • Activando un *tick* situado junto al nombre del aula al que pertenece el equipo (lo que provoca que se seleccionen todos los equipos de dicha aula). • Pinchando en una de las condiciones de selección que aparecen en el menú que se activa con el botón derecho del ratón (siempre y cuando el equipo cumpla con dicha condición). Estas condiciones son: ○ Ningún equipo. ○ Sólo los equipos que se encuentren en *Windows*. ○ Sólo los equipos que se encuentren en *Debian*. ○ Sólo los equipos que se encuentren en *PXE*. ○ Sólo los equipos que se encuentren encendidos. ○ Sólo los equipos que se encuentren apagados. ○ Todos los equipos.		
Procedencia:	UC-03		

Tabla 29: Requisito de software SF-03

SF-04			
Título:	Encendido de equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para encender un equipo se enviará a través de la red una trama llamada *paquete mágico* mediante la utilidad Etherwake.		
Procedencia:	UC-04		

Tabla 30: Requisito de software SF-04

SF-05			
Título:	Apagado de equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para apagar un equipo se realizará una conexión remota mediante *Winexe* (en el caso de *Windows*) o *SSH* (en el caso de *Linux*) y se ejecutará un script local que apague el equipo.		
Procedencia:	UC-04		

Tabla 31: Requisito de software SF-05

SF-06			
Título:	Reinicio de equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para reiniciar un equipo se realizará una conexión remota mediante *Winexe* (en el caso de *Windows*) o *SSH* (en el caso de *Linux*) y se ejecutará un script local que modifique las entradas de la tabla *EFI* mediante *bcedit* (*Windows*) o *efibootmgr* (*Linux*) con el fin de reiniciar el equipo en el sistema operativo correspondiente.		
Procedencia:	UC-04		

Tabla 32: Requisito de software SF-06

SF-07			
Título:	Conexión a terminal remota.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☑ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La terminal remota con los equipos *Debian* se lanzará como una opción del menú desplegable con el botón derecho del ratón, la cual llevará a cabo una conexión *SSH* con el equipo al que se desea conectarse.		
Procedencia:	UC-05		

Tabla 33: Requisito de software SF-07

SF-08			
Título:	Conexión a terminal local.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☐ Deseable	☑ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para lanzar comandos en el servidor donde se ejecuta la aplicación habrá una terminal integrada en la propia interfaz de ésta.		
Procedencia:	UC-06		

Tabla 34: Requisito de software SF-08

SF-09			
Título:	Transferencia de máquinas virtuales.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para llevar a cabo la transferencia de una máquina virtual concreta se realizará una conexión remota mediante *SSH* para lanzar una instancia de *udp-receiver* en cada uno de los equipos a los que se quieren copiar los datos. Tras esto se ejecutará *udp-sender* en el servidor para enviarlos en *multicast*.		
Procedencia:	UC-07		

Tabla 35: Requisito de software SF-09

SF-10			
Título:	Modificación remota de permisos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para cambiar los permisos de un directorio de *Windows* se realizará una conexión remota mediante *Winexe* y se ejecutará el comando de modificación de listas de control de acceso *icalcs* con las opciones correspondientes.		
Procedencia:	UC-08		

Tabla 36: Requisito de software SF-10

SF-11			
Título:	Instalación *Particionar (crear particiones)*.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para llevar a cabo la instalación relativa a las particiones del disco duro se realizará una conexión remota mediante *SSH* y se ejecutará un script local que cree las particiones correspondientes y formatee el disco duro.		
Procedencia:	UC-09		

Tabla 37: Requisito de software SF-11

SF-12			
Título:	Instalación *Establecer arranque*.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para llevar a cabo la instalación *Establecer arranque* se realizará una conexión remota mediante *SSH* y se ejecutará un script local que configure la tabla de particiones *EFI* o la configuración del *GRUB* en el caso de equipos con sistema de arranque por BIOS.		
Procedencia:	UC-09		

Tabla 38: Requisito de software SF-12

SF-13			
Título:	Instalaciones de transferencia de *Máquinas virtuales*, *Debian* y *Windows*.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para llevar a cabo las instalaciones de transferencia de *Máquinas virtuales*, *Debian* y *Windows* se realizará una conexión remota mediante *SSH* para lanzar una instancia de *udp-receiver* en cada uno de los equipos a los que se quieren copiar los datos. Tras esto se ejecutará *udp-sender* en el servidor para enviarlos en *multicast*.		
Procedencia:	UC-09		

Tabla 39: Requisito de software SF-13

SF-14			
Título:	Concurrencia de tareas.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Las operaciones de encendido, reiniciado, apagado, instalación, copiado de máquinas virtuales y modificación de permisos se ejecutarán de forma concurrente mediante hilos.		
Procedencia:	UC-10		

Tabla 40: Requisito de software SF-14

SF-15			
Título:	Mapa de aulas.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☐ Deseable	☑ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Los mapas de las aulas serán imágenes con formato *SVG* que permitan su modificación automática.		
Procedencia:	UC-11		

Tabla 41: Requisito de software SF-15

3.4.2. Requisitos no funcionales

A continuación se enumeran los requisitos no funcionales, los cuales definen restricciones de los servicios o funciones que proporciona el sistema:

SN-01			
Título:	Lenguaje de programación multiplataforma.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La aplicación se desarrollará en el lenguaje de programación multiplataforma Python.		
Procedencia:	UR-01		

Tabla 42: Requisito de software SN-01

SN-02			
Título:	Interfaz gráfica multiplataforma.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La interfaz de la aplicación se implementará haciendo uso de las bibliotecas gráficas de GTK+.		
Procedencia:	UR-01		

Tabla 43: Requisito de software SN-02

SN-03			
Título:	Taxonomía de la interfaz.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☑ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La interfaz estará dividida en dos zonas cuyo tamaño podrá modificarse arrastrando sus respectivos bordes: • La primera mostrará la información relativa a los equipos de las aulas desde la última actualización de estado y ocupación. • La segunda ofrecerá una serie de utilidades de gestión organizadas en pestañas que englobarán la terminal integrada, el copiado de máquinas virtuales, la asignación de permisos a directorios de *Windows* y la realización de instalaciones.		
Procedencia:	UR-02		

Tabla 44: Requisito de software SN-03

SN-04			
Título:	Implementación de la tabla de información de las aulas.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☑ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La tabla de información de las aulas se implementará mediante un objeto de la biblioteca GTK llamado GtkTreeView, al cual se le asociará un GtkTreeModel que es el que almacenará los datos del modelo.		
Procedencia:	UR-03		

Tabla 45: Requisito de software SN-04

SN-05			
Título:	Columnas de información adicional de los equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☐ Alta	☐ Media	☑ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	La información relativa a los equipos de las aulas aparecerá en columnas en el siguiente orden: • Estado. • Ocupación. • Fecha y hora en que se encendió el equipo. • Versión del *kernel*.		
Procedencia:	UR-03		

Tabla 46: Requisito de software SN-05

SN-06			
Título:	Almacenamiento del código de color de los equipos.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☐ Alta	☐ Media	☑ Baja
Necesidad:	☐ Esencial	☐ Deseable	☑ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	El código del color en el que se muestran los datos de un equipo se almacenará junto con el resto de información en el modelo de datos.		
Procedencia:	UR-04		

Tabla 47: Requisito de software SN-06

SN-07			
Título:	Menú desplegable con el botón derecho del ratón.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	El menú desplegable con el botón derecho del ratón se implementará con GtkMenu y ofrecerá distintas opciones de aplicación de la tarea en función de si se ha hecho *click* sobre un equipo, un aula o sobre cualquier otro lugar de la tabla de información de las aulas.		
Procedencia:	UR-05		

Tabla 48: Requisito de software SN-07

SN-08			
Título:	Ventana de selección del sistema operativo.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Para que el usuario pueda elegir el sistema operativo en el que se reiniciará un equipo se abrirá una ventana adicional con las opciones disponibles, junto a las cuales se situará un botón de radio para seleccionar una de ellas y un icono del sistema operativo.		
Procedencia:	UR-06		

Tabla 49: Requisito de software SN-08

SN-09			
Título:	Mensaje de error al realizar una instalación.		
Fuente:	☐ Cliente	☑ Desarrollador	
Prioridad:	☑ Alta	☐ Media	☐ Baja
Necesidad:	☑ Esencial	☐ Deseable	☐ Opcional
Verificabilidad:	☑ Alta	☐ Media	☐ Baja
Estabilidad:	Durante toda la vida del sistema.		
Descripción:	Si al intentar lanzar una instalación algún equipo no se encuentra preparado, la aplicación detendrá el proceso y abrirá un *MessageDialog* enumerando todos aquellos equipos que no se encuentren en *PXE*.		
Procedencia:	UR-07		

Tabla 50: Requisito de software SN-09

3.5. Revisión y depuración de requisitos

Durante la etapa de análisis del proyecto se llevaron a cabo diversas técnicas que permitieron la correcta identificación de las necesidades que perseguía satisfacer el cliente. Entre algunas de las técnicas que se pusieron en práctica se encontraban las entrevistas periódicas con los técnicos del Laboratorio, en las cuales estos exponían los objetivos que debía cumplir la aplicación final, o la programación de una maqueta muy básica de la aplicación. El desarrollo de este primer prototipo se realizó con el objetivo de facilitar la identificación de nuevos requisitos no contemplados previamente y corregir aquellos que presentaban ambigüedades en su definición.

En el Apéndice I: Descripción del prototipo inicial se lleva a cabo una descripción detallada de la maqueta básica que sirvió de base para la programación de la aplicación final. Como es lógico, gran parte de las funcionalidades no se encontraban aún implementadas, pero resultó muy útil para el cliente ya que a pesar de tener una interfaz muy rudimentaria le permitió hacerse una idea de la apariencia así como de la usabilidad que tendría la aplicación final. De hecho, gracias a la elaboración de este prototipo se pudo realizar una modificación sustancial de la interfaz que supuso una ventaja considerable con respecto a esta primera versión.

Cambios con respecto a la aplicación final

La versión final de la aplicación sufrió varios cambios sustanciales con respecto a la maqueta básica que se desarrolló. Estas modificaciones se describirán a lo largo de esta sección con el fin de dejar constancia de las decisiones que tomó el cliente durante la fase de análisis del proyecto.

El cambio más importante que se llevó a cabo durante este proceso estaba relacionado con los mapas de estado de aulas. El cliente estimó oportuno modificar la interfaz de la aplicación para que mostrase un árbol desplegable de aulas y equipos en lugar de un grupo de pestañas independientes por cada aula. Una de las ventajas más útiles de esta nueva organización es que permite realizar instalaciones en ordenadores de distintas aulas al mismo tiempo. Una desventaja es la saturación provocada por la gran cantidad de información que se muestra a la vez en la pantalla, pues los datos ya no se presentan de forma gráfica.

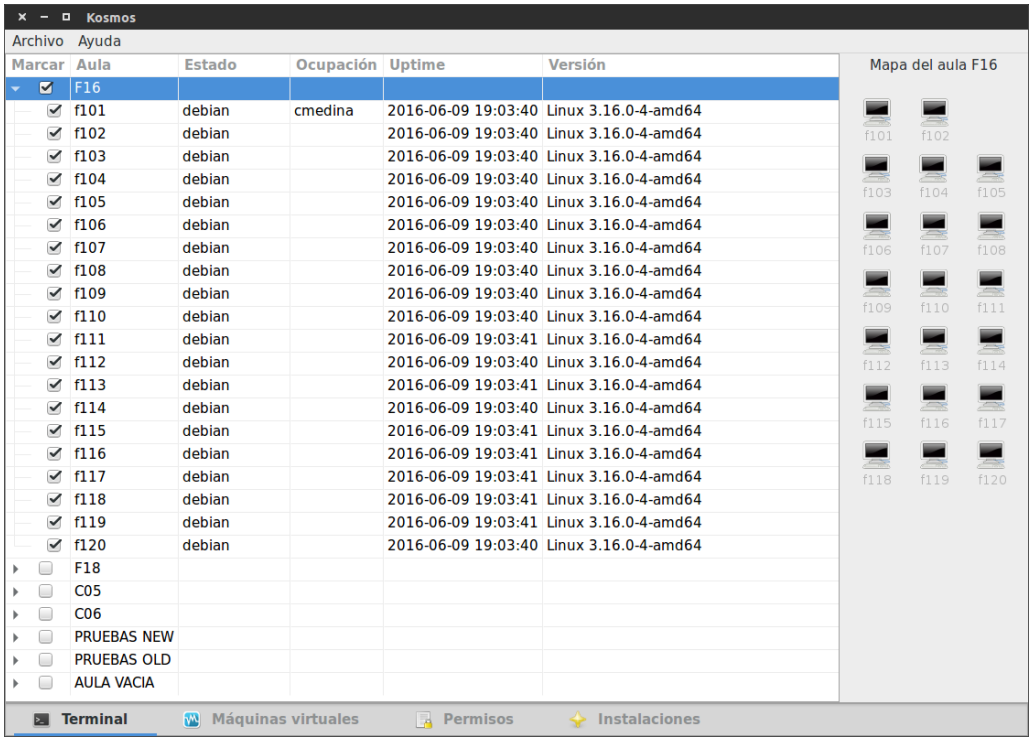


Ilustración 15: Captura del mapa de aulas modificado

A causa de esta modificación también se vio alterada la estructura del archivo *XML* que guarda la información de los equipos de las aulas, puesto que ya no era necesario el almacenamiento de los llamados *huecos* (espacios dónde no había ordenadores pero que eran indispensables para organizar el dibujo del aula en la pantalla).

Otra de las modificaciones que se llevaron a cabo respecto del prototipo inicial es que el cuadro de acciones básicas no se muestra de manera explícita en ningún lugar de la interfaz, sino que se mantiene oculto en un menú desplegable con el botón derecho del ratón. Otra diferencia relativa a este cuadro de acciones básicas es que la tarea que comprueba la ocupación de los equipos era una acción bloqueante en el prototipo, característica que se mejoró en la versión final de la aplicación separando dicha acción en un *thread* independiente que no bloqueaba el flujo de ejecución principal.

Como ya se ha comentado, con el fin de acelerar el desarrollo de esta maqueta se decidió no incluir la funcionalidad de muchas tareas en el código de la aplicación, sino realizar llamadas a los scripts que se utilizan en el Laboratorio para llevarlas a cabo. Posteriormente se integraron en el código de la aplicación. La posibilidad de editar los mapas de aulas se descartó en la versión final de la aplicación a petición del cliente.

Otro de los cambios que sufrió la aplicación fue la separación de la utilidad de máquinas virtuales en las dos funcionalidades que integraba en el prototipo: por un lado, el copiado de máquinas virtuales; por otro, el cambio de permisos de directorios. Esto ofrece la ventaja de poder cambiar los permisos de cualquier máquina virtual o directorio en cualquier momento sin verse obligado a copiar también la máquina virtual asociada.

3.6. Matrices de trazabilidad

Las matrices de trazabilidad son una herramienta muy importante para realizar un correcto seguimiento de los requisitos a lo largo del ciclo de desarrollo de un proyecto. A continuación se expone la matriz de trazabilidad que relaciona los requisitos anteriormente detallados.

3.6.1. Matriz de trazabilidad entre requisitos de usuario y requisitos de software

La siguiente matriz de trazabilidad nos permite verificar que todos los requisitos de usuario quedan cubiertos por un requisito de software:

REQUISITOS DE SOFTWARE	REQUISITOS DE USUARIO																	
	UC-01	UC-02	UC-03	UC-04	UC-05	UC-06	UC-07	UC-08	UC-09	UC-10	UC-11	UR-01	UR-02	UR-03	UR-04	UR-05	UR-06	UR-07
SF-01	X																	
SF-02		X																
SF-03			X															
SF-04				X														
SF-05				X														
SF-06				X														
SF-07					X													
SF-08						X												
SF-09							X											
SF-10								X										
SF-11									X									
SF-12									X									
SF-13									X									
SF-14										X								
SF-15											X							
SN-01												X						
SN-02												X						
SN-03													X					
SN-04														X				
SN-05														X				
SN-06															X			
SN-07																X		
SN-08																	X	
SN-09																		X

Tabla 51: Matriz de trazabilidad entre requisitos de usuario y requisitos de software

4. Diseño

A lo largo de este apartado se muestran todas las especificaciones de construcción y decisiones de diseño relativas al sistema que se desea construir con el fin de resolver el problema modelado en la sección anterior.

4.1. Definición de la arquitectura del sistema

El patrón de diseño de software que mejor se adapta al problema que se quiere resolver es el **Modelo-Vista-Controlador (MVC)**. De esta forma, el sistema se divide en tres subsistemas distintos, los cuales quedan definidos de la siguiente manera:

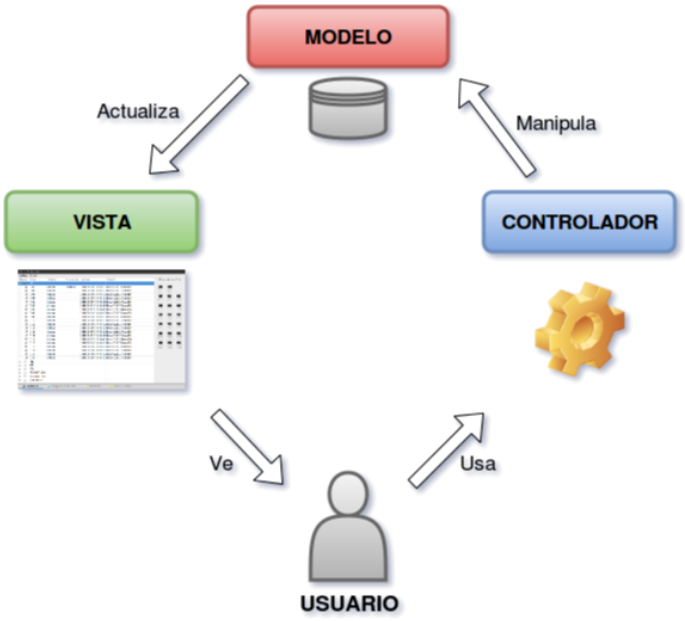


Ilustración 16: Diagrama Modelo-Vista-Controlador

- **Modelo (capa de datos)**: es la parte responsable del almacenamiento, recuperación y procesamiento de los datos con los que trabaja la aplicación. Se comunica con el componente controlador ya que es a través de este componente como recibe las peticiones de acceso a la información, tales como la recuperación del número o el nombre de los equipos que conforman un aula o el estado en el que se encuentran dichos equipos. También se comunica con el componente vista para transmitirle la información una vez ha sido recuperada y procesada.
- **Vista (capa de presentación)**: es la parte encargada de presentar los datos a los usuarios. Cuando recibe nueva información del componente modelo la actualiza en la interfaz gráfica, por ejemplo, mostrando en el campo *ocupación* el nombre del usuario que ha iniciado sesión cuando se comprueba el estado de un equipo.
- **Controlador (capa de negocio)**: es el componente que se ocupa de resolver las peticiones que realizan los usuarios a la aplicación. Al recibir una petición simplemente la transmite al modelo, por ejemplo, cuando un usuario quiere reiniciar remotamente todos los equipos de un aula determinada.

4.2. Diseño de casos de uso

A lo largo de esta sección se detallarán los casos de uso que se derivan de los requisitos de usuario definidos anteriormente, así como las clases y las interfaces de las que se compone la aplicación a desarrollar.

4.2.1. Especificación de los casos de uso

Para la especificación técnica de los casos de uso se ha optado por el uso de una tabla como la que se presenta a continuación:

Identificador	
Título:	
Actores:	
Objetivo:	
Escenario:	
Precondiciones:	
Postcondiciones:	

Tabla 52: Plantilla para casos de uso

En la cual cada campo especificará la siguiente información:

- **Identificador**: será un código asociado a cada caso de uso que lo diferenciará unívocamente. Este identificador tendrá el formato **CU-NN** donde:
  - **NN**: es un número consecutivo del 01 al 99.
- **Título**: se corresponderá con el nombre del caso de uso.
- **Actores**: será el rol que juega un usuario al interactuar con el sistema en el caso de uso correspondiente.
- **Objetivo**: definirá el propósito del caso del uso.
- **Escenario**: enumerará los pasos que realiza el actor al interactuar con la aplicación y la respuesta que el sistema le ofrece.
- **Precondiciones**: definirán las condiciones que deben cumplirse para poder realizar la operación definida por el caso de uso.
- **Postcondiciones**: describen el estado en el que queda el sistema tras realizar una operación.

A continuación se facilita un diagrama *UML* por cada caso de uso que ayuda visualizar los escenarios en los que interactúan los actores con el sistema. En estos diagramas aparecen representados los actores como muñecos y los casos de uso como elipses.

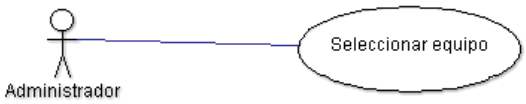


Ilustración 17: Diagrama del caso de uso CU-01

CU-01	
Título:	Seleccionar equipo.
Actores:	Administrador.
Objetivo:	El administrador selecciona los equipos sobre los que quiere aplicar una tarea.
Escenario:	El administrador marca uno o varios equipos de un aula o aulas haciendo *click* sobre el *checkbox* que hay junto al nombre. También puede marcarlos ejecutando la opción *seleccionar* del menú desplegable con el botón derecho del ratón y eligiendo una de las condiciones disponibles.
Precondiciones:	El equipo o equipos no están previamente seleccionados.
Postcondiciones:	El equipo o equipos elegidos se encuentran seleccionados.

Tabla 53: Caso de uso CU-01

Diseño

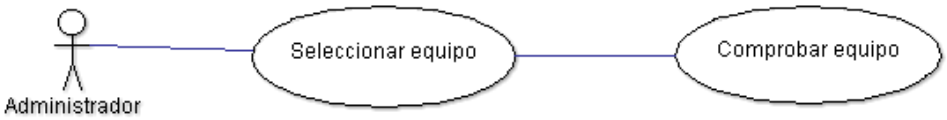


Ilustración 18: Diagrama del caso de uso CU-02

CU-02	
Título:	Comprobar equipo.
Actores:	Administrador.
Objetivo:	El administrador obtiene el estado, la ocupación e información adicional de los equipos de un aula o aulas.
Escenario:	• El administrador selecciona los equipos de los que quiere obtener la información. • El administrador despliega el menú de acciones básicas pinchando con el botón derecho del ratón sobre la tabla de información de las aulas y selecciona la opción *comprobar*. • Se muestra la información en pantalla.
Precondiciones:	Haber seleccionado los equipos de un aula o aulas.
Postcondiciones:	La aplicación muestra el estado, ocupación e información adicional sobre la fecha y hora en que se encendió el equipo y la versión del *kernel* del sistema operativo que se está ejecutando en el equipo o equipos.

Tabla 54: Caso de uso CU-02

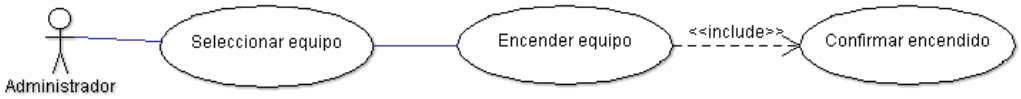


Ilustración 19: Diagrama del caso de uso CU-03

CU-03	
Título:	Encender equipo.
Actores:	Administrador.
Objetivo:	El administrador enciende los equipos de un aula o aulas de forma remota.
Escenario:	• El administrador selecciona los equipos que quiere encender. • El administrador despliega el menú de acciones básicas pinchando con el botón derecho del ratón sobre la tabla de información de las aulas y selecciona la opción *encender*. • El administrador confirma la acción en el cuadro de dialogo que aparece en pantalla.
Precondiciones:	Haber seleccionado los equipos de un aula o aulas.
Postcondiciones:	El equipo o equipos elegidos se encuentran encendidos.

Tabla 55: Caso de uso CU-03

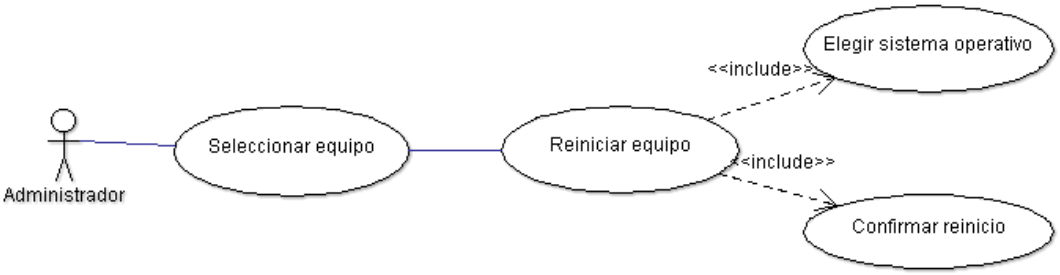


Ilustración 20: Diagrama del caso de uso CU-04

CU-04	
Título:	Reiniciar equipo.
Actores:	Administrador.
Objetivo:	El administrador reinicia los equipos de un aula o aulas de forma remota.
Escenario:	• El administrador selecciona los equipos que quiere reiniciar. • El administrador despliega el menú de acciones básicas pinchando con el botón derecho del ratón sobre la tabla de información de las aulas y selecciona la opción *reiniciar*. • El administrador selecciona el sistema operativo en el que quiere reiniciar los equipos y confirma la acción en el cuadro de dialogo que aparece en pantalla.
Precondiciones:	Haber seleccionado los equipos de un aula o aulas.
Postcondiciones:	El equipo o equipos elegidos se encuentran en el sistema operativo correspondiente después de haber reiniciado.

Tabla 56: Caso de uso CU-04

Ilustración 21: Diagrama del caso de uso CU-05

CU-05	
Título:	Apagar equipo.
Actores:	Administrador.
Objetivo:	El administrador apaga los equipos de un aula o aulas de forma remota.
Escenario:	• El administrador selecciona los equipos que quiere apagar. • El administrador despliega el menú de acciones básicas pinchando con el botón derecho del ratón sobre la tabla de información de las aulas y selecciona la opción *apagar*. • El administrador confirma la acción en el cuadro de dialogo que aparece en pantalla.
Precondiciones:	Haber seleccionado los equipos de un aula o aulas. Que los equipos seleccionados no se encuentren apagados.
Postcondiciones:	El equipo o equipos elegidos se encuentran apagados.

Tabla 57: Caso de uso CU-05

Diseño

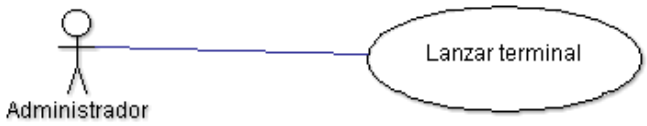


Ilustración 22: Diagrama del caso de uso CU-06

CU-06	
Título:	Lanzar terminal.
Actores:	Administrador.
Objetivo:	El administrador se conecta a la terminal de los equipos *Debian* de un aula o aulas de forma remota.
Escenario:	• El administrador hace *click* con el botón derecho sobre el equipo con el que desea conectarse. • El administrador selecciona en el menú desplegable la opción *lanzar terminal*.
Precondiciones:	Que los equipos seleccionados se encuentren encendidos en *Debian*.
Postcondiciones:	Se abre una terminal de conexión con el equipo.

Tabla 58: Caso de uso CU-06

Ilustración 23: Diagrama del caso de uso CU-07

CU-07	
Título:	Copiar máquina virtual.
Actores:	Administrador.
Objetivo:	El administrador transfiere una máquina virtual concreta a los equipos de un aula o aulas de forma remota.
Escenario:	• El administrador selecciona los equipos a los que quiere transferir la máquina virtual. • El administrador selecciona la máquina virtual que quiere copiar. • El administrador pulsa el botón correspondiente para iniciar la transferencia. • El administrador confirma la acción en el cuadro de diálogo que aparece en pantalla.
Precondiciones:	Los equipos seleccionados se encuentran encendidos en *PXE*.
Postcondiciones:	La transferencia se ha llevado a cabo correctamente en los equipos seleccionados.

Tabla 59: Caso de uso CU-07

Ilustración 24: Diagrama del caso de uso CU-08

CU-08	
Título:	Cambiar permisos.
Actores:	Administrador.
Objetivo:	El administrador cambia los permisos de un directorio en los equipos *Windows* de un aula o aulas de forma remota.
Escenario:	• El administrador selecciona los equipos en los que quiere cambiar los permisos. • El administrador escribe la ruta del directorio al que quiere modificar los permisos. • El administrador elige los grupos y usuarios a los que quiere modificar los permisos. • El administrador establece los permisos que quiere modificar. • El administrador pulsa el botón correspondiente para iniciar la modificación de permisos. • El administrador confirma la acción en el cuadro de diálogo que aparece en pantalla.
Precondiciones:	• Los equipos seleccionados se encuentran encendidos en *Windows*. • El directorio al que se le quieren cambiar los permisos existe.
Postcondiciones:	Los permisos del directorio correspondiente se encuentran modificados correctamente.

Tabla 60: Caso de uso CU-08

Diseño

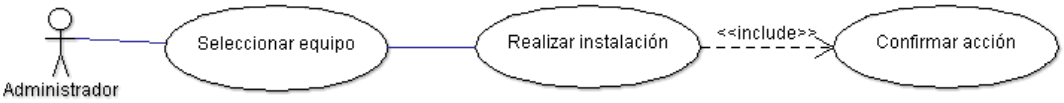


Ilustración 25: Diagrama del caso de uso CU-09

CU-09	
Título:	Realizar instalación.
Actores:	Administrador.
Objetivo:	El administrador realiza una instalación en los equipos de un aula o aulas de forma remota.
Escenario:	• El administrador selecciona los equipos sobre los que quiere llevar a cabo la instalación. • El administrador elige la instalación que quiere realizar. • El administrador pulsa el botón correspondiente para iniciar la instalación. • El administrador confirma la acción en el cuadro de diálogo que aparece en pantalla.
Precondiciones:	Los equipos seleccionados se encuentran encendidos en *PXE*.
Postcondiciones:	La instalación se ha llevado a cabo correctamente en los equipos seleccionados.

Tabla 61: Caso de uso CU-09

4.3. Diseño de clases

Tras haber llevado a cabo la identificación de los casos de uso se procede al diseño de las clases que se derivan de estos. Para ofrecer una visión más general de las clases y sus relaciones se presenta el siguiente esquema *UML*:

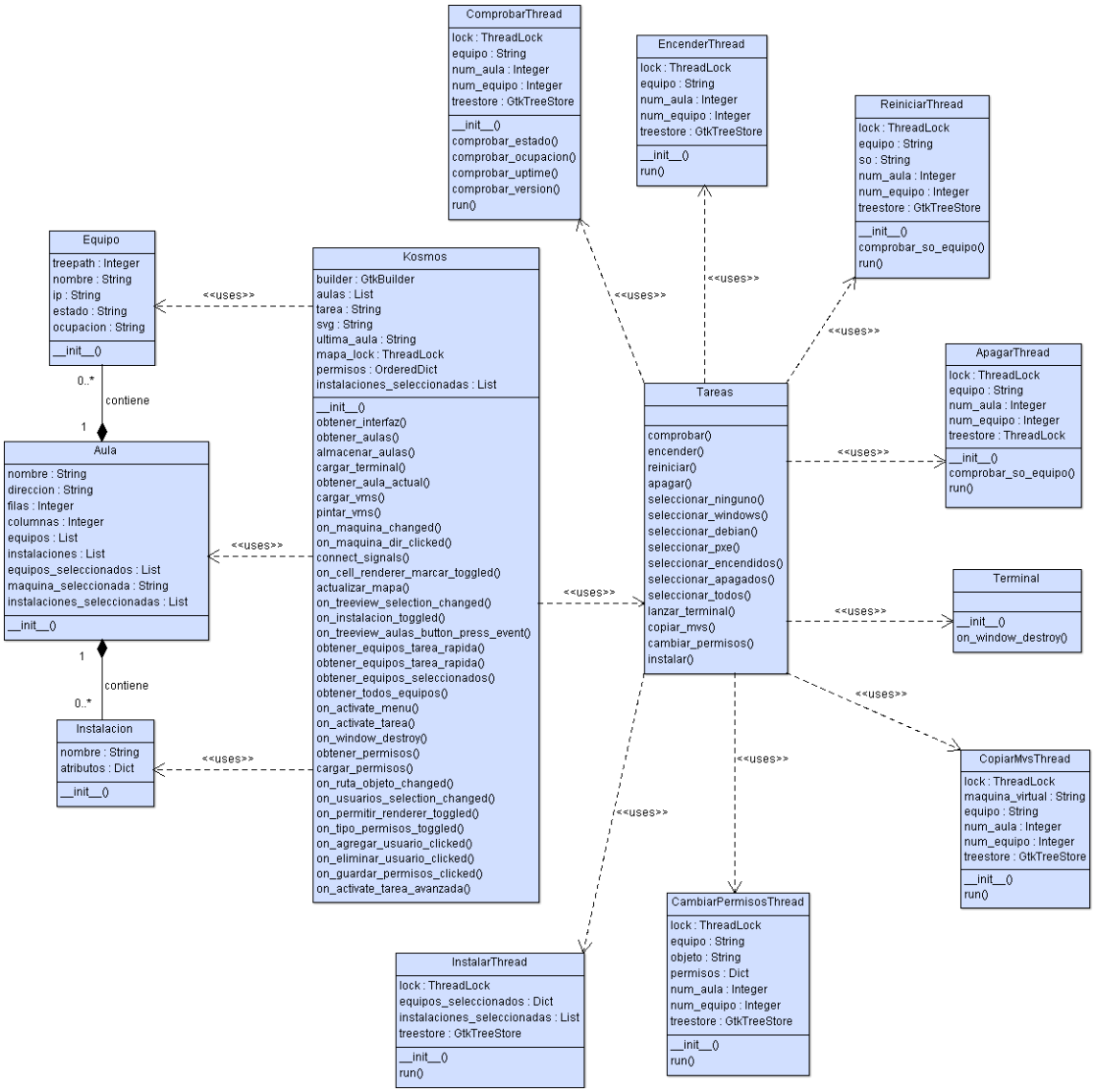


Ilustración 26: Diagrama de clases

A continuación se detallan los métodos que componen cada una de estas clases. En primer lugar se expone la clase *Kosmos*, que se corresponde con la clase principal del programa. Esta clase se encarga de obtener toda la información de las aulas desde los ficheros de configuración y cargarla en la interfaz al inicializar la aplicación. También contiene los métodos manejadores, los cuales se ocupan de realizar las operaciones correspondientes al activarse las señales de los objetos de la interfaz.

Clase Kosmos	
__init__(self)	
Objetivo:	Configura la ventana principal de la aplicación y la carga en pantalla.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Llama a las funciones que se ocupan de obtener la interfaz de la aplicación y de cargar y almacenar la información de las aulas, además de crear una ventana y dibujar los objetos de la interfaz en ella.
Retorno:	-
obtener_interfaz(self)	
Objetivo:	Obtiene los objetos que conforman la interfaz de la aplicación.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Carga los objetos de la interfaz desde un archivo externo y conecta las señales de dichos objetos con sus manejadores correspondientes.
Retorno:	-
obtener_aulas(self)	
Objetivo:	Obtiene la información relativa a los equipos que componen las aulas.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Carga los datos de las aulas desde un archivo externo recorriendo la estructura del fichero para obtener los atributos de cada aula, los equipos que las componen y las instalaciones que pueden llevarse a cabo en cada una de ellas. Por último agrega las aulas a una lista accesible desde cualquier otro método de la clase.
Retorno:	-
almacenar_aulas(self)	
Objetivo:	Guarda la información de las aulas en el modelo.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Obtiene el modelo donde se guardará la información de las aulas e itera la lista de aulas para almacenar la información en dicho modelo.
Retorno:	-
cargar_terminal(self)	
Objetivo:	Genera una terminal del sistema en el grupo de utilidades de gestión avanzada.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Obtiene el objeto de la interfaz donde se insertará la terminal, genera una terminal con la configuración correspondiente y la agrega a dicho objeto.
Retorno:	-

obtener_aula_actual(self)	
Objetivo:	Obtiene el aula sobre la que se ha hecho *click* o el aula a la que pertenece el equipo sobre el que se ha hecho *click*.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Obtiene el objeto sobre el que se ha hecho *click*, analiza si se trata de un aula o un equipo y devuelve el aula seleccionada en el primer caso y el aula a la que pertenece el equipo en el segundo.
Retorno:	Objeto de la clase Aula.
cargar_vms(self)	
Objetivo:	Configura la herramienta de copiado de máquinas virtuales del grupo de utilidades de gestión avanzada.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Obtiene los objetos de la interfaz correspondientes a la herramienta de copiado de máquinas virtuales así como el directorio por defecto en el que se encuentran almacenadas y que viene especificado en un archivo de configuración externo. En caso de no estar especificado se escoge aquel en el que se esté ejecutando la aplicación y se llama al método *pintar_vms()* pasándole como argumentos todos estos parámetros.
Retorno:	-
pintar_vms(self, kosmos_window, desplegable, directorio, copiar_mvs, img_copiar_mvs)	
Objetivo:	Busca las máquinas virtuales existentes en el directorio indicado por parámetro y las muestra en el desplegable correspondiente de la herramienta de copiado de máquinas virtuales.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkWindow* correspondiente a la ventana de la aplicación, objeto *GtkComboBox* correspondiente al cuadro desplegable de máquinas virtuales, objeto *GtkButton* correspondiente al botón de selección del directorio, objeto *GtkButton* correspondiente al botón que permite iniciar el copiado y objeto *GtkImage* correspondiente a la imagen de dicho botón.
Funcionamiento:	Configura el desplegable y comprueba si existe el directorio indicado por parámetro. En caso afirmativo incluye el directorio en un archivo de configuración externo y analiza dicho directorio agregando una entrada en el desplegable por cada máquina virtual que encuentre. Si no encuentra máquinas virtuales deshabilita el botón que permite iniciar el copiado.
Retorno:	-
obtener_permisos(self, usuarios_liststore, dicionario_permisos)	
Objetivo:	Obtiene los usuarios y sus permisos del archivo de configuración correspondiente.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkListStore* que almacena los usuarios que se muestran por pantalla y diccionario que almacena la información de todos los usuarios y sus respectivos permisos.
Funcionamiento:	Recorre el archivo de configuración que contiene los usuarios y permisos por defecto y los almacena en un diccionario.
Retorno:	-

cargar_permisos(self)	
Objetivo:	Obtiene los usuarios y sus permisos y prepara la pestaña correspondiente del grupo de utilidades avanzadas.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Llama a la función *obtener_permisos()* y conecta las señales correspondientes a la selección de usuarios y permisos de la interfaz.
Retorno:	-
on_ruta_objeto_changed(self, ruta_objeto, instalar, img_instalar)	
Objetivo:	Manejador que activa o desactiva el botón de ejecutar el cambio de permisos cuando se inserta una ruta para el objeto al que se cambiarán los permisos.
Argumentos:	Referencia al objeto instanciado (self), cadena de caracteres correspondiente a la ruta del objeto al que se le cambiarán los permisos, objeto *GtkButton* correspondiente al botón que permite ejecutar el cambio de permisos y *GtkImage* correspondiente a la imagen insertada en el botón.
Funcionamiento:	Si la longitud de la ruta del objeto es mayor que cero habilita el botón correspondiente al cambio de permisos. En caso contrario se deshabilita.
Retorno:	-
on_usuarios_selection_changed(self, usuarios_selection, permisos, permisos_liststore, dic_permisos)	
Objetivo:	Manejador que obtiene el nombre del usuario seleccionado y muestra sus permisos en la pestaña correspondiente del grupo de utilidades avanzadas cuando se selecciona un usuario distinto.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkTreeSelection* que almacena la información del usuario seleccionado, objeto *GtkTreeView* que muestra los permisos en la interfaz, objeto *GtkListStore* que almacena los permisos que se muestran por pantalla y diccionario que almacena la información de todos los usuarios y sus respectivos permisos.
Funcionamiento:	Obtiene el nombre del usuario seleccionado, cambia el título de la columna para indicar a quién pertenecen los permisos, borra todas las filas de permisos y carga los nuevos valores de éstos.
Retorno:	-
on_permitir_renderer_toggled(self, permitir_renderer, path, model, dic_permisos, usuarios_selection)	
Objetivo:	Manejador que almacena el nuevo valor de un permiso al modificarlo en el widget de permisos.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkCellRendererToggle* correspondiente a los *checkboxes* asociados a cada permiso, objeto *GtkTreePath* correspondiente a la posición que ocupa el objeto, *GtkListStore* que almacena los permisos que se muestran por pantalla, diccionario que almacena la información de todos los usuarios y sus respectivos permisos y objeto *GtkTreeSelection* que almacena la información del usuario que se encuentra seleccionado.
Funcionamiento:	Obtiene el permiso que se quiere cambiar y el usuario al que pertenece y los almacena en el diccionario de permisos.
Retorno:	-

on_tipo_permisos_toggled(self, tipo_permisos, label, usuarios_liststore, usuarios_selection, dic_permisos, usuarios_box, permisos_box)	
Objetivo:	Manejador que prepara los widgets de la pestaña de cambio de permisos cuando el usuario selecciona el tipo de permisos que desea modificar de un objeto.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkRadioButton* correspondiente al botón de radio activado, cadena de caracteres correspondiente al nombre de la elección, objeto *GtkListStore* que almacena los usuarios que se muestran por pantalla, objeto *GtkTreeSelection* que almacena la información del usuario que se encuentra seleccionado, diccionario que almacena la información de todos los usuarios y sus respectivos permisos y objetos *GtkBox* correspondientes a las cajas donde se insertan los usuarios y los permisos editables.
Funcionamiento:	Evalúa la elección que ha realizado el usuario y en función de ella habilita la edición de usuarios y permisos o por el contrario la deshabilita para poder aplicar los permisos por defecto (permisos generales).
Retorno:	-
on_agregar_usuario_clicked(self, agregar_usuario_button, kosmos_window, dic_permisos, usuarios_liststore, usuarios_selection)	
Objetivo:	Manejador que permite agregar un grupo o usuario en el widget correspondiente de la utilidad de cambio de permisos.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkButton* correspondiente al botón que permite agregar un nuevo usuario, objeto *GtkWindow* correspondiente a la ventana de la aplicación, diccionario que almacena la información de todos los usuarios y sus respectivos permisos, objeto *GtkListStore* que almacena los usuarios que se muestran por pantalla y objeto *GtkTreeSelection* que almacena la información del usuario que se encuentra seleccionado.
Funcionamiento:	Crea una ventana para introducir el nuevo usuario y comprueba la entrada para mostrar un mensaje de error en caso de que el usuario ya exista. En caso contrario agrega el usuario introducido junto con sus permisos al diccionario de permisos.
Retorno:	-

on_eliminar_usuario_clicked(self, eliminar_usuario_button, kosmos_window, dic_permisos, usuarios_liststore, usuarios_selection, permisos, permisos_liststore)	
Objetivo:	Manejador que permite eliminar un grupo o usuario en el widget correspondiente de la utilidad de cambio de permisos.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkButton* correspondiente al botón que permite eliminar un usuario, objeto *GtkWindow* correspondiente a la ventana de la aplicación, diccionario que almacena la información de todos los usuarios y sus respectivos permisos, objeto *GtkListStore* que almacena los usuarios que se muestran por pantalla, objeto *GtkTreeSelection* que almacena la información del usuario que se encuentra seleccionado, objeto *GtkTreeView* que muestra los permisos en la interfaz y objeto *GtkListStore* que almacena los permisos que se muestran por pantalla.
Funcionamiento:	Obtiene el usuario seleccionado y la posición que ocupa. Una vez hecho esto elimina el usuario de la pestaña de cambio de permisos y del diccionario de permisos. Utiliza la posición del usuario eliminado para seleccionar el siguiente usuario de la lista y mostrar sus permisos por pantalla.
Retorno:	-
on_guardar_permisos_clicked(self, guardar_permisos_button, dic_permisos, permisos_generales_button)	
Objetivo:	Manejador que permite almacenar la configuración actual de usuarios y permisos como permisos por defecto.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkButton* correspondiente al botón que permite guardar la configuración actual como permisos por defecto, diccionario que almacena la información de todos los usuarios y sus respectivos permisos y objeto *GtkRadioButton* correspondiente al botón de radio que permite seleccionar los permisos generales.
Funcionamiento:	En primer lugar borra los permisos que había almacenados en el archivo de configuración. Después, por cada usuario que haya en el diccionario de permisos, inserta una entrada en el archivo con el usuario y sus permisos correspondientes. Por último activa el botón de radio correspondiente a la opción de permisos generales.
Retorno:	-
on_maquina_changed(self, widget)	
Objetivo:	Manejador que obtiene el nombre de la máquina virtual cuando el usuario selecciona una máquina distinta en el desplegable.
Argumentos:	Referencia al objeto instanciado (self) y objeto *GtkComboBox* correspondiente al desplegable que ha generado la señal.
Funcionamiento:	Obtiene el directorio de máquinas virtuales de un archivo de configuración externo y almacena la ruta de la máquina virtual seleccionada.
Retorno:	-

on_maquina_dir_clicked(self, widget)	
Objetivo:	Manejador que permite seleccionar un directorio distinto del cual obtener las máquinas virtuales.
Argumentos:	Referencia al objeto instanciado (self) y objeto *GtkButton* correspondiente al botón que permite cambiar el directorio de máquinas virtuales.
Funcionamiento:	Genera una ventana que permite seleccionar un directorio de máquinas virtuales y llama al método *pintar_vms()* para que las muestre en el desplegable correspondiente.
Retorno:	-
connect_signals(self)	
Objetivo:	Conecta las señales asociadas a los objetos de la interfaz con sus correspondientes manejadores.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Obtiene todos los objetos de la interfaz que generan señales y asocia dichas señales con los métodos que permiten manejarlas, especificando como argumentos los parámetros correspondientes.
Retorno:	-
on_cell_renderer_marcar_toggled(self, treestore_aulas, path)	
Objetivo:	Manejador que actualiza en el modelo los equipos y aulas seleccionados cuando el usuario marca el *checkbox* correspondiente.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *GtkTreePath* correspondiente a la posición que ocupa el objeto seleccionado.
Funcionamiento:	Evalúa si el objeto seleccionado es un equipo o un aula y cambia en el modelo el valor de la marca. En el caso de ser un aula cambia el valor de todos los equipos de dicha aula.
Retorno:	-
actualizar_mapa(self, treeview_selection)	
Objetivo:	Refresca el mapa del aula resaltando el nombre del equipo sobre el que se ha hecho *click*.
Argumentos:	Referencia al objeto instanciado (self) y objeto *GtkTreeSelection* que almacena la información del equipo sobre el que se ha hecho *click*.
Funcionamiento:	Obtiene el equipo sobre el que se ha hecho *click* y adquiere el *mutex* asociado al mapa de aulas. Tras esto abre el fichero *SVG* que contiene la información de la imagen, vuelca el contenido en una cadena de caracteres para trabajar con ella y hace visible la etiqueta roja del equipo seleccionado. Por último oculta las etiquetas rojas del resto de equipos, vuelve a convertir la cadena de caracteres en una imagen *SVG* y la carga en pantalla liberando el *mutex* del mapa.
Retorno:	-

on_treeview_selection_changed(self, treeview_selection)	
Objetivo:	Manejador que obtiene el nombre del equipo o aula sobre el que se ha hecho *click* y que realiza los cambios pertinentes sobre la interfaz.
Argumentos:	Referencia al objeto instanciado (self) y objeto *GtkTreeSelection* que almacena la información del equipo seleccionado.
Funcionamiento:	En primer lugar lanza un *thread* para ejecutar el método *actualizar_mapa()* y evitar que no bloquee el flujo principal del programa. Después obtiene el objeto sobre el que se ha hecho *click* y en función del aula a la que pertenece (en el caso de ser un equipo) se ofrecen las instalaciones disponibles para dicha aula y se establece la máquina virtual seleccionada con anterioridad en el desplegable.
Retorno:	-
on_instalacion_toggled(self, checkbutton, instalaciones_disponibles_widget, aula, instalar, img_instalar)	
Objetivo:	Manejador que obtiene el nombre de las instalaciones seleccionadas.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkCheckButton* correspondiente a la instalación seleccionada, objeto *GtkGrid* correspondiente al objeto de la interfaz donde se muestran las instalaciones disponibles, objeto *Aula* correspondiente al último aula seleccionada por el usuario, objeto *GtkButton* correspondiente al botón que permite iniciar la instalación y objeto *GtkImage* correspondiente a la imagen de dicho botón.
Funcionamiento:	Cuando el usuario marca o desmarca una instalación recorre el objeto de la interfaz que contiene las instalaciones disponibles y las añade a una lista de instalaciones seleccionadas. En caso de no haber ninguna instalación seleccionada deshabilita el botón que permite iniciar las instalaciones.
Retorno:	-
on_treeview_aulas_button_press_event(self, treeview_aulas, event)	
Objetivo:	Manejador para el *click* derecho del ratón sobre la tabla de información de las aulas.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkTreeView* correspondiente a la tabla de información de las aulas y objeto *GdkEvent* que contiene la información sobre el evento que activó la señal.
Funcionamiento:	Evalúa el botón que ha sido pulsado y si se corresponde con el botón derecho del ratón obtiene el objeto seleccionado y configura el menú desplegable en función de dicho objeto. Esto consiste en cambiar la etiqueta de algunas opciones concretas del menú para indicar el aula o el equipo sobre el que se realizarán. En el caso de haber hecho *click* con cualquier otro botón del ratón se desmarcan todos los elementos seleccionados.
Retorno:	-

obtener_equipos_tarea_rapida(self)	
Objetivo:	Obtiene el nombre del equipo o aula sobre el que se desea ejecutar una tarea de forma rápida (sin necesidad de seleccionarlo previamente).
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Obtiene el objeto sobre el que se ha hecho *click* y si se trata de un aula la itera añadiendo todos los equipos de dicha aula a una lista de equipos. Si se tratase de un equipo se añade únicamente el equipo sobre el que se ha hecho *click*. Por último se devuelve la lista de equipos sobre los que se debe ejecutar la tarea rápida.
Retorno:	Diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen.
obtener_equipos_seleccionados(self)	
Objetivo:	Obtiene una lista de los equipos que tengan marcados sus *checkboxes* correspondientes.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Itera todas las aulas comprobando equipo por equipo si el *checkbox* asociado a los mismos se encuentra marcado o no. En caso afirmativo los añade a una lista de equipos que se devuelve al final del método.
Retorno:	Diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen.
obtener_todos_equipos(self)	
Objetivo:	Obtiene una lista de todos los equipos de las aulas.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Itera todas las aulas añadiendo cada uno de los equipos a una lista que se devuelve al final del método.
Retorno:	Diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen.
on_activate_menu(self, menu_item)	
Objetivo:	Manejador que obtiene el nombre de las opciones del menú desplegable que contienen opciones secundarias.
Argumentos:	Referencia al objeto instanciado (self) y objeto *GtkMenuItem* correspondiente a la opción del menú que emitió la señal.
Funcionamiento:	Cuando el usuario pasa el ratón por encima de una opción del menú que contiene opciones secundarias se almacena el nombre de la primera opción en una variable global.
Retorno:	-

on_activate_tarea(self, menu_item, tarea_rapida)	
Objetivo:	Manejador que realiza llamadas a las correspondientes funciones dependiendo de la tarea activada desde el menú desplegable.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkMenuItem* correspondiente a la opción del menú que emitió la señal y *Boolean* que indica si la tarea debe aplicarse al objeto sobre el que se ha hecho *click* o sobre todos los objetos seleccionados en la tabla de información de las aulas.
Funcionamiento:	Obtiene el nombre de las opciones del menú y del submenú seleccionadas y comprueba si la tarea debe aplicarse al objeto sobre el que se ha hecho *click* o sobre todos los objetos seleccionados en la tabla de información de las aulas. En función de estos datos llama a las funciones correspondientes de la clase Tareas.
Retorno:	-
on_activate_tarea_avanzada(self, widget, tarea_avanzada)	
Objetivo:	Manejador que realiza llamadas a las correspondientes funciones dependiendo de la tarea avanzada activada.
Argumentos:	Referencia al objeto instanciado (self), objeto *GtkButton* correspondiente al botón que activa la tarea avanzada y cadena de caracteres correspondiente al nombre de la tarea.
Funcionamiento:	Obtiene los equipos sobre los que se ejecutará la tarea avanzada. Tras esto evalúa el nombre de la tarea seleccionada y obtiene los objetos que se pasarán como parámetro a la función correspondiente de la clase Tareas.
Retorno:	-
on_window_destroy(self, kosmos)	
Objetivo:	Destruye la ventana cuando el usuario cierra la aplicación.
Argumentos:	Referencia al objeto instanciado (self) y objeto *GtkWindow* que emitió la señal.
Funcionamiento:	Llama a la función correspondiente de Gtk que se encarga de cerrar la ventana de forma controlada.
Retorno:	-

Tabla 62: Clase Kosmos

La clase *Tareas* contiene los métodos que lanzan un *thread* por cada equipo al aplicar las distintas tareas. También contiene métodos para seleccionar los equipos en función de ciertas condiciones o para abrir una terminal remota en un equipo.

Clase Tareas	
comprobar(equipos_seleccionados, treestore_aulas)	
Objetivo:	Lanza *threads* que comprueban el estado de los equipos seleccionados.
Argumentos:	Diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Adquiere el *mutex* asociado a la tabla de información de las aulas, crea un *thread* por cada equipo seleccionado y los lanza.
Retorno:	-
encender(on_off_dialog, on_off_pregunta, on_off_img_prev, on_off_img_next, on_off_equipos, equipos_seleccionados, treestore_aulas)	
Objetivo:	Lanza *threads* que encienden los equipos seleccionados.
Argumentos:	Objetos *GtkDialog*, *GtkLabel* y *GtkImage* relacionados con el cuadro de confirmación del encendido de equipos, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Configura el cuadro de confirmación del encendido de equipos y lo muestra por pantalla. Tras obtener la confirmación del usuario adquiere el *mutex* asociado a la tabla de información de las aulas, crea un *thread* por cada equipo seleccionado y los lanza.
Retorno:	-
reiniciar(reiniciar_dialog, sistemas_reiniciar, equipos_seleccionados, treestore_aulas)	
Objetivo:	Lanza *threads* que reinician los equipos seleccionados en un sistema operativo determinado.
Argumentos:	Objetos *GtkDialog* y *GtkBox* relacionados con el cuadro de selección del sistema operativo en el que se reiniciarán los equipos, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Configura el cuadro de selección del sistema operativo en el que se reiniciarán los equipos y lo muestra por pantalla. Tras obtener la confirmación del usuario adquiere el *mutex* asociado a la tabla de información de las aulas, crea un *thread* por cada equipo seleccionado y los lanza.
Retorno:	-

apagar(on_off_dialog, on_off_pregunta, on_off_img_prev, on_off_img_next, on_off_equipos, equipos_seleccionados, treestore_aulas)	
Objetivo:	Lanza *threads* que apagan los equipos seleccionados.
Argumentos:	Objetos *GtkDialog*, *GtkLabel* y *GtkImage* relacionados con el cuadro de confirmación del apagado de equipos, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Configura el cuadro de confirmación del apagado de equipos y lo muestra por pantalla. Tras obtener la confirmación del usuario adquiere el *mutex* asociado a la tabla de información de las aulas, crea un *thread* por cada equipo seleccionado y los lanza.
Retorno:	-
seleccionar_ninguno(treestore_aulas, aula)	
Objetivo:	Desmarca todos los equipos que se encuentren seleccionados.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes y los desmarca en la tabla de información de las aulas.
Retorno:	-
seleccionar_windows(treestore_aulas, aula)	
Objetivo:	Selecciona todos los equipos que se encuentren encendidos en *Windows*.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes marcando en la tabla de información de las aulas aquellos que se encuentren encendidos en *Windows*.
Retorno:	-
seleccionar_debian(treestore_aulas, aula)	
Objetivo:	Selecciona todos los equipos que se encuentren encendidos en *Debian*.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes marcando en la tabla de información de las aulas aquellos que se encuentren encendidos en *Debian*.
Retorno:	-

seleccionar_pxe(treestore_aulas, aula)	
Objetivo:	Selecciona todos los equipos que se encuentren encendidos en *PXE*.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes marcando en la tabla de información de las aulas aquellos que se encuentren encendidos en *PXE*.
Retorno:	-
seleccionar_encendidos(treestore_aulas, aula)	
Objetivo:	Selecciona todos los equipos que se encuentren encendidos.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes marcando en la tabla de información de las aulas aquellos que se encuentren en *Windows*, *Debian* o *PXE*.
Retorno:	-
seleccionar_apagados(treestore_aulas, aula)	
Objetivo:	Selecciona todos los equipos que se encuentren apagados.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes marcando en la tabla de información de las aulas aquellos que se encuentren apagados.
Retorno:	-
seleccionar_todos(treestore_aulas, aula)	
Objetivo:	Selecciona todos los equipos de las aulas.
Argumentos:	Objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas y objeto *Aula* para el caso en el que se aplique la tarea a un aula concreta.
Funcionamiento:	Comprueba si la selección se realiza sobre un aula concreta o sobre todas las aulas. Después recorre los equipos correspondientes y los marca en la tabla de información de las aulas.
Retorno:	-

lanzar_terminal(equipo)	
Objetivo:	Lanza un subproceso para abrir una conexión remota con la terminal de un equipo.
Argumentos:	Cadena de caracteres correspondiente al nombre del equipo con el que se abrirá la conexión.
Funcionamiento:	Comprueba si la conexión se debe llevar a cabo con el equipo recibido por parámetro o por el contrario se debe lanzar una terminal en el servidor donde se está ejecutando la aplicación. En ambos casos se genera un nuevo subproceso para abrir la conexión con la terminal del equipo.
Retorno:	-
copiar_mvs(kosmos_window, on_off_dialog, on_off_pregunta, on_off_img_prev, on_off_img_next, on_off_equipos, maquina_virtual, equipos_seleccionados, treestore_aulas)	
Objetivo:	Realiza las preparaciones pertinentes y ejecuta la tarea que transfiere una máquina virtual a los equipos seleccionados.
Argumentos:	Objeto *GtkWindow* correspondiente a la ventana de la aplicación, objetos *GtkDialog*, *GtkLabel* y *GtkImage* relacionados con el cuadro de confirmación del encendido de equipos, cadena de caracteres correspondiente a la máquina virtual que se transferirá a los equipos, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	En primer lugar comprueba si hay algún equipo seleccionado; en caso contrario muestra un mensaje de error. Tras esto comprueba si todos los equipos seleccionados se encuentran iniciados en *PXE*; en caso contrario muestra un mensaje de error indicando los equipos que no cumplen con dicha condición. Por último muestra un dialogo de confirmación tras el cual se ejecuta la tarea que transfiere una máquina virtual a los equipos seleccionados.
Retorno:	-

cambiar_permisos(kosmos_window, on_off_dialog, on_off_pregunta, on_off_img_prev, on_off_img_next, on_off_equipos, ruta_objeto, permisos, equipos_seleccionados, treestore_aulas)	
Objetivo:	Realiza las preparaciones pertinentes y ejecuta la tarea que cambia los permisos de los equipos seleccionados.
Argumentos:	Objeto *GtkWindow* correspondiente a la ventana de la aplicación, objetos *GtkDialog*, *GtkLabel* y *GtkImage* relacionados con el cuadro de confirmación del encendido de equipos, cadena de caracteres correspondiente a la ruta del objeto al que se le cambiarán los permisos, diccionario con la información de todos los usuarios y sus respectivos permisos, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	En primer lugar comprueba si hay algún equipo seleccionado; en caso contrario muestra un mensaje de error. Tras esto comprueba si todos los equipos seleccionados se encuentran iniciados en *Windows*; en caso contrario muestra un mensaje de error indicando los equipos que no cumplen con dicha condición. Por último muestra un dialogo de confirmación tras el cual se ejecuta la tarea que modifica los permisos en los equipos seleccionados.
Retorno:	-
instalar(kosmos_window, on_off_dialog, on_off_pregunta, on_off_img_prev, on_off_img_next, on_off_equipos, instalaciones_seleccionadas, equipos_seleccionados, treestore_aulas)	
Objetivo:	Realiza las preparaciones pertinentes y ejecuta la tarea que aplica una o más instalaciones en los equipos seleccionados.
Argumentos:	Objeto *GtkWindow* correspondiente a la ventana de la aplicación, objetos *GtkDialog*, *GtkLabel* y *GtkImage* relacionados con el cuadro de confirmación del encendido de equipos, lista de las instalaciones seleccionadas, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, así como un objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	En primer lugar comprueba si hay algún equipo seleccionado; en caso contrario muestra un mensaje de error. Tras esto comprueba si todos los equipos seleccionados se encuentran iniciados en *PXE*; en caso contrario muestra un mensaje de error indicando los equipos que no cumplen con dicha condición. Por último muestra un dialogo de confirmación tras el cual se ejecuta la tarea que lleva a cabo las instalaciones en los equipos seleccionados.
Retorno:	-

Tabla 63: Clase Tareas

La clase *Equipo* permite crear objetos que almacenan la información de los equipos de las aulas, como su nombre, su IP, el estado en el que se encuentra y el nombre del usuario que ha iniciado sesión en él.

Clase Equipo	
__init__(self)	
Objetivo:	Establece los valores iniciales al crear un objeto de la clase Equipo.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Inicializa los atributos de los objetos instanciados con valores por defecto.
Retorno:	-

Tabla 64: Clase Equipo

La clase *Aula* permite crear objetos que almacenan la información de un aula. Entre estos datos se encuentra el nombre, las instalaciones que se pueden realizar en dicha aula o una lista formada por objetos de tipo equipo correspondiente a los ordenadores de los que se compone.

Clase Aula	
__init__(self)	
Objetivo:	Establece los valores iniciales al crear un objeto de la clase Aula.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Inicializa los atributos de los objetos instanciados con valores por defecto.
Retorno:	-

Tabla 65: Clase Aula

La clase *Instalacion* permite crear objetos con la información relativa a una instalación. Estas instalaciones se almacenan en una lista en cada objeto de tipo aula en la que se pueden aplicar.

Clase Instalacion	
__init__(self)	
Objetivo:	Establece los valores iniciales al crear un objeto de la clase Instalacion.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Inicializa los atributos de los objetos instanciados con valores por defecto.
Retorno:	-

Tabla 66: Clase Instalacion

La clase *Terminal* permite crear una ventana nueva con conexión remota a la terminal de un equipo. También tiene un método que maneja su destrucción cuando el usuario la cierra.

Clase Terminal	
__init__(self, equipo)	
Objetivo:	Abre una conexión remota con la terminal de un equipo.
Argumentos:	Referencia al objeto instanciado (self) y cadena de caracteres correspondiente al nombre del equipo con el que se abrirá la conexión.
Funcionamiento:	Crea una ventana e inserta en ella una terminal que, en función del si el parámetro que recibe como argumento está vacío o contiene el nombre de un equipo, abre una terminal en el servidor en el que está ejecutando la aplicación o realiza una conexión remota por *SSH* al equipo especificado, respectivamente.
Retorno:	-
on_window_destroy(self)	
Objetivo:	Destruye la terminal cuando el usuario cierra la ventana.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Llama a la función correspondiente de Gtk que se encarga de cerrar la ventana de forma controlada.
Retorno:	-

Tabla 67: Clase Terminal

Diseño

La clase *ComprobarThread* se corresponde con el código del *thread* que realiza la comprobación del estado, la ocupación, la fecha y hora en que se encendió un equipo y la versión del kernel.

Clase ComprobarThread	
__init__(self, lock, equipo, num_aula, num_equipo, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con el equipo en el que se llevarán a cabo las comprobaciones.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, nombre del equipo a comprobar, número del aula en la que se encuentra el equipo, número del equipo en dicha aula y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales relativas al equipo en el cual se va a aplicar la tarea.
Retorno:	-
comprobar_estado(self, puerto=3389)	
Objetivo:	Obtiene el sistema operativo en el que se encuentra un equipo.
Argumentos:	Referencia al objeto instanciado (self) y número del puerto que se utilizará en el análisis con *NMAP*.
Funcionamiento:	Escanea el equipo mediante la utilidad *NMAP* con el objetivo de verificar si el puerto 3389 se encuentra abierto (*Windows*) o cerrado (*Linux*). En caso de detectar un sistema *Linux* accede a un directorio oculto en el equipo para comprobar si se encuentra en *Debian* o en *PXE*. En cualquier otro caso se analiza el puerto 24444 para confirmar que el equipo se encuentra apagado. Por último devuelve el estado.
Retorno:	Cadena de caracteres correspondiente al sistema operativo.
comprobar_ocupacion(self, estado)	
Objetivo:	Obtiene el nombre del usuario que se encuentra conectado a un equipo.
Argumentos:	Referencia al objeto instanciado (self) y cadena de caracteres correspondiente al sistema operativo en el que se encuentra el equipo.
Funcionamiento:	En función del sistema operativo en el que el equipo está encendido realiza una conexión remota mediante *Winexe* (*Windows*) o *SSH* (*Linux*) y ejecuta el comando correspondiente para obtener el nombre del usuario que se encuentra conectado.
Retorno:	Cadena de caracteres correspondiente al usuario.
comprobar_uptime(self, estado)	
Objetivo:	Obtiene la fecha y hora en que se encendió el equipo.
Argumentos:	Referencia al objeto instanciado (self) y cadena de caracteres correspondiente al sistema operativo en el que se encuentra el equipo.
Funcionamiento:	En función del sistema operativo en el que el equipo está encendido realiza una conexión remota mediante *Winexe* (*Windows*) o *SSH* (*Linux*) y ejecuta el comando correspondiente para obtener la fecha y la hora.
Retorno:	Cadena de caracteres correspondiente a la fecha y hora.

comprobar_version(self, estado)	
Objetivo:	Obtiene la versión del *kernel* del sistema operativo que se está ejecutando en un equipo.
Argumentos:	Referencia al objeto instanciado (self) y cadena de caracteres correspondiente al sistema operativo en el que se encuentra el equipo.
Funcionamiento:	En función del sistema operativo en el que el equipo está encendido realiza una conexión remota mediante *Winexe* (*Windows*) o *SSH* (*Linux*) y ejecuta el comando correspondiente para comprobar la versión del *kernel* del sistema operativo.
Retorno:	Cadena de caracteres correspondiente a la versión del *kernel*.
run(self, puerto=3389)	
Objetivo:	Ejecuta el *thread* que lleva a cabo las comprobaciones del estado, ocupación, fecha y hora en que se encendió el equipo y versión del *kernel*.
Argumentos:	Referencia al objeto instanciado (self) y número del puerto que se utilizará en el análisis con *NMAP*.
Funcionamiento:	Llama a *comprobar_estado*(), *comprobar_ocupacion*(), *comprobar_uptime*() y *comprobar_version*(), almacenando los valores que devuelven estos métodos en las variables globales definidas en *__init__*(). Tras esto adquiere el *mutex* asociado a la tabla de información de las aulas y escribe en el modelo los valores obtenidos.
Retorno:	-

Tabla 68: Clase ComprobarThread

La clase *EncenderThread* se corresponde con el código del *thread* que enciende los equipos de forma remota. Para ello hace uso de la utilidad *Etherwake*.

Clase EncenderThread	
__init__(self, lock, equipo, num_aula, num_equipo, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con el equipo en el que se llevará a cabo el encendido.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, nombre del equipo a encender, número del aula en la que se encuentra el equipo, número del equipo en dicha aula y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales relativas al equipo en el cual se va a aplicar la tarea.
Retorno:	-
run(self)	
Objetivo:	Ejecuta el *thread* que lleva a cabo el encendido de un equipo.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Envía un *paquete mágico* al equipo correspondiente mediante la utilidad Etherwake. Tras esto adquiere el *mutex* asociado a la tabla de información de las aulas y establece como *desconocido* el estado del equipo.
Retorno:	-

Tabla 69: Clase EncenderThread

La clase *ReiniciarThread* se corresponde con el código del *thread* que permite reiniciar los equipos en un sistema operativo concreto. Antes de reiniciar un equipo debe comprobar el sistema operativo en el que se encuentra para lanzar el comando correspondiente.

Clase ReiniciarThread	
__init__(self, lock, equipo, so, num_aula, num_equipo, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con el equipo en el que se llevará a cabo el reinicio.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, nombre del equipo a reiniciar, sistema operativo en el que deberá reiniciarse, número del aula en la que se encuentra el equipo, número del equipo en dicha aula y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales relativas al equipo en el cual se va a aplicar la tarea.
Retorno:	-
comprobar_so_equipo(self, equipo, puerto=3389)	
Objetivo:	Obtiene el sistema operativo en el que se encuentra un equipo.
Argumentos:	Referencia al objeto instanciado (self), cadena de caracteres correspondiente al nombre del equipo a comprobar y número del puerto que se utilizará en el análisis con *NMAP*.
Funcionamiento:	Escanea el equipo mediante la utilidad *NMAP* con el objetivo de verificar si el puerto 3389 se encuentra abierto (*Windows*) o cerrado (*Linux*). En caso de detectar un sistema *Linux* accede a un directorio oculto en el equipo para comprobar si se encuentra en *Debian* o en *PXE*. En cualquier otro caso se analiza el puerto 24444 para confirmar que el equipo se encuentra apagado. Por último devuelve el estado.
Retorno:	Cadena de caracteres correspondiente al sistema operativo.
run(self)	
Objetivo:	Ejecuta el *thread* que lleva a cabo el reiniciado de un equipo.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Comprueba el sistema operativo en el que se encuentra el equipo que se quiere reiniciar y en función de éste realiza una conexión remota mediante *Winexe* (*Windows*) o *SSH* (*Linux*) para ejecutar el comando correspondiente que reinicia el equipo en el sistema operativo escogido. Tras esto adquiere el *mutex* asociado a la tabla de información de las aulas y establece como *desconocido* el estado del equipo.
Retorno:	-

Tabla 70: Clase ReiniciarThread

La clase *ApagarThread* se corresponde con el código del *thread* que apaga los equipos de manera remota. Igual que al reiniciar un equipo se debe comprobar el sistema operativo en el que se encuentre el equipo a apagar para lanzar el comando correspondiente.

Clase ApagarThread	
__init__(self, lock, equipo, num_aula, num_equipo, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con el equipo en el que se llevará a cabo el apagado.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, nombre del equipo a apagar, número del aula en la que se encuentra el equipo, número del equipo en dicha aula y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales relativas al equipo en el cual se va a aplicar la tarea.
Retorno:	-
comprobar_so_equipo(self, equipo, puerto=3389)	
Objetivo:	Obtiene el sistema operativo en el que se encuentra un equipo.
Argumentos:	Referencia al objeto instanciado (self), cadena de caracteres correspondiente al nombre del equipo a comprobar y número del puerto que se utilizará en el análisis con *NMAP*.
Funcionamiento:	Escanea el equipo mediante la utilidad *NMAP* con el objetivo de verificar si el puerto 3389 se encuentra abierto (*Windows*) o cerrado (*Linux*). En caso de detectar un sistema *Linux* accede a un directorio oculto en el equipo para comprobar si se encuentra en *Debian* o en *PXE*. En cualquier otro caso se analiza el puerto 24444 para confirmar que el equipo se encuentra apagado. Por último devuelve el estado.
Retorno:	Cadena de caracteres correspondiente al sistema operativo.
run(self)	
Objetivo:	Ejecuta el *thread* que lleva a cabo el apagado de un equipo.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Comprueba el sistema operativo en el que se encuentra el equipo que se quiere apagar y en función de éste realiza una conexión remota mediante *Winexe* (*Windows*) o *SSH* (*Linux*) para ejecutar el comando correspondiente que apaga el equipo. Tras esto adquiere el *mutex* asociado a la tabla de información de las aulas y establece como *desconocido* el estado del equipo.
Retorno:	-

Tabla 71: Clase ApagarThread

La clase *CopiarMvsThread* se corresponde con el código del *thread* que transfiere las máquinas virtuales a los equipos de las aulas. En este caso la comprobación del sistema operativo en el que se encuentran los equipos no se realiza en el propio thread si no en la clase *Tareas*.

Clase CopiarMvsThread	
__init__(self, lock, maquina_virtual, equipo, num_aula, num_equipo, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con la copia de máquinas virtuales.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, cadena de caracteres correspondiente a la máquina virtual que se transferirá a los equipos, nombre del equipo al que se transferirá la máquina virtual, número del aula en la que se encuentra el equipo, número del equipo en dicha aula y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales para utilizarlas al copiar la máquina virtual.
Retorno:	-
run(self)	
Objetivo:	Ejecuta el *thread* que lleva a cabo la transferencia de una máquina virtual.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Ejecuta los scripts necesarios para realizar la transferencia de la máquina virtual a los equipos correspondientes.
Retorno:	-

Tabla 72: Clase CopiarMvsThread

La clase *CambiarPermisosThread* se corresponde con el código del *thread* que cambia los permisos de un fichero o directorio en los equipos de las aulas. La comprobación del sistema operativo en el que se encuentran los equipos no se realiza en el propio *thread* si no en la clase *Tareas*.

Clase CambiarPermisosThread	
__init__(self, lock, equipo, objeto, permisos, num_aula, num_equipo, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con el cambio de permisos en un equipo.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, cadena de caracteres correspondiente a la ruta del objeto al que se le cambiarán los permisos, diccionario con la información de todos los usuarios y sus respectivos permisos, nombre del equipo en el que se deben cambiar los permisos, número del aula en la que se encuentra el equipo, número del equipo en dicha aula y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales para utilizarlas al aplicar el cambio de permisos en un equipo.
Retorno:	-
run(self)	
Objetivo:	Ejecuta el *thread* que lleva a cabo el cambio de permisos.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	En función de los permisos que deben modificarse construye un comando de *icalcs* que se ejecuta de forma remota en los equipos seleccionados.
Retorno:	-

Tabla 73: Clase CambiarPermisosThread

Diseño

La clase *InstalarThread* se corresponde con el código del *thread* que permite aplicar instalaciones en los equipos de las aulas. Al igual que en las dos clases anteriores la comprobación del sistema operativo en el que se encuentran los equipos se realiza en la clase *Tareas*.

Clase InstalarThread	
__init__(self, lock, equipos_seleccionados, instalaciones_seleccionadas, treestore)	
Objetivo:	Almacena en variables globales los valores relacionados con las instalaciones que se llevarán a cabo.
Argumentos:	Referencia al objeto instanciado (self), *mutex* asociado a la tabla de información de las aulas, diccionario de pares clave-valor con el nombre de los equipos y su posición y número de aula a la cual pertenecen, lista de las instalaciones seleccionadas y objeto *GtkTreeStore* correspondiente al modelo que almacena los datos de las aulas.
Funcionamiento:	Asigna los parámetros recibidos a variables globales para utilizarlas al aplicar las instalaciones correspondientes.
Retorno:	-
run(self)	
Objetivo:	Ejecuta el *thread* que lleva a cabo las correspondientes instalaciones.
Argumentos:	Referencia al objeto instanciado (self).
Funcionamiento:	Por cada una de las instalaciones seleccionadas ejecuta un script que aplica dicha instalación en los equipos correspondientes.
Retorno:	-

Tabla 74: Clase InstalarThread

4.4. Diseño físico de datos

Dada la limitada complejidad y el reducido volumen de los datos con los que trata la aplicación se decidió utilizar ficheros estructurados mediante lenguajes de marcado en lugar de una base de datos al uso.

Se estudiaron dos alternativas para almacenar estos datos de forma persistente: **XML** y **JSON**. Aunque en muchos aspectos no hay gran diferencia entre ambos formatos, *JSON* destaca en términos de simplicidad y legibilidad, así como en velocidad de procesamiento de datos. Además, los archivos *JSON* también tienden a ser más pequeños en tamaño comparados con los *XML*. No obstante, *XML* puede albergar estructuras de datos mucho más complejas y está diseñado para poder añadir metadatos (característica que permite el almacenamiento de atributos adicionales como la localización de un aula, número de filas y columnas, etc.), posibilidad que no existe en *JSON* (aunque podría simularse).

Por otra parte, la información que debe almacenar la aplicación puede dividirse en dos tipos de datos:

- Datos relacionados con la **configuración general de Kosmos**. Incluyen algunos parámetros concretos que permiten conservar la configuración establecida en anteriores sesiones, como el directorio del cual se cargarán las máquinas virtuales o los permisos por defecto que se asignan a una carpeta. Estos datos se guardan en un archivo llamado **kosmos.xml** que hay dentro de la carpeta de recursos "*rsc*".
- Datos relacionados con la **composición de las aulas**. Incluyen información relativa a las características de los equipos de cada aula y de las propias aulas. Esta se almacena en un archivo llamado **aulas.xml** que hay dentro de la carpeta "*rsc*".

El diseño elegido para estructurar la información del fichero de almacenamiento de la configuración general de Kosmos es el siguiente:

kosmos.xml
<kosmos>   <maquinas-virtuales>**Directorio de máquinas virtuales**</maquinas-virtuales>   <permisos-generales>   <usuario>   <nombre>**Nombre del usuario**</nombre>   <permiso permitir="**True** | **False**">**Nombre del permiso**</permiso>   </usuario>   </permisos-generales> </kosmos>

Tabla 75: Diseño del fichero de configuración kosmos.xml

Diseño

El diseño elegido para estructurar el fichero que contiene la información de las aulas es el siguiente:

aulas.xml

```
<aulas>
    <aula>
        <nombre>Nombre del aula</nombre>
        <direccion>Dirección del aula</direccion>
        <filas>Número de filas del aula</filas>
        <columnas>Número de columnas del aula</columnas>
        <equipos>
            <equipo ip="Dirección IP del equipo">Nombre del equipo</equipo>
        </equipos>
        <instalaciones>
            <instalación nombre="Nombre de la instalación" parámetro-de-instalación=”valor”/>
        </instalaciones>
    </aula>
</aulas>
```

Tabla 76: Diseño del fichero aulas.xml

Al cargar la aplicación estos datos se almacenan en una estructura auxiliar de *GTK* llamada *TreeStore* que se corresponde con el subsistema de la capa de datos (modelo) visto en el apartado *4.1 Definición de la arquitectura del sistema*.

4.5. Revisión de la interfaz de usuario

En esta sección se llevará a cabo una descripción detallada de la interfaz gráfica diseñada para la aplicación Kosmos. Está interfaz se compone de una sola ventana estructurada en dos partes bien diferenciadas, cuyo tamaño puede modificarse arrastrando el borde que las separa:

- **Tabla de información de las aulas**: muestra la información relativa a los equipos de las aulas desde la última actualización de estado y ocupación.
- **Grupo de utilidades de gestión avanzada**: permiten llevar a cabo tareas más complejas y que se realizan con menos frecuencia en el Laboratorio, como el copiado de máquinas virtuales, la asignación de permisos a directorios de *Windows* o la realización de instalaciones.

4.5.1. Tabla de información de las aulas

La tabla de información de las aulas se corresponde con la mitad superior de la interfaz, y se encarga de mostrar el estado actual de los equipos de las aulas y la información de las tareas que se estén realizando o se hayan realizado en ellos. Además, incluye un mapa que muestra la distribución de los equipos en cada una de las aulas.

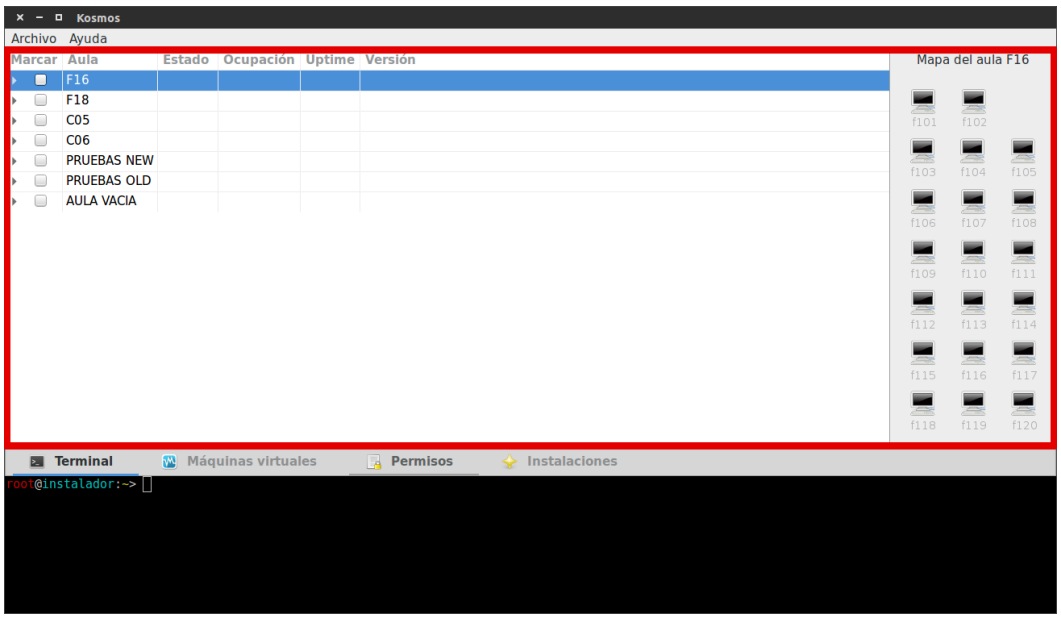


Ilustración 27: Tabla de información de las aulas

Las aulas se disponen en forma de lista y como elementos desplegables que incluyen a su vez una fila por cada uno de los equipos que las componen. Junto al nombre de cada aula o equipo, hay un *checkbox* que permite seleccionarlos. En el caso de marcar un aula, se seleccionan automáticamente todos los equipos de dicha aula.

Ilustración 28: Selección de varios equipos en un aula

Diseño

La tabla de información de las aulas se divide a su vez en columnas que muestran la información correspondiente al estado, los usuarios que han iniciado sesión, la fecha y hora en que se encendió y la versión del *kernel* del sistema operativo en el cual se encuentra encendido cada equipo.

Marcar	Aula	Estado	Ocupación	Uptime	Versión
☑	F16				
☑	f101	debian	cmedina	2016-06-13 17:27:07	Linux 3.16.0-4-amd64
☑	f102	apagado			
☑	f103	windows		2016-06-13 15:29:48	Microsoft Windows [Versión 6.1.7601]
☑	f104	windows		2016-06-13 17:29:47	Microsoft Windows [Versión 6.1.7601]
☑	f105	apagado			
☑	f106	debian		2016-06-13 17:27:07	Linux 3.16.0-4-amd64
☑	f107	debian		2016-06-13 17:27:08	Linux 3.16.0-4-amd64
☑	f108	debian		2016-06-13 17:27:07	Linux 3.16.0-4-amd64

Ilustración 29: Información que puede obtenerse de los equipos en la tabla

Desde esta tabla también se pueden ejecutar las tareas más básicas que se llevan a cabo en el Laboratorio, como son el encendido, apagado, reiniciado y comprobación del estado de los equipos, además de la apertura de una terminal remota. El menú que incluye estas tareas se abre pinchando con el botón derecho del ratón sobre un aula, un equipo, o cualquier otro lugar de la tabla.

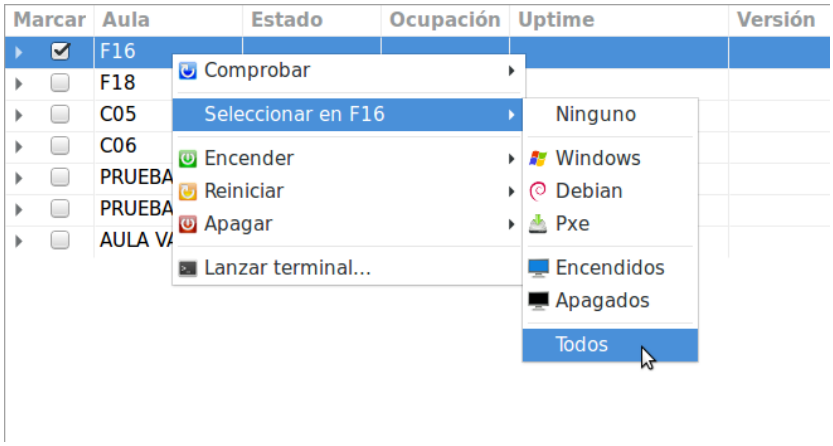


Ilustración 30: Menú desplegable con el botón derecho del ratón

En la parte derecha de la tabla se muestra el mapa correspondiente a la última aula sobre la cual se pinchó con el ratón (o al aula a la que pertenece el último equipo que se seleccionó). Este mapa se actualiza cada vez que se selecciona un equipo del aula, resaltando su nombre en rojo.

Ilustración 31: Mapa de distribución de los equipos en un aula

4.5.2. Grupo de utilidades de gestión avanzada

El grupo de utilidades de gestión avanzada se corresponde con la mitad inferior de la interfaz, la cual está formada por una serie de pestañas que permiten llevar a cabo cada una de las tareas avanzadas. Estas tareas incluyen una terminal integrada, una herramienta para copiar máquinas virtuales, otra para cambiar los permisos y una última para realizar instalaciones en los equipos de las aulas.

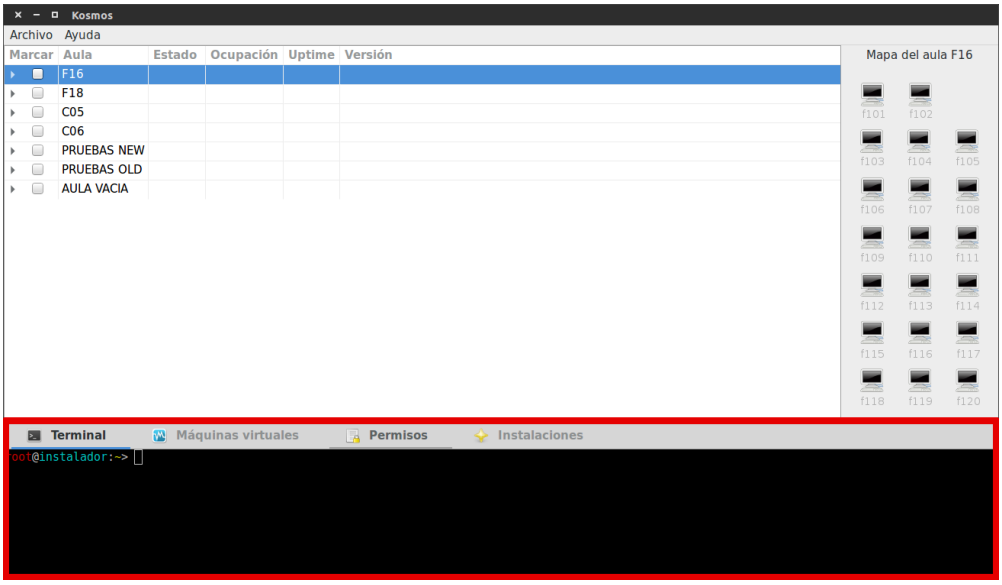


Ilustración 32: Grupo de utilidades de gestión avanzada

La terminal integrada ocupa el espacio completo de la pestaña y permite ejecutar cualquier comando en el servidor en el que se aloja la aplicación.

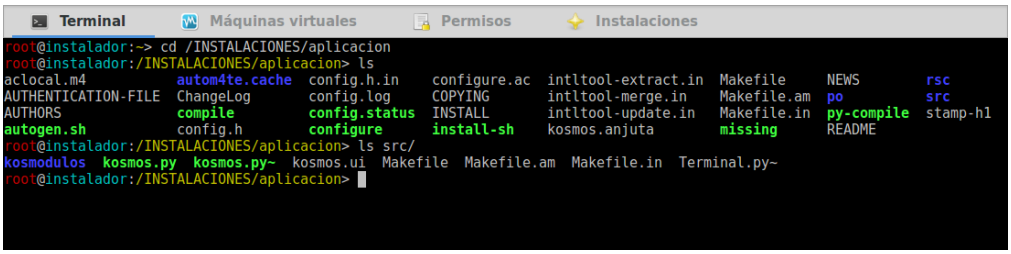


Ilustración 33: Pestaña de la terminal integrada

Diseño

La herramienta de transferencia de máquinas virtuales se compone de una lista desplegable y un botón que permiten seleccionar la máquina virtual que se copiará y el directorio del cual se obtienen las máquinas virtuales del desplegable, respectivamente. En la parte derecha de la pestaña se incluyen dos botones comunes a todas las utilidades de gestión avanzada (excepto la terminal integrada) que permiten lanzar el proceso y detenerlo.

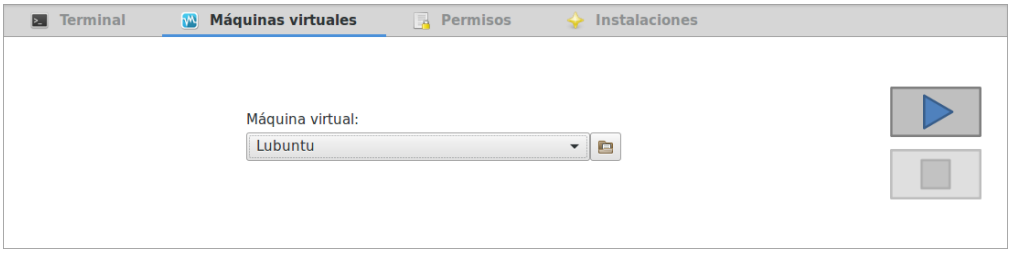


Ilustración 34: Pestaña de transferencia de máquinas virtuales

La pestaña de permisos incluye un cuadro de texto que permite introducir la ruta del fichero o directorio al cual se modificarán los permisos, dos botones de radio que permiten elegir si deben aplicarse los permisos por defecto o los que defina el usuario y dos tablas en las que se pueden configurar dichos permisos. Bajo estas tablas también hay tres botones que permiten agregar o eliminar nuevos grupos o usuarios a los que conceder permisos y guardar la configuración actual como permisos por defecto.

Aunque en principio la funcionalidad de cambio de permisos iba a estar integrada en la herramienta de transferencia de máquinas virtuales, finalmente se decidió separarla en dos herramientas distintas para ofrecer la posibilidad de cambiar los permisos a cualquier directorio sin tener que estar asociado a la copia de una máquina virtual.

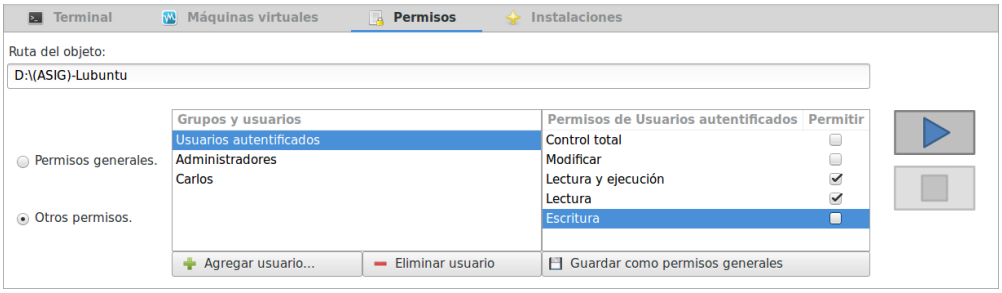


Ilustración 35: Pestaña de cambio de permisos

La última pestaña se corresponde con la utilidad de instalaciones. Esta herramienta está compuesta simplemente del nombre de las instalaciones que pueden realizarse en los equipos y un *checkbox* que permite seleccionarlas.

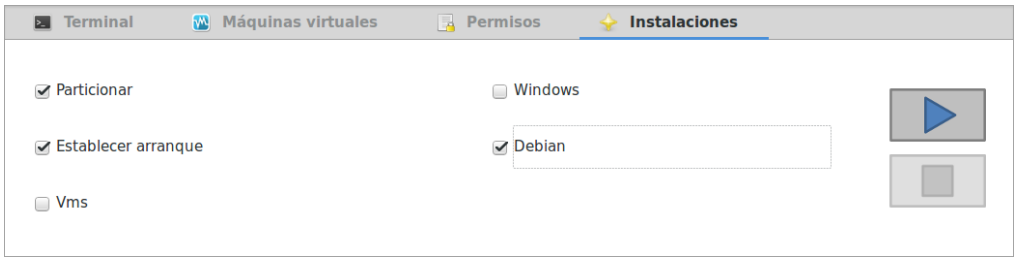


Ilustración 36: Pestaña de instalaciones

4.5.3. Cuadros de diálogo adicionales

Como se ha comentado con anterioridad, la interfaz de la aplicación consta de una sola ventana. No obstante, en ocasiones concretas sí se generan cuadros de diálogo adicionales, como la ventana de conexión con la terminal remota de un equipo:

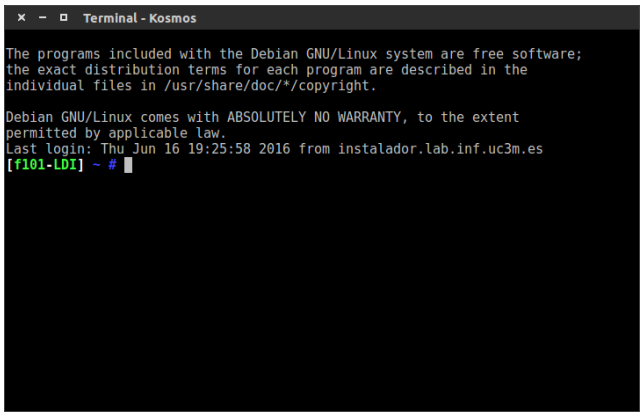


Ilustración 37: Ventana de conexión con la terminal remota de un equipo

También se genera un cuadro de diálogo adicional al reiniciar los equipos, el cual permite seleccionar el sistema operativo en el que iniciarán:

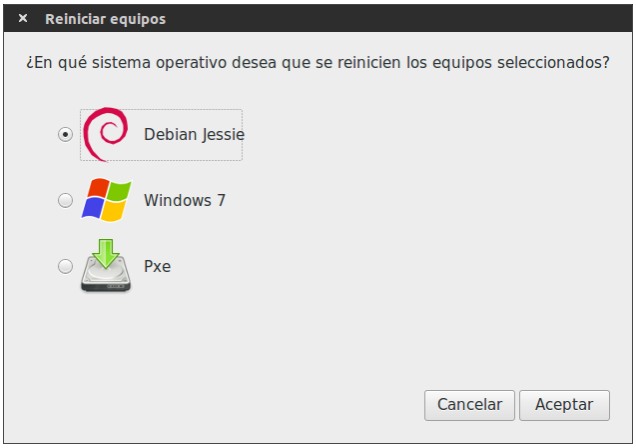


Ilustración 38: Cuadro de diálogo para seleccionar el sistema operativo

O los cuadros de diálogo que permiten confirmar acciones delicadas como la realización de instalaciones, el cambio de permisos o la transferencia de máquinas virtuales:

Ilustración 39: Cuadro de confirmación al copiar máquinas virtuales

5. Implementación

En este apartado se describe el proceso de preparación e instalación del entorno de operación, así como la configuración del entorno de desarrollo con el que se lleva a cabo la implementación del sistema.

5.1. Configuración del entorno operacional

A continuación se describen las fases de instalación y configuración del servidor que aloja la aplicación y de los dos ordenadores portátiles que servirán de interfaz gráfica para la instalación de los equipos. Por último, se describen algunas configuraciones particulares que se llevaron a cabo en el IDE antes de pasar a la fase de codificación.

5.1.1. Servidor Instalador

Durante el proceso de instalación del servidor se decidió que la organización de las particiones dispusiera de un sistema de almacenamiento **RAID1** con redundancia en espejo para alojar el sistema operativo. De esta forma, se llevaron a cabo dos particiones en los discos **sda** y **sdb**: una primaria con sistema de ficheros *ext4* y otra que haría las funciones de área de intercambio *swap*. Sobre este *RAID1* se instaló una distribución **Debian GNU/Linux 8.4 (Jessie)**.

El resto de discos duros (**sdc**, **sdd** y **sde**) se configuraron como un sistema de almacenamiento **RAID0** y se asignaron a la partición **INSTALACIONES**. Esta partición se dedicó única y exclusivamente a las tareas de gestión de los equipos y se diseñó con la siguiente estructura de directorios:

- **aplicacion**: en este directorio se alojaría la utilidad desarrollada en el presente proyecto.
- **imagenes**: en este directorio se almacenarían las imágenes de los sistemas operativos que se transfieren a los equipos de las aulas en el proceso de instalación.
- **pxe**: aquí se alojaría la imagen de red que se exporta a los equipos durante el arranque por red. Esta imagen se genera con la utilidad live-build.
- **generar_cliente**: aquí se almacenarían los scripts necesarios para actualizar un cliente *PXE* de arranque por red.

Además de otras configuraciones adicionales como las relativas a las fuentes de repositorios de *Debian*, a la conexión de red o a las *iptables*, también hubo que instalar algunos paquetes necesarios para el correcto funcionamiento de la aplicación Kosmos. Estos paquetes incluyen:

- **nmap**: esencial para la comprobación remota del sistema operativo de los equipos.
- **ntfs-3g**: necesario para trabajar con sistemas de ficheros *NTFS* (*Windows*).
- **nfs-kernel-server**: servidor *NFS* necesario para exportar directorios por red.
- **winexe**: aplicación que permite ejecutar comandos de forma remota en sistemas *Windows*.
- **udpcast**: este servicio permite transmitir datos a varios destinos al mismo tiempo dentro de una misma red LAN. Incluye las utilidades **udp-receiver** y **udp-sender**, encargadas de recibirlos y enviarlos respectivamente. Kosmos utiliza este servicio para transferir las imágenes de los sistemas operativos al realizar las correspondientes instalaciones.

También son necesarios algunos paquetes de los que depende la herramienta *PXE*:

- **tftpd-hpa**: servidor *tftp* necesario para arrancar un sistema operativo por red.
- **openbsd-inet**: demonio encargado de manejar las conexiones entrantes de un equipo.
- **grub-efi-amd64**: necesario para generar el sistema de arranque en equipos con sistema de partición EFI.

Por último se instalaron algunos paquetes esenciales para el funcionamiento de Kosmos como Python, ciertos módulos concretos como el VTE (para le integración de una terminal virtual en la aplicación) o la biblioteca gráfica GTK+:

- **python3**
- **python3-gi**
- **python-vte**
- **gir1.2-vte-2.91**
- **python-gtk2**
- **gir1.2-gtk-3.0**

5.1.2. Equipos Toleman y Minardi

También poseerán una partición dedicada a las **INSTALACIONES** con la siguiente estructura de directorios:

- **aplicacion**: en este directorio se alojará la utilidad Kosmos.
- **imagenes**: en este directorio se almacenarán las imágenes de los sistemas operativos que se transfieren a los equipos de las aulas en el proceso de instalación.
- **scripts**: en este directorio se alojarán los scripts de sincronización de los anteriores directorios con el servidor Instalador.

5.1.3. Configuración del IDE

Aunque la configuración del entorno de desarrollo no fue excesivamente compleja, sí que fue necesario establecer algunos parámetros concretos para facilitar el posterior desarrollo de la aplicación. Un ejemplo de esto fue la elección en Anjuta del tipo de proyecto que se iba a realizar. Para ello se eligió **PyGTK (automake)**, puesto que genera automáticamente un código inicial de ejemplo asociado a una interfaz gráfica a partir del cual es mucho más sencillo codificar la aplicación.

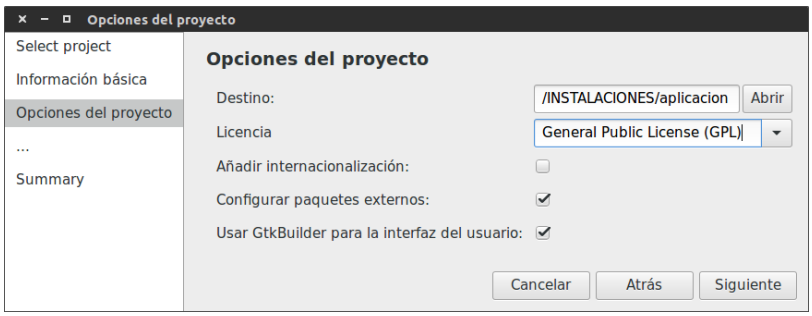


Ilustración 40: Configuración del proyecto en Anjuta

También es importante indicar el **directorio donde se almacenará el proyecto** de Anjuta y el **tipo de licencia** con el que se distribuirá la aplicación. Con esta última opción el propio editor genera un texto con la información básica de la licencia escogida y la agrega al comienzo de cada clase creada. Marcando la opción de **configuración de paquetes externos** nos permitirá seleccionar posteriormente dichos paquetes para que los importe automáticamente. La opción de usar **GtkBuilder** permite construir la interfaz de las aplicaciones y almacenarla en un fichero de marcas (muy similar a *XML*) y acceder más fácilmente desde el código de la aplicación a los objetos de dicha interfaz.

Además se creyó conveniente utilizar la versión más actualizada de la biblioteca gráfica GTK+. Para ello Anjuta ofrece la posibilidad de escoger entre 7 versiones diferentes de esta, de la cual se eligió la 3.12.

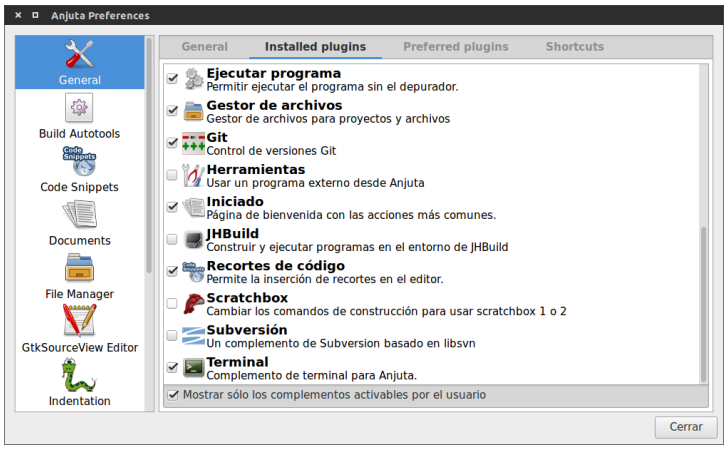


Ilustración 41: Complementos que ofrece Anjuta

Para facilitar el desarrollo del proyecto también se aprovechó una de las características más útiles que ofrece Anjuta: la capacidad de instalar complementos que añaden funcionalidades adicionales al entorno de desarrollo. Entre los complementos que se activaron se encontraban un complemento para la construcción automática de la aplicación con *autotools*, un gestor de archivos y proyectos, una terminal integrada y un complemento de *Git* para el control de versiones.

5.2. Implementación del código

Para una correcta implementación del código de la aplicación se decidió estructurar el proyecto en un árbol de directorios como el que se define a continuación. Las carpetas que no se enumeran aquí no se encontraban incluidas en el diseño original ya que fueron generadas por el propio entorno de desarrollo:

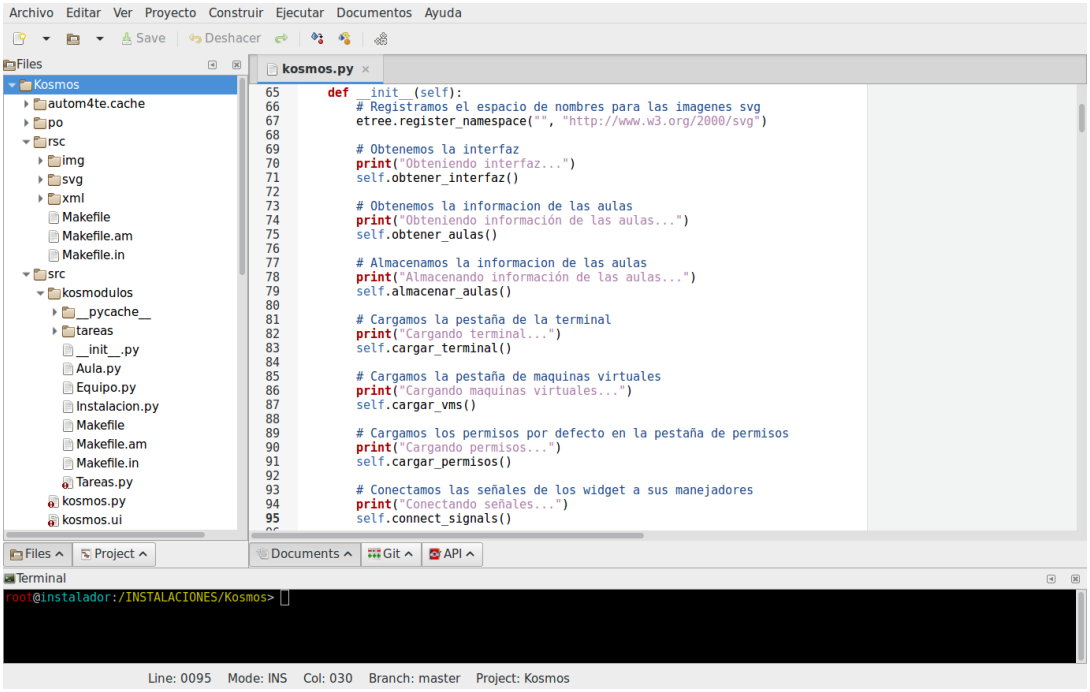


Ilustración 42: Implementación del código de la aplicación con Anjuta

Kosmos (directorio principal del proyecto):

- **rsc**: se corresponde con el directorio de recursos que utiliza la aplicación. Se compone a su vez de las siguientes carpetas:
  - **img**: directorio de imágenes de mapa de bits.
  - **svg**: directorio de imágenes vectoriales.
  - **xml**: directorio de almacenamiento de archivos con extensión *XML*. Incluye el fichero de configuración general de Kosmos (*kosmos.xml*) y el fichero que contiene la información de las aulas.
- **src**: contiene los ficheros de código de la aplicación. Incluye además una carpeta con las clases principales de la aplicación:
  - **kosmodulos**: Además de las clases Aula.py, Equipo.py, Instalacion.py y Tareas.py se incluye otro directorio:
    - **tareas**: en esta carpeta se encuentran las clases asociadas a cada una de las tareas que puede llevar a cabo la aplicación.

Por último cabe destacar que el código de la aplicación Kosmos es software libre, es decir, se distribuye bajo los términos de la **Licencia Pública General de GNU versión 3**[19] (GPL3). Para acceder a dicho código se habilita un repositorio online en la siguiente dirección:

https://bitbucket.org/carlosmesan/kosmos

6. Evaluación

Este apartado engloba las diversas pruebas que permiten valorar la calidad del sistema y verificar si el sistema cumple las necesidades establecidas por el cliente.

6.1. Especificación del plan de pruebas

Con el objetivo de evaluar el cumplimiento de los requisitos del sistema se aplicará una batería de pruebas sobre la aplicación desarrollada. Desde una perspectiva general, estas pruebas pueden clasificarse en dos tipos[20] [21]:

- **Pruebas de caja blanca**. Son pruebas que verifican aspectos relacionados con el código fuente de un componente concreto. Por ejemplo, suelen utilizarse para comprobar el funcionamiento de cada uno de los métodos de una clase. Debido a su complejidad (puesto que son mucho más rigurosas y detalladas) y a que su ausencia puede suplirse con pruebas de caja negra, no han sido incluidas en el plan de pruebas de este proyecto.
- **Pruebas de caja negra**. Comprueban que el sistema produce la salida esperada en función de las entradas que recibe, todo ello sin centrarse en aspectos del código fuente (que aparece como una caja negra, de ahí su nombre).

Por otra parte, el plan de pruebas del proyecto se ha dividido en cuatro grupos en función del nivel de prueba correspondiente:

- **Pruebas unitarias**. Sirven para comprobar el correcto funcionamiento de un módulo concreto de código.
- **Pruebas de integración**. Tienen como objetivo evaluar que los distintos componentes del sistema funcionan de manera correcta unos con otros.
- **Pruebas de implantación**. Sirven para verificar que el sistema se ha instalado correctamente. La especificación técnica de estas pruebas se ha ubicado tras la fase de implementación, en el apartado *7.4.1 Pruebas de implantación*.
- **Pruebas de aceptación**. Tienen como objetivo comprobar que el sistema cumple los requisitos establecidos por el cliente y se encuentra preparado para el paso a producción. La especificación técnica de estas pruebas se ha ubicado en el apartado *7.4.2 Pruebas de aceptación*.

6.2. Especificación técnica del plan de pruebas

Cada prueba estará definida en una tabla como la que se presenta a continuación:

Identificador	
Objetivo:	
Necesidades del entorno:	
Clase:	
Método:	
Entradas:	
Secuencia:	
Salidas:	
Requisitos relacionados:	

Tabla 77: Plantilla para pruebas

En la cual cada campo especificará la siguiente información:

- **Identificador**: será un código asociado a cada prueba que la diferenciará unívocamente. Este identificador tendrá el formato **PX-NN** donde:
  - **X**: indica el tipo de prueba, que puede ser:
    - **U**: prueba unitaria.
    - **N**: prueba de integración.
    - **M**: prueba de implantación.
  - **NN**: es un número consecutivo del 01 al 99.
- **Objetivo**: definirá el propósito de la prueba.
- **Necesidades del entorno**: definirá las condiciones que deben cumplirse antes de llevar a cabo la prueba.
- **Clase**: indicará la clase a la que pertenece el método relativo a una prueba unitaria.
- **Método**: indicará el nombre del método que se evalúa en una prueba unitaria.
- **Entradas**: identificará los valores de entrada que deben proporcionarse para la realización de la prueba.
- **Secuencia**: enumerará los pasos a realizar para llevar a cabo la prueba.
- **Salidas**: indicará los valores de salida generados por el sistema tras realizar la prueba.
- **Requisitos relacionados**: requisitos de usuario de los que se ha derivado la prueba y cuya implementación se pretende verificar.

6.2.1. Pruebas unitarias

A continuación se enumeran las pruebas unitarias, las cuales permiten verificar el correcto funcionamiento de algunos métodos concretos:

PU-01	
Objetivo:	Comprobar que el método que obtiene el estado de un equipo devuelve una cadena de caracteres que se corresponde con su estado real.
Necesidades del entorno:	Equipos de pruebas en todos los estados disponibles.
Clase:	ComprobarThread
Método:	*comprobar_estado(self, puerto=3389)*
Entradas:	Número entero correspondiente al puerto sobre el que se llevará a cabo el examen de puertos abiertos.
Salidas:	Cadena de caracteres correspondiente al estado en el que se encuentra el equipo.
Requisitos relacionados:	UC-01, SF-01

Tabla 78: Prueba unitaria PU-01

PU-02	
Objetivo:	Comprobar que el método que obtiene la ocupación de un equipo devuelve una cadena de caracteres que se corresponde con el nombre del usuario que ha iniciado sesión en el equipo.
Necesidades del entorno:	Distintos usuarios con sesión iniciada en varios equipos.
Clase:	ComprobarThread
Método:	*comprobar_ocupacion(self, estado)*
Entradas:	Cadena de caracteres correspondiente al estado en el que se encuentra el equipo.
Salidas:	Cadena de caracteres correspondiente al nombre de usuario que tiene iniciada una sesión en el equipo.
Requisitos relacionados:	UC-02, SF-02

Tabla 79: Prueba unitaria PU-02

Evaluación

PU-03	
Objetivo:	Comprobar que el método que obtiene la fecha y hora en la que se encendió un equipo devuelve una cadena de caracteres que se corresponde con dichos datos.
Necesidades del entorno:	Equipos de pruebas encendidos en distinta fecha y hora.
Clase:	ComprobarThread
Método:	*comprobar_uptime(self, estado)*
Entradas:	Cadena de caracteres correspondiente al estado en el que se encuentra el equipo.
Salidas:	Cadena de caracteres correspondiente a la fecha y hora en que se encendió el equipo.
Requisitos relacionados:	UC-02, SF-02

Tabla 80: Prueba unitaria PU-03

PU-04	
Objetivo:	Comprobar que el método que obtiene la versión del *kernel* de un sistema operativo devuelve una cadena de caracteres que se corresponde con dichos datos.
Necesidades del entorno:	Equipos de pruebas encendidos en distintos sistemas operativos.
Clase:	ComprobarThread
Método:	*comprobar_version(self, estado)*
Entradas:	Cadena de caracteres correspondiente al estado en el que se encuentra el equipo.
Salidas:	Cadena de caracteres correspondiente a la versión del *kernel* del sistema operativo.
Requisitos relacionados:	UC-02, SF-02

Tabla 81: Prueba unitaria PU-04

Evaluación

PU-05	
Objetivo:	Comprobar que el método que obtiene el nombre de los equipos seleccionados realiza dicha tarea correctamente.
Necesidades del entorno:	Equipos de pruebas seleccionados en varias aulas.
Clase:	Kosmos
Método:	*obtener_equipos_seleccionados(self)*
Entradas:	Ninguna.
Salidas:	Diccionario de pares clave-valor con el nombre de los equipos seleccionados y su posición y número de aula a la cual pertenecen.
Requisitos relacionados:	UC-03, SF-03

Tabla 82: Prueba unitaria PU-05

PU-06	
Objetivo:	Comprobar que el método que obtiene el nombre de todos los equipos realiza dicha tarea correctamente.
Necesidades del entorno:	Ninguna.
Clase:	Kosmos
Método:	*obtener_todos_equipos(self)*
Entradas:	Ninguna.
Salidas:	Diccionario de pares clave-valor con el nombre de todos los equipos de las aulas y su posición y número de aula a la cual pertenecen.
Requisitos relacionados:	UC-03, SF-03

Tabla 83: Prueba unitaria PU-06

Evaluación

PU-07	
Objetivo:	Comprobar que el método que actualiza el mapa de aulas realiza dicha tarea correctamente.
Necesidades del entorno:	Equipo de pruebas seleccionado en la tabla de información de las aulas.
Clase:	Kosmos
Método:	*actualizar_mapa(self, treeview_selection)*
Entradas:	Objeto TreeViewSelection que se corresponde con la selección realizada en la tabla de información de las aulas.
Salidas:	Ninguna.
Requisitos relacionados:	UC-11, SF-15

Tabla 84: Prueba unitaria PU-07

6.2.2. Pruebas de integración

A continuación se enumeran las pruebas de integración, que al contrario que las pruebas unitarias permiten verificar el funcionamiento de la aplicación a más alto nivel:

PN-01	
Objetivo:	Comprobar que se seleccionan los equipos correspondientes al pinchar en cada una de las opciones de selección del menú desplegable con el botón derecho del ratón.
Necesidades del entorno:	Equipos de pruebas en todos los estados disponibles.
Entradas:	Ninguna.
Secuencia:	Obtener el estado/ocupación de todos los equipos de las aulas. Pinchar en una condición de selección del menú desplegable con el botón derecho del ratón. Comprobar que se han seleccionado los equipos correctos. Repetir los pasos [2] y [3] con cada una de las condiciones de selección.
Salidas:	Equipos seleccionados correspondientes a cada condición de selección.
Requisitos relacionados:	UC-03, SF-03

Tabla 85: Prueba de integración PN-01

Evaluación

PN-02	
Objetivo:	Comprobar que un equipo es capaz de encenderse, reiniciarse en cada uno de los tres sistemas operativos disponibles y apagarse de forma remota.
Necesidades del entorno:	Equipo de pruebas apagado.
Entradas:	Ninguna.
Secuencia:	Seleccionar el equipo sobre el que se realizará la prueba. Abrir el menú desplegable con el botón derecho del ratón y encender el equipo de forma remota. Comprobar que el equipo se ha encendido. Reiniciar el equipo de forma remota en los tres sistemas operativos disponibles. Comprobar que el equipo se ha encendido en los sistemas operativos correspondientes. Apagar el equipo de forma remota. Comprobar que el equipo se ha apagado.
Salidas:	Equipo en el estado correspondiente a la acción previa realizada.
Requisitos relacionados:	UC-04, SF-04, SF-05, SF-06

Tabla 86: Prueba de integración PN-02

Evaluación

PN-03	
Objetivo:	Comprobar que se pueden ejecutar comandos en la terminal remota de un equipo tras ejecutar la orden de lanzar una terminal.
Necesidades del entorno:	Equipo de pruebas encendido en *Debian*.
Entradas:	Comando de prueba.
Secuencia:	Abrir el menú desplegable pinchando con el botón derecho del ratón sobre el equipo de pruebas y seleccionar la opción *Lanzar terminal*. Ejecutar el comando *hostname* en la terminal abierta. Comprobar que la salida se corresponde con el nombre del equipo sobre el que se ha lanzado la terminal.
Salidas:	Ventana de terminal con conexión remota a la consola de comandos del equipo y salida del comando de prueba.
Requisitos relacionados:	UC-05, SF-07

Tabla 87: Prueba de integración PN-03

PN-04	
Objetivo:	Comprobar que se pueden ejecutar comandos en el servidor donde se aloja la aplicación.
Necesidades del entorno:	Ninguna.
Entradas:	Comando de prueba.
Secuencia:	Ejecutar el comando *hostname* en la pestaña de la terminal del grupo de utilidades de gestión avanzada. Comprobar que la salida se corresponde con el nombre del servidor en el cual se ejecuta la aplicación.
Salidas:	Salida del comando de prueba.
Requisitos relacionados:	UC-06, SF-08

Tabla 88: Prueba de integración PN-04

Evaluación

PN-05	
Objetivo:	Comprobar que los ficheros y directorios de una máquina virtual se transfieren de forma íntegra al equipo seleccionado cuando se ejecuta la orden de copiar una máquina virtual.
Necesidades del entorno:	Equipo de pruebas encendido en *PXE*.
Entradas:	Máquina virtual a transferir.
Secuencia:	Seleccionar el equipo de pruebas. Seleccionar la máquina virtual que se va a transferir. Lanzar el proceso de transferencia. Comprobar que la máquina virtual se ha copiado de forma íntegra.
Salidas:	Máquina virtual transferida al equipo.
Requisitos relacionados:	UC-07, SF-09

Tabla 89: Prueba de integración PN-05

PN-06	
Objetivo:	Comprobar que los permisos de un directorio de *Windows* se han modificado convenientemente en el equipo seleccionado al ejecutar la orden de cambiar los permisos de dicho directorio.
Necesidades del entorno:	Equipo de pruebas encendido en *Windows*.
Entradas:	Ruta del fichero/directorio al que se quieren modificar los permisos así como los grupos y usuarios y el valor de los permisos que se quieren cambiar.
Secuencia:	Seleccionar el equipo de pruebas. Escribir la ruta del objeto al que se quieren cambiar los permisos. Agregar o eliminar los grupos y usuarios a los que se quieren modificar los permisos. Modificar el valor de los permisos. Lanzar el proceso de cambio de permisos. Comprobar que los permisos se han modificado correctamente.
Salidas:	Permisos del fichero/directorio modificados.
Requisitos relacionados:	UC-08, SF-10

Tabla 90: Prueba de integración PN-06

Evaluación

PN-07	
Objetivo:	Comprobar que las instalaciones *Particionar (crear particiones)*, *Establecer arranque* y transferencia de *Máquinas virtuales*, *Debian* y *Windows* se realizan correctamente en un equipo.
Necesidades del entorno:	Equipo de pruebas encendido en *PXE*.
Entradas:	Instalaciones a realizar.
Secuencia:	Seleccionar el equipo de pruebas. Marcar las opciones *Particionar*, *Establecer arranque*, *Vms*, *Debian* y *Windows* en la pestaña de instalaciones. Lanzar el proceso de instalación. Comprobar que el equipo lanza la ventana de selección del sistema operativo al inicio y todos ellos se ejecutan correctamente, además de tener almacenadas todas las máquinas virtuales en la partición correspondiente.
Salidas:	Instalaciones realizadas con éxito en el equipo.
Requisitos relacionados:	UC-09, SF-11, SF-12, SF-13

Tabla 91: Prueba de integración PN-07

PN-08	
Objetivo:	Comprobar que las operaciones de encendido, reiniciado, apagado, instalación, copiado de máquinas virtuales y modificación de permisos se pueden realizar a la vez en dos equipos de dos aulas distintas.
Necesidades del entorno:	Las asociadas a cada prueba concreta.
Entradas:	Las asociadas a cada prueba concreta.
Secuencia:	Realizar las pruebas PN-02, PN-05, PN-06 y PN-07 en dos equipos de dos aulas distintas. Comprobar que ambos sistemas han pasado las pruebas sin errores.
Salidas:	Las asociadas a cada prueba concreta.
Requisitos relacionados:	UC-10, SF-14

Tabla 92: Prueba de integración PN-08

Evaluación

PN-09	
Objetivo:	Comprobar que la aplicación puede ejecutarse tanto en *Debian* como en *Windows*.
Necesidades del entorno:	Equipos de pruebas encendidos en *Debian* y en *Windows*.
Entradas:	Ninguna.
Secuencia:	Abrir por consola de comandos una conexión remota con el servidor que aloja la aplicación. Ejecutar la aplicación. Comprobar que la aplicación se ejecuta correctamente.
Salidas:	Aplicación abierta sin errores.
Requisitos relacionados:	UR-01, SN-01, SN-02

Tabla 93: Prueba de integración PN-09

PN-10	
Objetivo:	Comprobar que los equipos apagados muestran su información en un tono grisáceo.
Necesidades del entorno:	Equipo de pruebas apagado.
Entradas:	Ninguna.
Secuencia:	Seleccionar el equipo de pruebas. Obtener el estado del equipo. Comprobar que la información aparece en color gris.
Salidas:	Información asociada al equipo en tono grisáceo.
Requisitos relacionados:	UR-04, SN-06

Tabla 94: Prueba de integración PN-10

Evaluación

PN-11	
Objetivo:	Comprobar que el menú desplegable con el botón derecho del ratón se activa al pinchar en un equipo, en un aula o en cualquier otro lugar de la tabla de información de las aulas.
Necesidades del entorno:	Ninguna.
Entradas:	Ninguna.
Secuencia:	Pinchar con el botón derecho sobre un equipo. Comprobar que el menú se despliega con la información correspondiente. Repetir los pasos [2] y [3] con un aula y con cualquier otro lugar de la tabla de información de las aulas.
Salidas:	Menú desplegado.
Requisitos relacionados:	UR-05, SN-07

Tabla 95: Prueba de integración PN-11

PN-12	
Objetivo:	Comprobar que antes de reiniciar un equipo la aplicación muestra una ventana que permite elegir el sistema operativo en el que se iniciará el equipo.
Necesidades del entorno:	Ninguna.
Entradas:	Ninguna.
Secuencia:	Abrir el menú desplegable pinchando con el botón derecho del ratón sobre el equipo de pruebas y seleccionar la opción *Reiniciar*. Comprobar que se abre una ventana que permite seleccionar el sistema operativo en el que se reiniciará el equipo.
Salidas:	Ventana de selección del sistema operativo abierta.
Requisitos relacionados:	UR-06, SN-08

Tabla 96: Prueba de integración PN-12

Evaluación

PN-13	
Objetivo:	Comprobar que al intentar realizar una instalación en un equipo que no se encuentre en *PXE* se muestra un mensaje de error.
Necesidades del entorno:	Equipos de pruebas en *Debian*, *Windows* y apagados.
Entradas:	Al menos una instalación a realizar.
Secuencia:	Seleccionar los equipos de pruebas. Marcar una instalación en la pestaña de instalaciones. Lanzar el proceso de instalación. Comprobar que aparece un mensaje de error.
Salidas:	Mensaje de error indicando los nombres de todos los equipos seleccionados.
Requisitos relacionados:	UR-07, SN-09

Tabla 97: Prueba de integración PN-13

6.3. Matrices de trazabilidad

Las matrices de trazabilidad son una herramienta muy importante para realizar un correcto seguimiento de los requisitos a lo largo del ciclo de desarrollo de un proyecto. A continuación se expone la matriz de trazabilidad que relaciona los requisitos de usuario y las pruebas.

6.3.1. Matriz de trazabilidad entre requisitos y pruebas

En la siguiente tabla se muestra una matriz de trazabilidad entre requisitos de usuario y pruebas:

Evaluación

		REQUISITOS DE USUARIO																	
		UC-01	UC-02	UC-03	UC-04	UC-05	UC-06	UC-07	UC-08	UC-09	UC-10	UC-11	UR-01	UR-02	UR-03	UR-04	UR-05	UR-06	UR-07
PRUEBAS	PU-01	X																	
	PU-02		X																
	PU-03		X																
	PU-04		X																
	PU-05			X															
	PU-06			X															
	PU-07											X							
	PN-01			X															
	PN-02				X														
	PN-03					X													
	PN-04						X												
	PN-05							X											
	PN-06								X										
	PN-07									X									
	PN-08										X								
	PN-09												X						
	PN-10															X			
	PN-11																X		
	PN-12																	X	
	PN-13																		X

Tabla 98: Matriz de trazabilidad entre requisitos y pruebas

7. Implantación

En este apartado se detalla la estrategia de implantación del sistema, incluyendo el proceso de despliegue, instalación y configuración de la aplicación desarrollada. También se incluye una referencia al manual de usuario.

A continuación se identifican las distintas fases de las que se compone el proceso de implantación del sistema:

- **Preparación** de la infraestructura necesaria para la incorporación del sistema al entorno de operación.
- **Instalación** de los componentes asociados al sistema.
- **Carga de datos** al entorno de operación.
- Ejecución del **plan de pruebas** de implantación y aceptación del sistema.
- **Plan de formación** de los usuarios finales de la aplicación (administradores).

Para la correcta puesta a punto del plan de implantación también es necesario identificar a los responsables que se encargarán de velar para que las tareas anteriores se lleven a cabo de forma eficaz. Dicho equipo de trabajo estará formado por los siguientes perfiles:

- **[1] Responsable de implantación**. Será el encargado de verificar que el proceso de implantación se lleva a cabo de forma correcta.
- **[2] Responsable del plan de formación**. Se encargará de planificar y organizar la formación a los usuarios finales de la aplicación.
- **[3] Responsable de la carga de datos**. Será el encargado de crear las estructuras que almacenarán la información de las aulas así como la carga de dichos datos.
- **[4] Responsable de pruebas**. Su tarea consistirá en realizar las pruebas de implantación y aceptación definidas en la especificación técnica del plan de pruebas.

Es conveniente que el responsable de pruebas sea un analista, no obstante el resto de tareas pueden ser desempeñadas por la misma persona siempre que cumpla con los requisitos de formación mínimos para llevarlas a cabo.

Implantación

7.1. Preparación de la infraestructura

El objetivo principal de esta fase es generar un paquete de distribución del proyecto y verificar que la configuración detallada en el apartado *5.1 Configuración del entorno* se encuentra correctamente instalada y preparada para la siguiente fase de la implantación.

Aunque el IDE Anjuta utilizado para el desarrollo de la aplicación es capaz de gestionar proyectos para instalaciones básicas con Makefile, su interfaz está diseñada especialmente para proyectos de **autotools**. Este tipo de proyectos se construyen en tres pasos: **generar**, **configurar** y **construir**[22].

En la fase de **generación** se crea el script *configure*, el cual se utiliza para realizar algunas comprobaciones del sistema antes de instalar un paquete. La fase de **configuración** ejecuta el script *configure*. Durante la fase de **construcción** se compilan y enlazan los archivos objeto del código fuente.

Para crear el paquete de distribución del proyecto se debe seleccionar la opción *Construir tarball* desde el menú *Construir* de Anjuta. Esto genera un fichero con el nombre del proyecto y la extensión *tar.gz* que se utilizará en la siguiente fase para instalar la aplicación en el sistema.

7.2. Instalación de componentes

En el caso de haber generado el paquete de distribución correctamente la instalación de la aplicación se convierte en un proceso muy sencillo. Simplemente hay que descomprimir el paquete, compilarlo e instalarlo utilizando la secuencia de comandos *configure*, *make* y *make install*, común a la instalación de paquetes con la herramienta *autotools*.

7.3. Carga de datos al entorno de operación

Esta tarea será realizada por el responsable de la carga de datos, y sólo podrá llevarse a cabo una vez completadas la fase de preparación de la infraestructura y la fase de instalación de los componentes del sistema.

7.3.1. Procedimiento de carga inicial

Una vez que el responsable de implantación da el visto bueno, el responsable de la carga de datos puede comenzar con el paso previo a la carga: la recopilación de la información relativa a la gestión de las aulas. Para ello deberá entrevistarse con los técnicos del Laboratorio y solicitarles los siguientes datos:

- Número total de aulas*****.
- Lista de aulas*****. Por cada aula:
  - Nombre*****.
  - Dirección.
  - Filas de equipos.
  - Columnas de equipos.
  - Lista de equipos que componen el aula. Por cada equipo:
    - Nombre del equipo*****.
    - Dirección IP del equipo.
  - Lista de instalaciones que se pueden realizar en el aula. Por cada instalación:
    - Nombre de la instalación*****.
    - Lista de parámetros de configuración de la instalación.
- Directorio dónde se almacenarán las máquinas virtuales*****.
- Lista de permisos por defecto*****:
  - Nombre del usuario o grupo al que se le otorgará un permiso.
  - Nombre y valor del permiso (True o False).

NOTA: la información que aparece sin asterisco es opcional y su ausencia no influye en la correcta ejecución de las tareas de administración que se pueden llevar a cabo con Kosmos.

Tras haber obtenido la información enumerada arriba, el responsable realizará la carga de datos en las estructuras definidas en el apartado *4.4 Diseño físico de datos*.

7.4. Ejecución del plan de pruebas de implantación y aceptación

A continuación se presenta la especificación técnica del plan de pruebas de implantación y aceptación, correspondiéndose estas últimas con una única encuesta que permite estimar la satisfacción de los usuarios en relación a la usabilidad de la aplicación.

7.4.1. Pruebas de implantación

A continuación se enumeran las pruebas de implantación, las cuales permiten comprobar si la aplicación se ha instalado de manera correcta:

PM-01	
Objetivo:	Comprobar que se puede acceder desde cualquier equipo de las aulas así como desde los equipos de los administradores al servidor en el que se encuentra alojada la aplicación.
Necesidades del entorno:	Equipos conectados a la misma red.
Entradas:	Nombre o dirección del servidor que aloja la aplicación y nombre de usuario y contraseña del administrador.
Secuencia:	Abrir una terminal en el equipo. Realizar una conexión por *SSH* con el servidor que aloja la aplicación. Comprobar que se pueden ejecutar comandos en la terminal remota.
Salidas:	Salida de los comandos de prueba.

Tabla 99: Prueba de implantación PM-01

PM-02	
Objetivo:	Comprobar que la aplicación carga todos sus componentes sin errores al iniciarse.
Necesidades del entorno:	Ninguna.
Entradas:	Ninguna.
Secuencia:	Abrir la aplicación. Comprobar que no se han producido errores durante la apertura.
Salidas:	Aplicación abierta sin errores.

Tabla 100: Prueba de implantación PM-02

Implantación

PM-03	
Objetivo:	Comprobar que la aplicación puede exportar su interfaz gráfica a cualquier equipo de las aulas, independientemente del sistema operativo que se encuentre iniciado.
Necesidades del entorno:	Equipos de pruebas encendidos en *Debian* y en *Windows*.
Entradas:	Ninguna.
Secuencia:	Abrir una terminal en el equipo. Realizar una conexión por *SSH* con la opción que permite exportar las X (sistema de ventanas) con el servidor que aloja la aplicación. Abrir la aplicación en la terminal remota abierta. Comprobar que la interfaz gráfica de la aplicación se muestra sin errores en el equipo de pruebas.
Salidas:	Aplicación abierta sin errores.

Tabla 101: Prueba de implantación PM-03

7.4.2. Pruebas de aceptación

Tras haber pasado satisfactoriamente las pruebas de implantación del sistema, la aplicación se encontrará mucho más cerca de cumplir los requisitos para el paso a producción. Sin embargo, antes será necesario realizar una última **presentación del proyecto** al cliente con el objetivo de mostrarle el resultado final del sistema y que posteriormente deje constancia de su aprobación. La presentación consistirá en una demostración de las funcionalidades del sistema desarrollado.

Además de esto y tras un **período de prueba de 30 días naturales** que tendrán como objetivo permitir al cliente probar con mayor profundidad la aplicación, se realizará una encuesta que permitirá a los desarrolladores estimar de manera objetiva la satisfacción de los usuarios. Una vez trascurrido este período el cliente deberá firmar un documento que acredite la **aprobación del sistema**.

7.5. Plan de formación

Es de vital importancia garantizar que los usuarios finales posean los conocimientos necesarios para la utilización de la aplicación. Para ello, se impartirán los siguientes cursos a los perfiles definidos al comienzo de este apartado:

- **[1] Curso básico de Python** (20 horas).
- **[2] Curso de programación de interfaces gráficas con GTK+** (20 horas).
- **[3] Curso básico de gestión y almacenamiento de información mediante lenguaje de marcas** (8 horas).
- **[4] Curso básico de administración de equipos con Kosmos** (2 horas).

Para asegurar la correcta implantación del sistema el perfil **[1]** deberá asistir a los cursos **[1]** y **[2]**. A los perfiles como el **[3]** sólo se les obligará a asistir al curso **[3]**, necesario para una apropiada carga de los datos en el sistema. Por otra parte y pese a que algunos de los usuarios finales (técnicos del Laboratorio) ya poseen los conocimientos impartidos en varios de estos cursos, deberán asistir al menos al curso **[4]**, así como a aquellos que consideren oportunos para reforzar sus habilidades en las correspondientes materias.

Será necesario garantizar la asistencia a los cursos solicitando una confirmación que lo acredite y una prueba que demuestre las capacidades adquiridas por el equipo encargado de la implantación. También será conveniente llevar a cabo un seguimiento de la formación impartida a los usuarios finales.

Con el fin de completar la documentación relativa al sistema y facilitar el uso de la aplicación por parte de los usuarios finales se adjunta el siguiente manual:

- Apéndice II: Manual de administración de equipos con Kosmos.

8. Planificación y presupuesto

En este apartado se expone un diagrama con la programación estimada de las fases del proyecto y el cálculo detallado de los costes y beneficios obtenidos.

8.1. Planificación

La programación del proyecto se estimó en base al período de entrega de la memoria de los trabajos de fin de grado, que en el presente año se estableció entre el 15 y el 22 de junio de 2016, y al número de horas correspondientes a cada crédito ECTS. Puesto que el TFG posee un valor de 12 ECTS y cada uno de estos créditos equivale a 25 horas de trabajo del estudiante, se estima una duración de **20 semanas** (con 15 horas de trabajo a la semana). De esta forma, la fecha de inicio del proyecto se fija el **25 de enero de 2016** y la fecha estimada de finalización el **10 de junio de 2016**.

Se ha decidido dividir la planificación del proyecto en las siguientes fases principales:

- **Documentación**.
- **Propuesta de proyecto**.
- **Análisis**:
  - Valoración de las alternativas.
  - Análisis de requisitos de usuario.
  - Análisis de requisitos de software.
- **Diseño**:
  - Diseño de la arquitectura.
  - Diseño de casos de uso.
  - Diseño de clases.
  - Diseño físico de datos.
- **Implementación**.
- **Implantación**:
  - Formación del equipo.
  - Preparación de la infraestructura.
  - Instalación de componentes.
  - Carga de datos.
- **Evaluación**:
  - Plan de pruebas.
  - Satisfacción de usuarios.

La estimación de cada una de estas fases se detalla en el siguiente diagrama de Gantt:

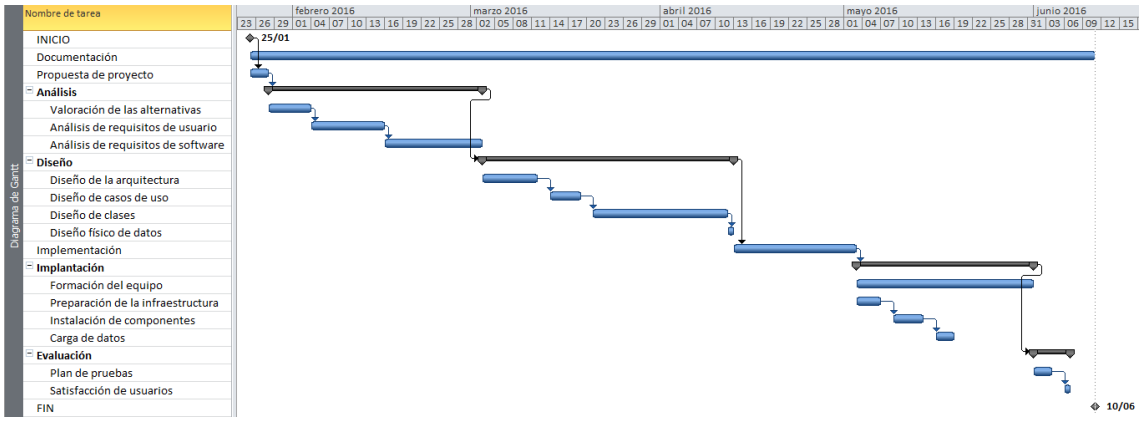


Ilustración 43: Diagrama de Gantt con la planificación estimada del proyecto

8.2. Presupuesto

En base a la planificación anterior y el salario medio de los distintos roles que trabajan en el desarrollo del proyecto se calculan los costes asociados al gasto de personal. Después se realiza el cálculo de costes del material amortizable, los gastos indirectos como el alquiler de un local de trabajo y otros costes relacionados y el monto total del presupuesto.

8.2.1. Coste de personal

Se han identificado dos roles principales que engloban las características básicas para llevar a cabo el desarrollo de las distintas fases de las que se compone el proyecto. A continuación se presenta una tabla que desglosa el coste asociado a los salarios de los trabajadores, realizadas a partir de estadísticas[23] de los salarios medios de estos roles en el mercado actual suponiendo que carecen de experiencia previa en el puesto de trabajo.

Para las cotizaciones correspondientes a la **Seguridad Social**[24] (cuota patronal) el cálculo se lleva a cabo sobre el salario bruto anual de cada trabajador. Las pagas extra no se han incluido en el desglose de costes porque se encuentran prorrateadas entre las distintas mensualidades.

SALARIOS		
	Analista de sistemas	**Programador**
Salario bruto anual	28.020,00 €	22.836,00 €
Salario bruto mensual	2.335,00 €	1.903,00 €

CUOTA PATRONAL		
Seguridad Social (23,6 %)	551,06 €	449,11 €
Desempleo (5,5 %)	128,43 €	104,67 €
Formación profesional (0,6 %)	14,01 €	11,42 €
Fogasa (0,2 %)	4,67 €	3,81 €
Cuota patronal mensual	698,17 €	569,00 €

PLANIFICACIÓN DEL PROYECTO		
Duración del proyecto (horas)	300	300
Duración del proyecto (semanas)	20	20
Horas trabajadas a la semana	15	15
Horas trabajadas al mes	40	40

COSTE DE PERSONAL		
Coste mensual del trabajador	3.033,17 €	2.472,00 €
Coste de la hora trabajada	75,83 €	61,80 €
Coste de personal asociado al trabajador	22.748,74 €	18.539,98 €
Coste de personal		**41.288,72 €**

Tabla 102: Coste de personal

8.2.2. Coste de los equipos

A continuación se detalla el precio de los equipos correspondientes al entorno operacional donde debe desplegarse el proyecto. Para la estimación del precio de estos recursos se han consultado proveedores que venden estos productos a través de Internet. Los precios incluyen el *Impuesto sobre el Valor Añadido (IVA)*.

COSTE DE LOS EQUIPOS		
Concepto	**Modelo**	**Precio**
Servidor Instalador	Sun Microsystems Sun Fire V40z	1.268,00 €
Equipo Minardi	Acer Ferrari One 200-312G25n	399,00 €
Equipo Toleman	Toshiba Portégé Z30 Ultrabook	699,00 €
Coste de los equipos		**2.366,00 €**

Tabla 103: Coste de los equipos

8.2.3. Costes indirectos

Para el cálculo de los costes asociados a los equipos físicos utilizados durante el desarrollo del proyecto se asume que todos ellos tienen una vida útil de 4 años. Para el software se asume una vida útil de 3 años. Por esta razón se ha tomado una tasa de amortización del 25% al año para los equipos (tomando un año de 52 semanas) y del 33% para los programas informáticos. Puesto que la duración del proyecto es de 20 semanas, tan sólo se imputan como gastos al proyecto los costes de amortización correspondientes a dicho período de tiempo. Los precios incluyen el *Impuesto sobre el Valor Añadido (IVA)*.

COSTE DE RECURSOS PARA EL DESARROLLO				
Concepto	**Modelo**	**Cantidad**	**Precio**	**Total**
Ordenadores de sobremesa	Acer AXC-705	2	399,00 €	798,00 €
Monitores	Acer V206HQLAb	4	75,00 €	300,00 €
Impresoras	HP LaserJet Pro M201n	1	179,00 €	179,00 €
Suite ofimática	Microsoft Office Home & Business 2010	2	279,00 €	558,00 €

AMORTIZACIONES				
Concepto	**Amortización anual**	**Amortización semanal**	**Semanas**	**Total**
Ordenadores de sobremesa	199,50 €	3,84 €	20	76,73 €
Monitores	75,00 €	1,44 €	20	28,85 €
Impresoras	44,75 €	0,86 €	20	17,21 €
Suite ofimática	184,14 €	3,54 €	20	70,82 €
Total				**193,61 €**

OTROS COSTES	
Concepto	**Precio**
Alquiler de local (incluye luz y agua)	450,00 €

COSTES INDIRECTOS	
Coste indirectos	**643,61 €**

Tabla 104: Costes indirectos

8.2.4. Precio final

El coste total será la suma de los costes anteriormente desglosados más un margen de riesgo (10%) y otro de beneficio (15%). A dicho monto se le aplica el 21% del *Impuesto sobre el Valor Añadido (IVA)*, resultando en un precio final del proyecto de **SESENTA Y SIETE MIL Y UN EUROS CON VEINTIDOS CÉNTIMOS**.

COSTES DIRECTOS	
Coste de personal	41.288,72 €
Coste de los equipos	2.366,00 €
COSTES INDIRECTOS	
Costes indirectos	643,61 €
SUBTOTAL	
Subtotal	44.298,33 €
MARGEN DE RIESGO (10%)	
Margen de riesgo	4.429,83 €
MARGEN DE BENEFICIO (15%)	
Margen de beneficio	6.644,75 €
COSTE TOTAL DEL PROYECTO	
Coste total	**55.372,91 €**
IVA (21%)	
Impuesto sobre el Valor Añadido	11.628,31 €
PRECIO FINAL DEL PROYECTO	
Precio final	**67.001,22 €**

Tabla 105: Precio final del proyecto

9. Conclusiones y trabajos futuros

En este apartado se exponen algunas consideraciones finales relativas a la elaboración del proyecto y algunas ideas y ampliaciones surgidas durante su desarrollo y que mejorarían las características de la aplicación.

9.1. Conclusiones

9.1.1. Personales

Personalmente me siento muy orgulloso de haber llevado a cabo un proyecto como este, ya que me hizo conocer en mayor profundidad mis propias capacidades a la hora de organizarme y trabajar de forma autónoma. A su vez, potenció mis aptitudes y me hizo comprender el valor que tiene la responsabilidad ante un proyecto. De hecho, una de las conclusiones que más he apreciado ha sido descubrir lo valiosa que es la constancia y, para mantener esa perseverancia durante todo el desarrollo del proyecto, establecer metas a corto, medio y largo plazo que uno sea capaz de cumplir.

Por otra parte me produce una gran satisfacción que el proyecto pueda llegar a utilizarse en el Laboratorio del Departamento de Informática. Sería una forma de agradecer a los técnicos todo lo que me han enseñado durante los dos años que he pasado trabajando con ellos como becario.

Sin duda gran parte de la formación que me ha permitido llevar a cabo el proyecto la he recibido durante la carrera universitaria, y aunque todas las asignaturas me han enseñado algo, los conceptos que más he utilizado los he adquirido en las siguientes:

- **Criptografía y Seguridad Informática**. Esencial para comprender algunos conceptos de seguridad relacionados con protocolos como *SSH*, el cual utiliza criptografía de clave pública para la autenticación de los equipos.
- **Programación**. Puesto que el proyecto se basa en el desarrollo de una aplicación informática, he necesitado aplicar los conocimientos relativos a la programación orientada a objetos que me enseñaron en esta asignatura.
- **Estructura de Datos y Algoritmos**. Determinadas estructuras de datos en las que se almacena la información de las aulas y los equipos son listas y diccionarios de pares clave-valor, cuyo tratamiento se relaciona con los algoritmos y estructuras que se estudian en esta asignatura.
- **Redes de Ordenadores**. Vital para la implementación de las tareas que se pueden llevar a cabo con Kosmos, ya que es necesario una base de conocimientos sobre configuración de redes y protocolos.

- **Sistemas Operativos**, **Diseño de Sistemas Operativos**, **Arquitectura de Computadores**, **Sistemas Distribuidos**. Sin los conceptos relacionados con la concurrencia y sincronización de procesos que me enseñaron en estas asignaturas no podría haber desarrollado una aplicación tan potente, robusta y rápida como Kosmos.
- **Interfaces de Usuario**. Fue necesaria a la hora de diseñar la interfaz de la aplicación, pues se aplicaron algunos patrones de diseño y ciertas heurísticas para elaborar la maqueta inicial.
- **Ingeniería del Software** y **Dirección de Proyectos de Desarrollo de Software**. Todo el proceso de documentación incluido en la presente memoria se estudia en estas dos asignaturas.

Además de estos conocimientos también he aprendido por cuenta propia algunas otras cuestiones específicas como programar en Python, el desarrollo de interfaces con bibliotecas gráficas como GTK+, aspectos relacionados con la instalación y configuración de servidores *Linux* o conceptos de redes en mayor profundidad como el estándar *Wake-on-LAN* o el protocolo *SSH*.

9.1.2. Del producto

La herramienta desarrollada en este proyecto ha cumplido satisfactoriamente los objetivos para los que fue diseñada, tal y como se plantearon en la introducción de este documento:

- La aplicación permite encender, reiniciar y apagar de forma remota los equipos de las aulas. También permite la instalación remota de imágenes de *Windows* y *Debian*.
- La aplicación permite transferir máquinas virtuales a los equipos y modificar los permisos de un fichero o directorio de *Windows*. Esto incluye los ficheros y directorios de las máquinas virtuales, que es una de las tareas para la cual se diseñó la aplicación.
- La aplicación permite llevar a cabo tareas en diferentes equipos de distintas aulas al mismo tiempo mediante la implementación de dichas tareas con *threads*.
- La aplicación posee una interfaz gráfica de usuario intuitiva y usable que facilita la realización de todas las tareas citadas anteriormente.
- La aplicación puede ejecutarse en ordenadores con *Windows* y *Debian*, ya que sólo es necesario conectarse al servidor que aloja la aplicación exportando el sistema de ventanas para ejecutarla.

9.1.3. Del proceso

En cuanto a las conclusiones relativas al proceso de desarrollo del proyecto, los plazos establecidos para la finalización del proyecto se han cumplido satisfactoriamente. De hecho, la programación estimada en 20 semanas se habría podido reducir hasta un mínimo de 18 semanas sin necesidad de ampliar el número de trabajadores o suponer una carga de trabajo excesiva para ellos. Además esto habría derivado en una reducción del precio final de la aplicación de hasta un 9,32%, puesto que el coste de personal habría sido más bajo al disminuir el número de horas de cada trabajador dedicadas al proyecto.

9.2. Trabajos futuros

Este apartado incluye algunas ideas y ampliaciones surgidas durante el desarrollo del proyecto y que mejorarían las características de la aplicación. La mayoría de ellas no pudieron implementarse por falta de tiempo o porque su desarrollo habría supuesto una tarea tan amplia como otro trabajo de fin de grado.

Seguridad

Uno de los aspectos que podrían mejorarse tiene que ver con la seguridad, ya que al carecer de un soporte oficial continuado como el que pueda ofrecer una gran empresa, también carece de actualizaciones periódicas que puedan corregir errores de seguridad no detectados durante su desarrollo. Además, la actualización de las librerías que utiliza la aplicación podría derivar en otros errores imprevistos al carecer también de un equipo de mantenimiento que corrija los problemas que puedan surgir durante dicho proceso.

Edición de aulas

La versión final de la aplicación obtiene la información de las aulas sobre las que se pueden aplicar las tareas de un fichero *XML*, el cual almacena las características de cada aula junto al número y nombre de los equipos que la componen. Sin embargo, pese a que este diseño es bastante flexible ya que permite añadir o modificar la configuración de las aulas en cualquier momento, es una solución poco usable.

Esto podría solventarse añadiendo una característica que permitiese la edición de las aulas a través de la interfaz. Por ejemplo, una ventana de edición que ofreciese distintas opciones como añadir nuevas aulas o nuevos equipos a las aulas que ya existen, modificar los nombres de éstos o las instalaciones que pueden llevarse a cabo en cada aula, o cambiar la distribución de los mapas de las aulas. La interfaz del editor seguiría obteniendo de la misma forma la información del fichero *XML*, con la única diferencia de que al modificar dicha información en

la interfaz tendría que estructurarla y sobrescribirla otra vez de forma automática en el fichero.

La razón principal por la que no se implementó esta mejora fue porque su desarrollo habría llevado una cantidad de tiempo del que desgraciadamente se carecía.

Mejoras en la usabilidad

También existen otras soluciones que pueden mejorar la usabilidad de la aplicación. Al igual que la anterior idea, su desarrollo no se llevó a cabo por falta de tiempo.

Entre algunas de estas mejoras se encuentra, por ejemplo, la introducción de indicadores de progreso para las tareas. Esto permitiría a los usuarios observar en tiempo real la evolución de cada tarea en cada equipo o en cada aula. En el caso de los equipos, podrían insertarse marcas de control distribuidas en los scripts de las tareas que permitiesen controlar el porcentaje de progreso de éstas. Por otra parte, el progreso de las tareas en un aula concreta podría calcularse mediante el cómputo de la media de las tareas que se estén realizando en los equipos de dicha aula.

Otra solución que mejoraría la usabilidad de la aplicación pasaría por incluir pequeños iconos junto al nombre de los equipos que permitiesen observar con un vistazo rápido el sistema operativo en el que se encuentran.

Tareas configurables

Otra mejora que no se pudo implementar por falta de tiempo fue la capacidad de configurar las tareas o instalaciones a través de la interfaz. Es una cuestión similar al editor de aulas, de tal forma que en este caso permitiría modificar desde la propia aplicación algunos parámetros relativos a la velocidad de transferencia de datos, tiempos máximos y mínimos de espera, nombre de la interfaz de red o el puerto a través del cual se realiza la instalación, entre otros.

También podrían incluirse mejoras adicionales a la tarea de copia de máquinas virtuales, como la aplicación de los permisos por defecto para los ficheros y directorios asociados a cada máquina transferida sin necesidad de modificarlos después manualmente.

En cierto modo, la ampliación del grado de configuración de todas estas tareas podría reducir las limitaciones en caso de desarrollar una aplicación más genérica que permitiese comercializar una versión para colegios, institutos y otras instituciones académicas.

Monitorización de rendimiento

Una de las mejoras más complejas que surgieron durante el desarrollo del proyecto fue la posibilidad de implementar un sistema de monitorización periódica para observar en cada momento la carga de los equipos de las aulas. En un principio se barajó la opción de utilizar la herramienta **Munin**, no obstante se decidió finalmente descartar la idea ya que llevarla a cabo habría supuesto un trabajo comparable al de otro trabajo de fin de grado.

En caso de implementarse, el proyecto podría incluso aplicarse en otros ámbitos no relacionados con el académico, sustituyendo los equipos de las aulas por servidores e incluyendo algunas de las otras mejoras que se exponen en este apartado.

Análisis de datos

Una mejora muy útil sería el desarrollo de un sistema de registro de logs así como de cualquier error que pueda producirse durante la aplicación de una tarea o instalación, en lugar de mostrarlos en la terminal del servidor desde la que se ha ejecutado la aplicación. El control de las salidas de los comandos podría utilizarse posteriormente para analizar los datos de manera automática o para su redirección a otra aplicación externa que se encargue de ello.

También podría implementarse un sistema que generase informes mensuales o cuatrimestrales con estadísticas acerca de cuántos ordenadores se han instalado o sobre el uso de los equipos, además de otros datos que puedan resultar valiosos para la administración y gestión del Laboratorio del Departamento de Informática. Sin embargo, al igual que el resto de mejoras, no se llevó a cabo por falta de tiempo.

Actualización remota de sistemas

Asimismo sería muy útil añadir una tarea adicional que permitiese la actualización de los equipos de forma remota. De esta forma podría incluirse la posibilidad de llevar a cabo actualizaciones de paquetes o programas concretos con una llamada al comando **apt-get** de *Linux*, o con la herramienta **chocolatey** en *Windows*. También podría implementarse la actualización remota de los sistemas operativos, pero todas estas mejoras suponen un trabajo arduo que no pudo llevarse a cabo por carecer del tiempo necesario.

Accesibilidad

La accesibilidad es vital para que todos los usuarios puedan utilizar un medio en condiciones de igualdad, independientemente de si poseen algún tipo de discapacidad. Por esta razón es muy importante evaluar si la aplicación cumple con todas las recomendaciones y estándares de accesibilidad, y en caso contrario adaptar el sistema para que lo haga. Esta mejora tampoco pudo llevarse a cabo por falta de tiempo.

Aplicación móvil

Otra idea bastante más elaborada pasaría por desarrollar una versión de la aplicación para sistemas Android o iOS, de tal forma que se pudiese comprobar la ocupación de las aulas desde un dispositivo móvil (sin necesidad de conectarse por consola al servidor dónde se aloja Kosmos). Puesto que la conexión podría llevarse a cabo desde cualquier lugar, esta mejora tendría que poner mucha atención a los mecanismos de seguridad que implemente.

Apéndice I: Descripción del prototipo inicial

Este apéndice incluye una descripción detallada de la maqueta básica que sirvió de base para la programación de la aplicación final. El objetivo principal fue facilitar la identificación de nuevos requisitos no contemplados previamente y corregir aquellos que presentaban ambigüedades en su definición

Cabe destacar que el diseño de la interfaz de este primer prototipo se inspira en algunos patrones y heurísticas de diseño de interfaces, como es el de la colocación de las tareas más utilizadas en la parte superior izquierda, la cual se corresponde con el lugar en el que los usuarios suelen posar inicialmente su vista.

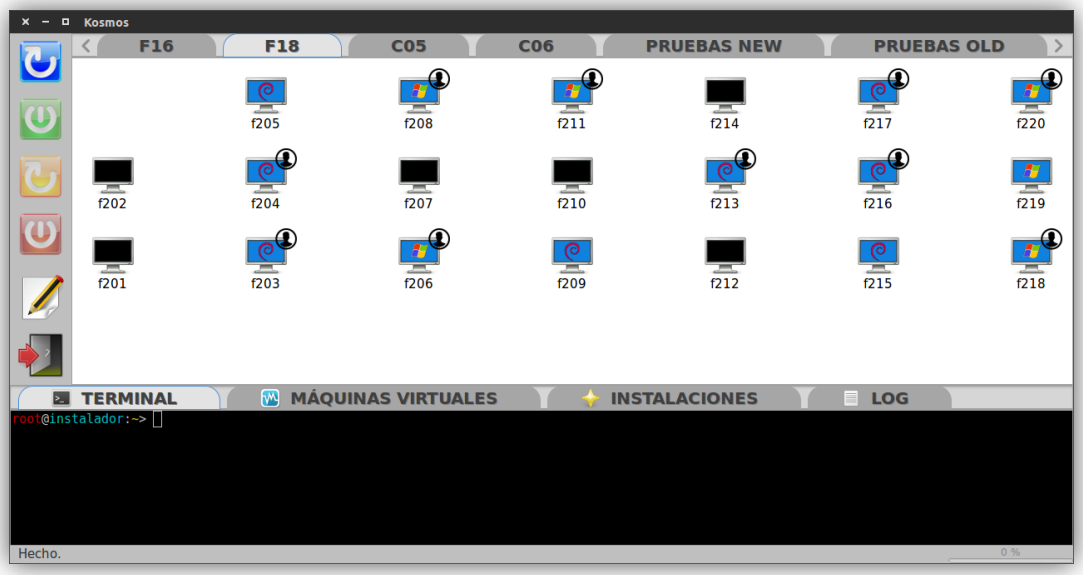


Ilustración 44: Interfaz del prototipo

La descripción del prototipo se ha dividido en las tres partes en que se estructura la aplicación; a saber: **mapas de estado de las aulas**, **cuadro de acciones básicas** y **utilidades de gestión avanzada**.

Mapas de estado de las aulas

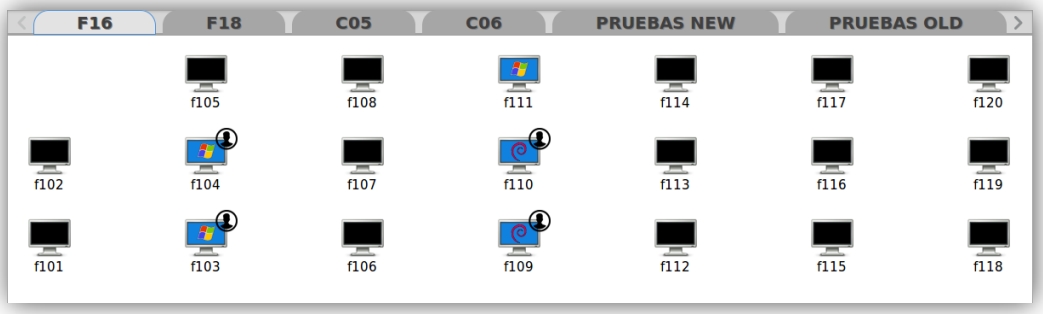


Ilustración 45: Mapa de estado de las aulas del prototipo

Los **mapas de estado de las aulas** ofrecen información sobre el estado y la ocupación de los equipos que conforman las aulas del Laboratorio. Estos mapas ocupan la mitad de la interfaz de la aplicación, aunque su tamaño puede modificarse arrastrando su borde inferior hacia arriba o hacia abajo. Se almacenan en memoria durante la carga inicial de la aplicación, obteniendo todos los datos relativos de un archivo en formato *XML*.

La información almacenada en dicho archivo contiene el nombre de las aulas, el número de filas y columnas de los equipos que las forman así como los huecos que hay entre estas, el nombre de los equipos y las instalaciones que se pueden realizar en un aula determinada. Durante la carga inicial de la aplicación, se genera una pestaña por cada aula y se dibujan los equipos y sus respectivos nombres en la interfaz de la aplicación en función de estos datos.

Estos mapas de estado permiten conocer el estado y la ocupación de los equipos desde la última actualización de dicha información (ver Botón de comprobación en el cuadro de acciones básicas). El estado se muestra en forma de dibujos de ordenadores apagados o encendidos, con el símbolo del sistema operativo sobre la pantalla según el caso. Para mostrar la ocupación, suponiendo que algún usuario haya iniciado sesión en un equipo, se coloca la silueta de una persona sobre la esquina superior derecha del ordenador en cuestión.

Cuadro de acciones básicas

El **cuadro de acciones básicas** permite llevar a cabo las tareas de gestión más simples y comunes. Se sitúa en la parte izquierda de la interfaz, junto a los mapas de estado de aulas. Consta de las siguientes herramientas:

Botón de comprobación

Presenta dos funcionalidades dependiendo de si se presiona con ningún equipo seleccionado o con uno o más equipos seleccionados. En el primer caso (ningún equipo seleccionado) comprobará el estado y la ocupación de todos los equipos del aula sobre la que se encuentre el usuario. En el segundo caso (uno o más equipos seleccionados) sólo actualizará la información de los equipos seleccionados.

Una característica a destacar es que en el prototipo no es posible realizar ninguna otra tarea durante la comprobación del estado de un aula o de un equipo, ya que esta acción es bloqueante y detiene toda la aplicación hasta que recibe la información de los equipos comprobados.

Botón de encendido

Este botón permite encender los ordenadores que se encuentren seleccionados en los mapas de estado de aulas. Puesto que no se implementó ningún método que ofreciese la posibilidad de actualizar el estado de un equipo de forma automática cuando éste terminase de encenderse, se decidió mostrar en la pantalla del equipo un signo de interrogación que obligase al usuario a comprobar el estado del equipo para averiguarlo.

Botón de reinicio

Permite reiniciar en un sistema operativo concreto los equipos que se encuentren seleccionados en los mapas de estado. La elección del sistema operativo se lleva a cabo al presionar el botón, momento tras el cual aparece un mensaje preguntando si se desea reiniciar los equipos en *Debian*, *Windows* o *PXE*.

Botón de apagado

Este botón permite apagar de forma remota los equipos seleccionados. Al igual que con el botón de encendido, se decidió establecer el estado del equipo a *desconocido* tras pulsarlo (dibujando un signo de interrogación sobre la pantalla del ordenador) con el fin de garantizar que al apagarlo el usuario se asegure posteriormente de que esta acción se ha ejecutado con éxito volviendo a comprobar el estado del equipo.

Botón de edición de mapas

Ofrece la posibilidad de editar la información de las aulas a través de la interfaz, en lugar de trabajar directamente con el archivo *XML* (con el riesgo que supone cometer algún error de sintaxis y alterar el correcto funcionamiento de la aplicación). La idea original era poder añadir nuevas aulas, borrarlas o modificar las ya existentes creando nuevos equipos o eliminándolos, o alterando la distribución de éstos en el aula.

Esta característica no se encontraba implementada en el prototipo, ya que fue una opción que se le propuso al cliente como mejora adicional y que este finalmente rechazó.

Botón de salir

Permite cerrar la aplicación de forma controlada. Como mejora adicional, durante el desarrollo del prototipo se decidió implementar un mensaje que advirtiera del peligro que supone salir de la aplicación en el caso de que hubiese instalaciones u otros procesos críticos en ejecución.

Utilidades de gestión avanzada

Las **utilidades de gestión avanzada** permiten llevar a cabo tareas más complejas y que se realizan con menos frecuencia en el Laboratorio. Estas utilidades están compuestas por una terminal integrada en la propia interfaz (que permite lanzar comandos en el servidor donde se ejecuta la aplicación), una herramienta que permite copiar máquinas virtuales específicas a cualquier ordenador de las aulas, otra herramienta que ofrece la posibilidad de realizar algunas instalaciones comunes en los equipos y una pestaña adicional que registra los errores y salidas de todas estas tareas. Estas utilidades apenas presentan cambios con respecto a la versión final de la aplicación.

Terminal integrada

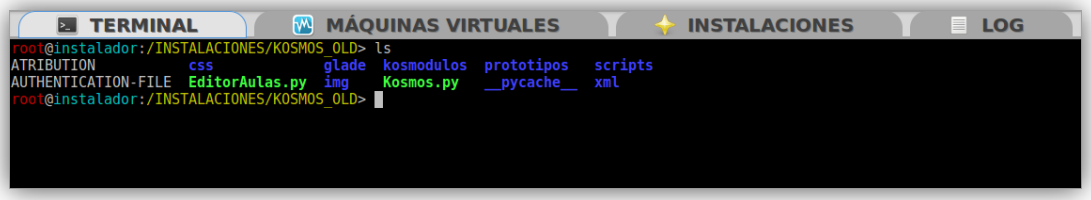


Ilustración 46: Terminal integrada en la interfaz del prototipo

Se decidió implementar esta utilidad con el objetivo de facilitar el acceso al servidor donde se ejecuta la aplicación sin necesidad de cambiar continuamente de ventana para lanzar los comandos. Esta terminal integrada es idéntica a la terminal *bash* de *Linux*, puesto que en realidad es un puente que ofrece el módulo *Vte* de *Python* para ejecutar comandos de sistema.

El cliente tuvo la oportunidad de probar esta terminal ya que se implementó en el prototipo inicial.

Máquinas virtuales

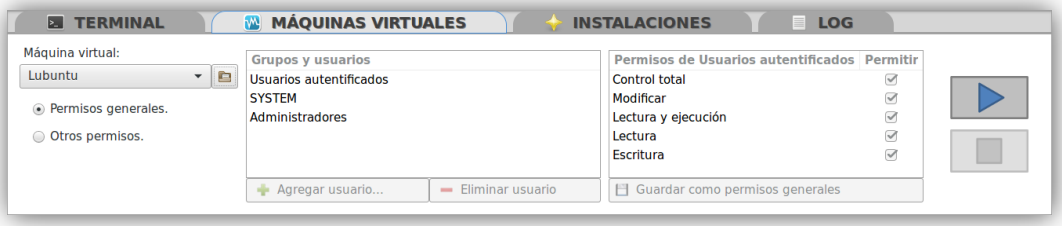


Ilustración 47: Utilidad de copia de máquinas virtuales del prototipo

Esta utilidad permite copiar una máquina virtual concreta a cualquier ordenador de las aulas. Debido a que el cambio de permisos de estas máquinas virtuales es una tarea muy común (para darle permisos de escritura a los ficheros y directorios de la máquina y poder ejecutarla), se decidió agregar una opción adicional que permitiese seleccionar los usuarios y los permisos que se les otorgarían tras el copiado de la máquina virtual. Como se descubrirá en el siguiente apartado, esta característica sufrió algunas modificaciones en la versión final para hacerla más útil.

Cabe destacar que la funcionalidad de esta utilidad no se encontraba implementada originalmente en el prototipo.

Instalaciones

Ilustración 48: Utilidad de instalaciones del prototipo

Proporciona la posibilidad de realizar diversas instalaciones habituales en el proceso de puesta a punto de un equipo. De hecho, la selección de todas las opciones que ofrece esta utilidad permite instalar un equipo nuevo desde cero. Estas opciones son:

- Crear particiones en el disco duro.
- Establecer la configuración de arranque de los equipos.
- Copiar todas las máquinas virtuales que se utilizan en las aulas.
- Copiar la imagen de sistema operativo *Debian*.
- Copiar la imagen de sistema operativo *Windows*.

Aunque presentaba algunos errores debido a su complejidad, esta utilidad se encontraba parcialmente implementada en el prototipo.

Registro de tareas

Registra la salida de los comandos (así como los errores que se producen) durante la ejecución de las dos utilidades anteriores. Esta utilidad sí estaba implementada en el prototipo, aunque presentaba algunos errores y no se pudo verificar su correcto funcionamiento con el copiado de máquinas virtuales puesto que, como ya se ha comentado, no se llegó a implementar dicha herramienta.

Apéndice II: Manual de administración de equipos con Kosmos

Este manual está dirigido a los usuarios finales de la aplicación, con el objetivo de que les sirva de guía de aprendizaje sobre la administración de equipos con la herramienta Kosmos.

Ilustración 49: Logo de la aplicación

INTERFAZ

La interfaz de Kosmos se divide en dos partes bien diferenciadas. En la mitad superior se encuentra la **tabla de información de las aulas** y el **mapa de distribución de cada aula**, mientras que en la mitad inferior de la interfaz se encuentra el **grupo de utilidades avanzadas**, el cual se organiza en cuatro pestañas individuales.

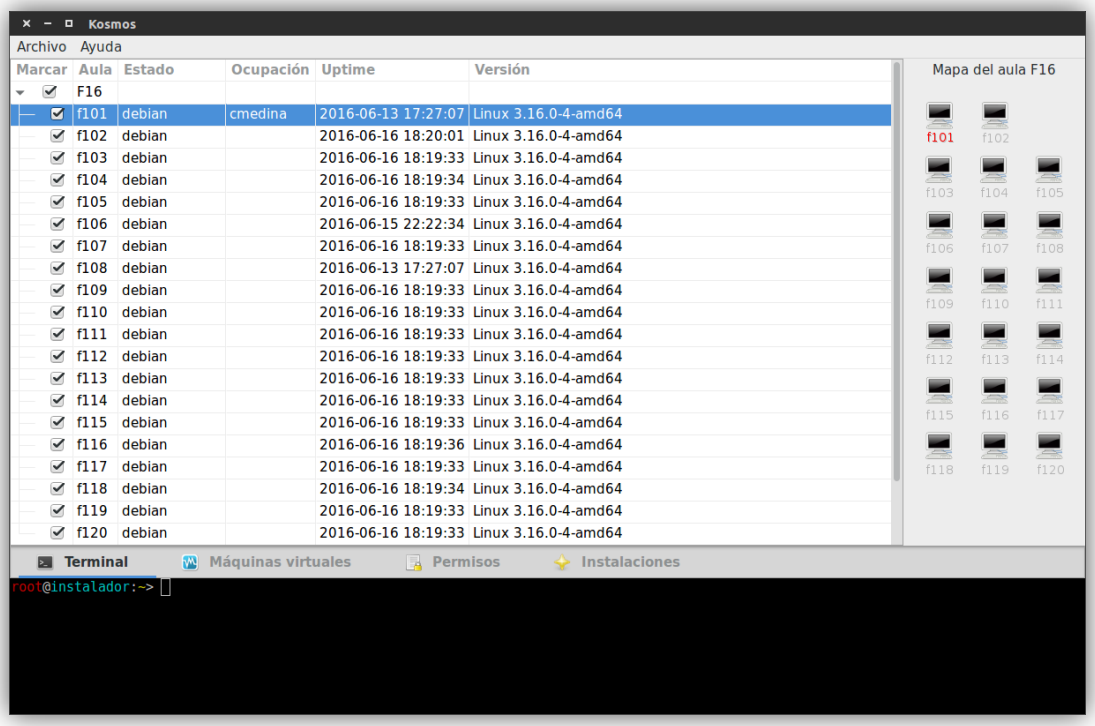
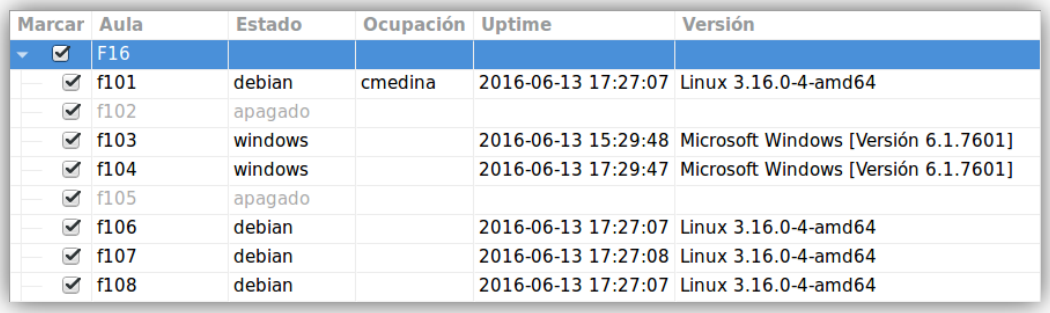


Ilustración 50: Ventana principal de Kosmos

La **tabla de información de las aulas** se encuentra organizada en varias columnas que muestran el estado de los equipos desde la última actualización. Para acceder a esta información se debe **desplegar la lista de equipos** pinchando con el ratón en la pequeña flecha que hay junto al nombre de cada una de las aulas.



Marcar	Aula	Estado	Ocupación	Uptime	Versión
☑	F16				
☑	f101	debian	cmedina	2016-06-13 17:27:07	Linux 3.16.0-4-amd64
☑	f102	apagado			
☑	f103	windows		2016-06-13 15:29:48	Microsoft Windows [Versión 6.1.7601]
☑	f104	windows		2016-06-13 17:29:47	Microsoft Windows [Versión 6.1.7601]
☑	f105	apagado			
☑	f106	debian		2016-06-13 17:27:07	Linux 3.16.0-4-amd64
☑	f107	debian		2016-06-13 17:27:08	Linux 3.16.0-4-amd64
☑	f108	debian		2016-06-13 17:27:07	Linux 3.16.0-4-amd64

Ilustración 51: Tabla de información de las aulas

En la primera columna (**ESTADO**) se muestra si el equipo se encuentra apagado, en *Debian*, en *Windows* o en *PXE*. La segunda columna (**OCUPACIÓN**) indica el nombre del usuario que ha iniciado sesión en el equipo. La tercera y cuarta columnas (**UPTIME** y **VERSIÓN**) muestran información un poco más específica, correspondiente a la fecha y hora en que se encendió el equipo y la versión del *kernel* del sistema operativo que está ejecutando, respectivamente.

Ilustración 52: Mapa de distribución de un aula

Por otra parte, el **mapa de distribución del aula** muestra la posición de cada equipo en un aula. Para descubrir qué posición ocupa un equipo concreto sólo tendremos que pinchar sobre él en la tabla de información de las aulas, para que su nombre se marque en color rojo.

El **grupo de utilidades avanzadas** se organiza en cuatro pestañas, las cuales se corresponden con una terminal integrada y tres tareas avanzadas que se estudiarán con mayor detalle más adelante. La terminal integrada permite ejecutar comandos en el servidor como si fuese una consola de comandos cualquiera.

Ahora que sabemos cómo se organiza la interfaz de la aplicación, podemos aprender cómo se realizan las diferentes tareas de administración con Kosmos. Las tareas de administración se agrupan en dos grupos principales: **tareas básicas** y **tareas avanzadas**.

TAREAS BÁSICAS

Las tareas de administración básicas son seleccionar un equipo, comprobar su estado, encenderlo, reiniciarlo, apagarlo o lanzar una terminal remota. Sin embargo, antes de realizar cualquier tarea en Kosmos tenemos que seleccionar los equipos sobre los que se llevará a cabo.

Hay dos formas de **seleccionar equipos** en Kosmos. Una es marcando la casilla de verificación que hay junto al nombre del equipo (o marcando el aula si queremos aplicar una tarea a un aula completa). La otra forma de seleccionar un equipo es a través del menú desplegable de tareas básicas que pasaremos a explicar ahora mismo.

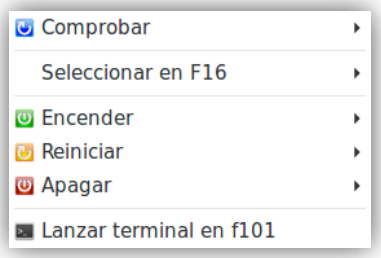


Ilustración 53: Menú de tareas básicas

El **menú de tareas básicas** se despliega pinchando con el botón derecho del ratón sobre cualquier lugar de la tabla de información de las aulas. Este menú cambia en función de si se pincha sobre un equipo o sobre un aula, ofreciendo la opción de aplicar cada tarea solamente en el equipo o aula en el que se ha pinchado o sobre los equipos que se encuentren seleccionados.

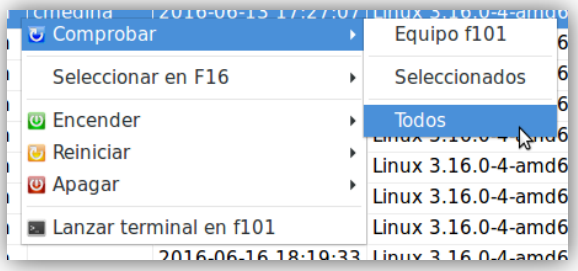


Ilustración 54: Distintas opciones donde aplicar una tarea básica

La mayoría de las tareas básicas se aplican de la misma forma. Primero, se seleccionan los equipos sobre los que se aplicará la tarea. Como se comentó anteriormente, desde el menú desplegable existe una opción que permite **seleccionar equipos** en función de determinadas condiciones. Estas condiciones son el sistema operativo que estén ejecutando o el estado en el que se encuentren (encendido o apagado). También se pueden seleccionar todos los equipos o ninguno.

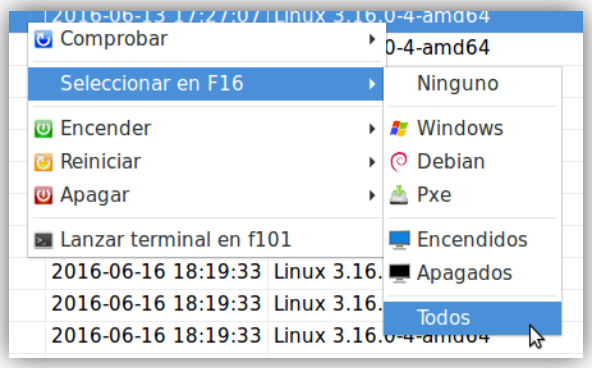


Ilustración 55: Condiciones de selección de equipos

Después hay que activar la tarea, a saber: **comprobar**, **encender**, **reiniciar**, **apagar** o **lanzar terminal**. En algunos casos, como encender, reiniciar o apagar (tareas un poco más delicadas) aplicar la acción necesitará una confirmación adicional del usuario. Para la tarea de reiniciar además deberá escoger el sistema operativo en el que se reiniciará el equipo.

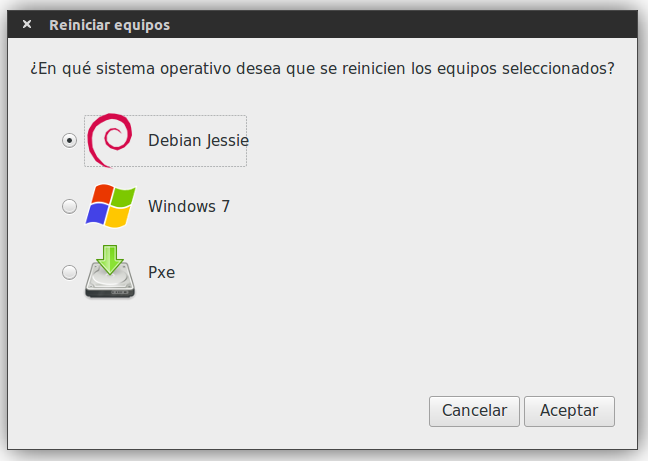


Ilustración 56: Selección del sistema operativo

La tarea de lanzar una terminal abrirá una ventana independiente con una conexión remota a la terminal local, la cual permite ejecutar comandos en el equipo.

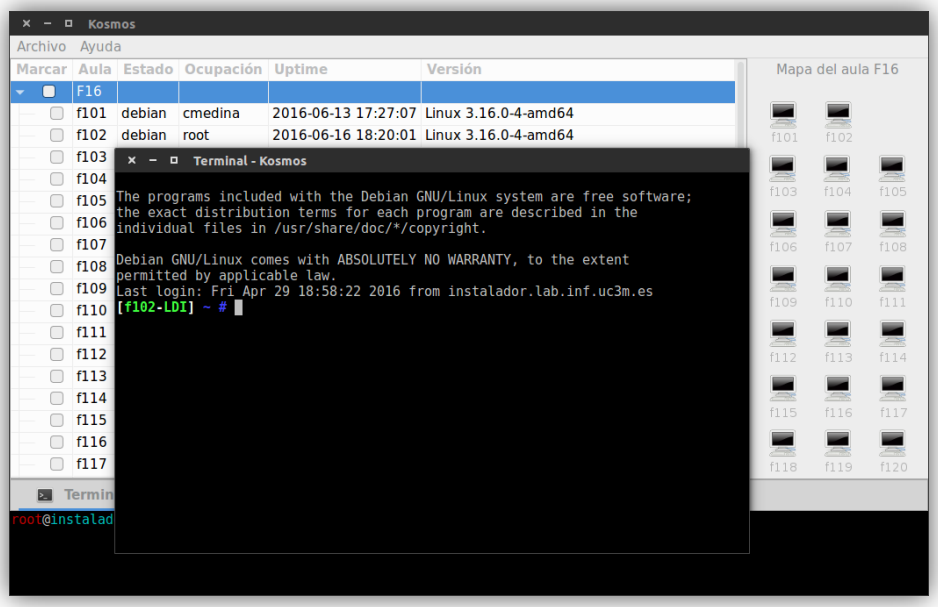


Ilustración 57: Conexión remota con la terminal de un equipo

TAREAS AVANZADAS

Las tareas de administración avanzadas son copiar una máquina virtual, cambiar los permisos de un directorio en Windows y realizar una instalación en un equipo. Además, se incluye una **terminal integrada** que permite ejecutar comandos en el servidor donde se ejecuta la aplicación.

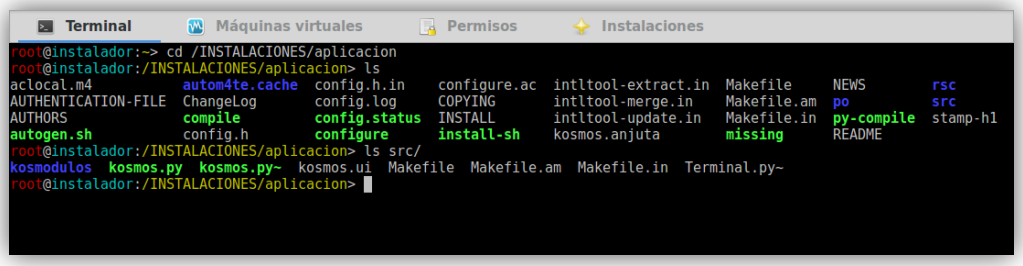


Ilustración 58: Terminal integrada

Al igual que con las tareas básicas el usuario tendrá que seleccionar previamente los equipos sobre los que quiere aplicar una tarea determinada. Para **copiar una máquina virtual**, primero tendrá que asegurarse de que el equipo se encuentra encendido en PXE. Tras esto, podrá seleccionar la máquina que quiere transferir en el desplegable correspondiente y aplicarla pulsando el botón de PLAY. También se incluye un botón junto al desplegable que permite seleccionar el directorio del que se obtienen las máquinas virtuales.

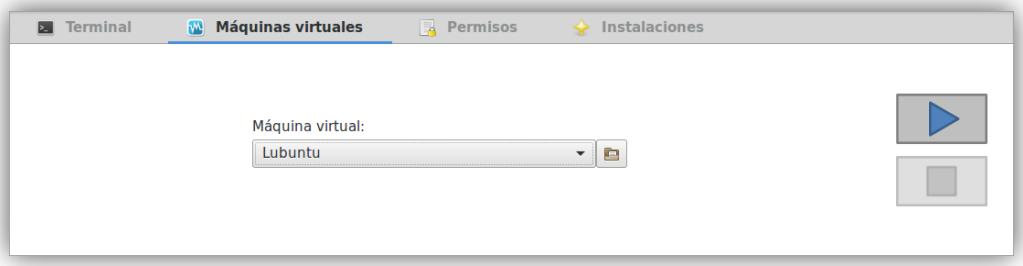


Ilustración 59: Copia de máquinas virtuales

En la pestaña correspondiente al **cambio de permisos** se puede escoger entre dos opciones principales: la de aplicar los permisos guardados como **permisos por defecto (permisos generales)** o aplicar una **configuración distinta (otros permisos)** de dichos permisos. La ruta del objeto hace referencia a la ruta absoluta del fichero o directorio en el equipo local al que se le modificarán los permisos. El usuario deberá asegurarse de que el equipo se encuentra encendido en Windows, de lo contrario no podrá llevarse a cabo el cambio de permisos.

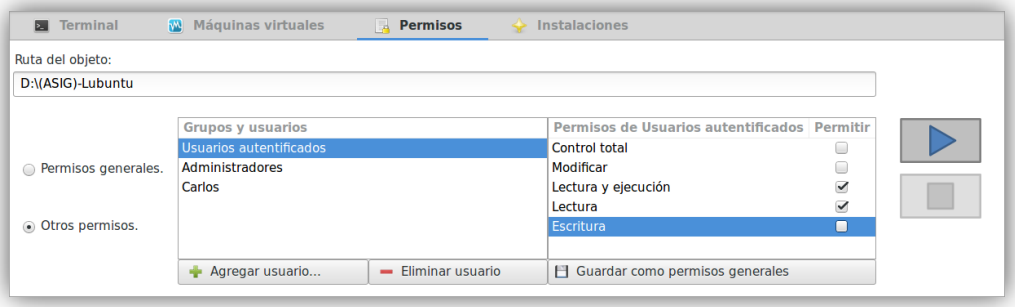


Ilustración 60: Cambio de permisos

La última pestaña se corresponde con la tarea de **instalaciones**. Las instalaciones que se pueden llevar a cabo en los equipos son: *particionar* (crear particiones en) el disco duro, establecer la configuración de arranque de los equipos, copiar todas las máquinas virtuales que se utilizan en las aulas, copiar la imagen del sistema operativo *Debian* o la del sistema operativo *Windows*. Al igual que con la tarea de máquinas virtuales, para aplicar esta tarea los equipos seleccionados deberán encontrarse en *PXE*.

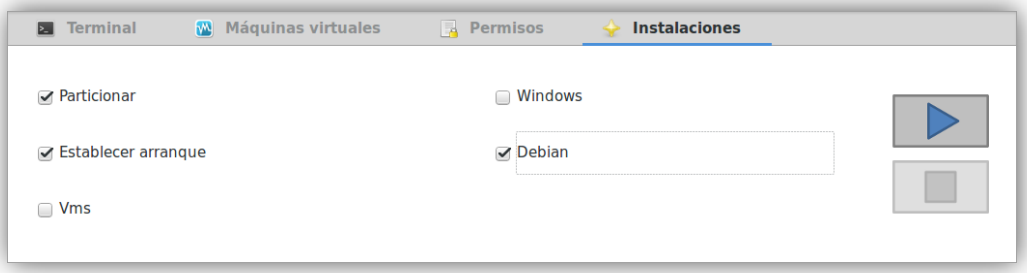


Ilustración 61: Instalaciones

AÑADIR NUEVOS EQUIPO O AULAS

Aunque es una tarea que se realiza con poca frecuencia en el Laboratorio, resulta conveniente saber cómo se pueden agregar nuevos equipos a un aula o una nueva aula.

La información de las aulas se almacena en un fichero en ***rsc/xml/aulas.xml***. Para agregar nuevos equipo o nuevas aulas, por tanto, tendremos que abrir dicho archivo con un editor de texto y modificar su contenido, teniendo en cuenta que los datos que introduzcamos tendrán que ir correctamente formateados con las etiquetas correspondientes.

Para **añadir un equipo** sólo tendremos que incluir, entre las etiquetas *<equipos></equipos>*, una etiqueta como la que se muestra en la imagen.

```
<equipos>
    <equipo ip="">p0</equipo>
    <equipo ip="">p1</equipo>
    <equipo ip="">p2</equipo>
</equipos>
```

Ilustración 62: Etiquetas necesarias para añadir un equipo nuevo

Si lo que queremos es **añadir un aula** nueva tendremos que incluir toda la información que se muestra en la siguiente imagen:

```
<aulas>
    <aula>
        <nombre>AULA VACIA</nombre>
        <direccion>0.0.0.0</direccion>
        <filas>2</filas>
        <columnas>3</columnas>
        <equipos>
            <equipo ip="">p0</equipo>
            <equipo ip="">p1</equipo>
            <equipo ip="">p2</equipo>
        </equipos>
        <instalaciones>
            <instalacion nombre="Particionar" tipo="efi"/>
            <instalacion nombre="Establecer arranque" tipo="efi"/>
            <instalacion nombre="Vms" origen="/INSTALACIONES/imagenes/vms-2015-2.gz" destino="sda3"/>
            <instalacion nombre="Windows" origen="/INSTALACIONES/imagenes/windows/sda2-intel-2015-cuentas-iscsi.gz" destino="sda2"
            <instalacion nombre="Debian" origen="/INSTALACIONES/imagenes/jessie.gz" destino="sda5"/>
        </instalaciones>
    </aula>
</aulas>
```

Ilustración 63: Etiquetas para añadir un aula nueva

Apéndice III: Resumen en Inglés

1. Introduction

This section provides a general approach to the project, detailing the context of its development and the goals to be achieved. This section also describes the structure chosen to organize the document.

1.1. Motivation

During the two years that I have been a trainee in the **Computer Science and Engineering Department at the Carlos III University of Madrid**, I learned to use many of the tools of its Laboratory. This is why I know the limitations of these utilities.

Most limitations are related to a **poor usability** of the tools, because many of these management programs are a simple group of scripts whose use (remember the specific command, parameters, etc.) becomes progressively more difficult as their number increases.

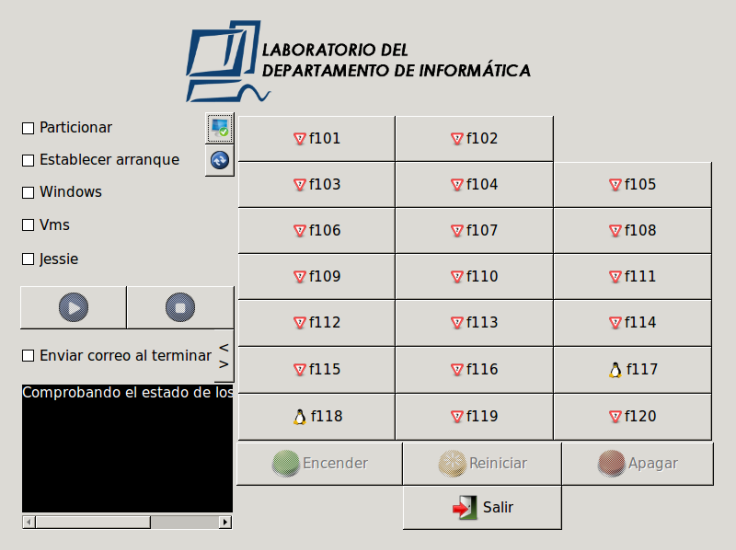


Figure 64: Preceding software of computer management

A few years ago a tool that tried to group together some of the most common task of management of the Laboratory was developed. With this tool you could turn on, turn off and install computers remotely via a graphical interface. One purpose of this project is to replace that old tool with a more powerful program and complete some of the tasks that were not incorporated. Other aspect to improve is to execute certain tasks concurrently.

Another motivation for developing the project was the fact that the tool would be used in a **real environment**, such as the Computer Science and Engineering Department. Some students do not have the opportunity to evaluate their projects in scenarios that are designed for.

1.2. Goals

After exposing the limitations of the tools, we list some of the objectives to be achieved by the desired solution. These are:

- The solution should be able to perform the **same tasks as the previous tool**: **turn the computer on**, **restart it**, **shutdown** it and to do **remote installations** of previously generated images of two different operating systems (*Debian* and *Windows*).
- The solution should be able to perform **other tasks** that are performed frequently in the laboratory, such as **sending virtual machines** to computers and **changing permissions** (read, write and execute) associated with files of these virtual machines.
- The solution should be able to apply the above described tasks on **different computers in different classrooms** simultaneously.
- The solution should implement a **graphical user interface** that helps to perform tasks on computers in the classroom.
- The solution should be **multiplatform**.

1.3. Document structure

Here are the different chapters in which the document is divided, including a brief summary of their content:

- **Chapter 1: Introduction**. This section provides a general approach to the project, detailing the context of its development and the goals to be achieved. This section also describes the structure chosen to organize the document.
- **Chapter 2: State of the art**. This section describes some tools that represent an approach to the problem. In addition to this, it is carried out a comparison of existing technologies that can be used to build the final solution.
- **Chapter 3: Analysis**. This section describes and models the problem to be solved with a detailed definition of the requirements of the system.
- **Chapter 4: Design**. This section shows all construction specifications and design decisions to solve the problem that was modeled in the previous section.
- **Chapter 5: Implementation**. This section describes the process of installing and configuring the operating environment.
- **Chapter 6: Evaluation**. This section covers the testing phase, which serves to assess the quality of the system and verify that the system meets the requirements set by the client, in addition to estimating user satisfaction.
- **Chapter 7: Implantation**. This section details the implantation strategy of the system, including the deployment process, installation and configuration of the application. It also includes a reference to the manual.
- **Chapter 8: Planning and budget**. This section provides a diagram with the estimated project schedule and a detailed calculation of the costs and benefits obtained.
- **Chapter 9: Conclusions and future work**. This section provides some concluding remarks about the project and some ideas and extensions that enhance the features of the application.
- **Appendix I: Description of initial prototype**. This appendix includes a detailed description of the basic model that was the basis for the final application.
- **Appendix II: Computer management manual with Kosmos**. The manual included in this appendix is intended for end users of the application, as a guide on computers management with Kosmos.
- **Appendix III: Summary in English**. Brief summary needed to demonstrate the skills of students in English.
- **Bibliography**. This section lists sources and references consulted during the development of the project.

2. Planning and budget

This section provides a diagram with the estimated project schedule and a detailed calculation of the costs and benefits obtained.

2.1. Planification

The project schedule was based on the delivery period of the final degree project, that this year was set between 15 and 22 June 2016, and on the number of hours for each credit ECTS. Because the final degree project has a value of 12 ECTS and each of these credit equals 25 hours of student work, a duration of **20 weeks** (15 working hours a week) is estimated. Thus, the project start date is set on **25 January, 2016** and the estimated date of completion on **June 10, 2016**.

Project planning is divided into the following main phases:

- **Documentation**.
- **Project proposal**.
- **Analysis**:
  - Assessment of alternatives.
  - User requirements analysis.
  - Software requirements analysis.
- **Design:**
  - Architecture design.
  - Use case design.
  - Class design.
  - Physical data design.
- **Implementation**.
- **Implantation**:
  - Team formation.
  - Preparation of infrastructure.
  - Installation of components.
  - Data load.
- **Evaluation**:
  - Testing plan.
  - User satisfaction.

This Gantt chart details the estimation of each of these phases:

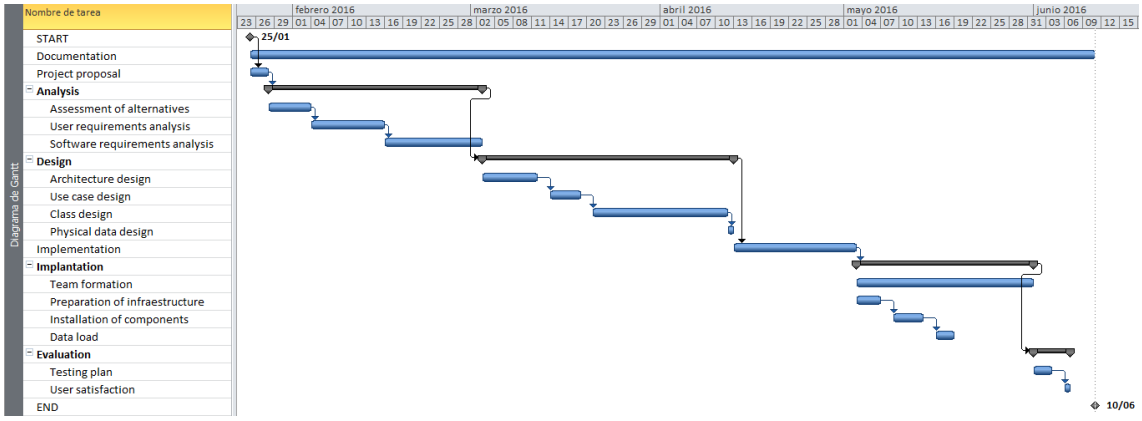


Figure 65: Gantt chart with estimated project planning

2.2. Budget

Personnel costs are calculated based on the previous planning and the average wage of different roles. After, the cost calculation of the redeemable material, overhead as renting a workplace and other related costs and total cost.

2.2.1. Personnel costs

We have identified two major roles that include the main features to carry out the development of the different phases of the project. Below is a table that breaks down the cost of salary of workers based on statistics of average salaries of these roles in the current market, assuming no previous experience in the workplace.

For contributions for **Social Security** (employer's contribution) the calculation is made on the annual gross salary of each worker. The extra payments are not included in the breakdown of costs, because they are prorated between the different monthly payments.

SALARIES		
	System analyst	**Programmmer**
Annual gross salary	28.020,00 €	22.836,00 €
Monthly gross salary	2.335,00 €	1.903,00 €
EMPLOYER'S CONTRIBUTION		
Social Security (23,6 %)	551,06 €	449,11 €
Unemployment (5,5 %)	128,43 €	104,67 €
Job training (0,6 %)	14,01 €	11,42 €
Fogasa (0,2 %)	4,67 €	3,81 €
Monthly employer contribution	698,17 €	569,00 €
PLANNING OF THE PROJECT		
Project duration (hours)	300	300
Project duration (weeks)	20	20
Hours worked per week	15	15
Hours worked per month	40	40
PERSONNEL COSTS		
Monthly cost of worker	3.033,17 €	2.472,00 €
Cost of hour worked	75,83 €	61,80 €
Personnel costs associated with the worker	22.748,74 €	18.539,98 €
Personnel costs		**41.288,72 €**

Table 106: Personnel costs

2.2.2. Equipment costs

Below is the price of equipment for operating environment. To estimate the price of these resources we have been consulted vendors who sell these products through Internet. All prices include *Value Added Tax (VAT)*.

EQUIPMENT COSTS		
Concept	**Model**	**Price**
Instalador server	Sun Microsystems Sun Fire V40z	1.268,00 €
Minardi computer	Acer Ferrari One 200-312G25n	399,00 €
Toleman computer	Toshiba Portégé Z30 Ultrabook	699,00 €
Equipment costs		**2.366,00 €**

Table 107: Equipment costs

2.2.3. Indirect costs

For the calculation of the costs associated with the physical equipment used during the project it is assumed that they all have a shelf life of 4 years. Therefore, it has taken a depreciation rate of 25% per year for computers (taking a year of 52 weeks) and 33% for software. Because the project duration is 20 weeks, we only included as an expense to the project the depreciation costs for that period of time. All prices include *Value Added Tax (VAT)*.

COSTS OF RESOURCES FOR DEVELOPMENT				
Concept	**Model**	**Quantity**	**Price**	**Total**
Desktop computers	Acer AXC-705	2	399,00 €	798,00 €
Monitors	Acer V206HQLAb	4	75,00 €	300,00 €
Printers	HP LaserJet Pro M201n	1	179,00 €	179,00 €
Office suite	Microsoft Office Home & Business 2010	2	279,00 €	558,00 €

AMORTIZATION				
Concept	**Annual amortization**	**Monthly amortization**	**Weeks**	**Total**
Desktop computers	199,50 €	3,84 €	20	76,73 €
Monitors	75,00 €	1,44 €	20	28,85 €
Printers	44,75 €	0,86 €	20	17,21 €
Office suite	184,14 €	3,54 €	20	70,82 €
Total				**193,61 €**

OTHER COSTS	
Concept	**Price**
Local rental (includes electricity and water)	450,00 €

INDIRECT COSTS	
Indirect costs	**643,61 €**

Table 108: Indirect costs

2.2.4. Total cost

Total cost is the sum of the costs previously disaggregated plus a margin of risk (10%) and a benefit (15%). To this amount is applied 21% of the value added tax (VAT), resulting in a final price of the project of **SIXTY-SEVEN THOUSAND AND ONE EUROS AND TWENTY TWO CENTS**.

DIRECT COSTS	
Personnel costs	41.288,72 €
Equipment costs	2.366,00 €
INDIRECT COSTS	
Indirect costs	643,61 €
SUBTOTAL	
Subtotal	44.298,33 €
RISK MARGIN (10%)	
Risk margin	4.429,83 €
PROFIT MARGIN (15%)	
Profit margin	6.644,75 €
TOTAL COST OF THE PROJECT	
Total cost	**55.372,91 €**
VAT (21%)	
Value Added Tax	11.628,31 €
FINAL PRICE OF THE PROJECT	
Final price	**67.001,22 €**

Table 109: Final price of the project

3. Conclusions and future work

This section provides some concluding remarks about the project and some ideas and extensions that enhance the features of the application.

3.1. Conclusions

3.1.1. Personal conclusions

Personally I feel very proud to have developed a project like this, because it made me knows more deeply my own abilities to organize my time and work autonomously. It also enhanced my skills and made me understand the value of responsibility before a project. In fact, one of the conclusions that I most appreciated was to discover how valuable perseverance is during the development of a project and set goals for the short, medium and long term.

On the other hand it gives me a great satisfaction that the project may eventually be used in the Laboratory of Computer Science and Engineering Department. It would be a way of thanking all the technicians who have taught me during the two years I spent working with them as a trainee.

Although all subjects of the degree allowed me to develop the project, the most important concepts I have acquired in the following:

- **Cryptography and Computer Security**. It is essential to understand some concepts related security protocols such as SSH, which uses public key cryptography for authentication of computers.
- **Programming**. Because the project is based on the development of a computer application, I needed to apply the knowledge on object-oriented programming I learned in this subject.
- **Algorithms and Data Structures**. Certain data structures in which information of classrooms and equipment is stored are lists and dictionaries of key-value pairs, whose treatment is related to the algorithms and structures studied in this subject.
- **Computer Networks**. For the implementation of the tasks you can perform with Kosmos, because it is necessary to know some concepts of network configuration and protocols.
- **Operating Systems, Operating Systems Design, Computer Architecture, Distributed Systems**. Without the concepts related to concurrency and synchronization of processes learned in these subjects I could not have developed such a powerful, robust and fast application as Kosmos.
- **User Interfaces**. It was necessary to design the application interface, because some design patterns and certain heuristics were applied to develop the initial model.

- **Software Engineering** and **Software Development Projects Management**. The whole process of documentation contained in this document is studied in these two subjects.

In addition to these skills, I have learned on my own some other specific issues such as programming in Python, the development of interfaces with graphic libraries such as *GTK+*, aspects related to the installation and configuration of *Linux* servers or networking concepts as the standard *Wake-on-LAN* or *SSH* protocol.

3.1.2. Conclusions related to the developed Kosmos

The tool developed in this project has successfully met the objectives for which it was designed, as they were presented in the introduction of this document:

- The application allows you to turn computers on, restart them and shutdown them remotely. It also enables remote installation of *Windows* and *Debian* images.
- The application allows you to transfer virtual machines to computers and modify the permissions of a file or directory. This includes files and directories for virtual machines, which is one of the tasks for which the application was designed.
- The application allows you to perform tasks on different computers in different classrooms at the same time by implementing these tasks with threads.
- The application has a graphical user interface that facilitates intuitive realization of the above tasks.
- The application can run on *Windows* and *Debian* because it is only necessary to connect to the server and export window system to run it.

3.1.3. Conclusions about the software process

The deadlines for completion of the project have been met satisfactorily. In fact, the estimated 20 weeks programming could have been reduced to a minimum of 18 weeks without need to expand the number of workers or increase their workload. In addition this would have resulted in a reduction of the final price of the application of up to 9.32%, since the personnel costs would have been lower by reducing the number of hours of each worker on the project.

3.2. Future work

This section includes some ideas and extensions that arose during the development of the project and could improve the characteristics of the application. Most of them could not be implemented for lack of time or because its development would have meant such a large task as another final degree project.

Security

One aspect that could be improved has to do with security: because it has no continued official support, also it lacks regular updates that can correct security errors that have not been detected during development. In addition, updating libraries used by the application could lead to other unexpected errors, because it also lacks a maintenance team that can correct problems that may arise during the process.

Classrooms editing

One improvement would be to include an option that allows editing the classroom through the interface. For example, an editing window with options such as adding new classrooms or new computers to classrooms that already exist, modify the names of these or the installations that can be performed in each classroom, or change the distribution of classroom's maps.

Usability enhancements

Indicators of progress for tasks can be introduced. This would allow users to observe in real time the progress of each task on each computer or each classroom. Another solution could be to include small icons next to the name of the computer showing the operating system.

Configurable tasks

Another improvement that can be implemented is the ability to configure tasks or installations through the interface. This enhancement would allow you to modify some parameters relating to the data transfer rate, maximum and minimum wait times, the name of the network interface or the port through which the installation is done, from the application itself.

Performance monitoring

One of the most complex improvements that emerged during the development of the project was the possibility of implementing a system of regular monitoring to observe the overload of the computers in real time. At first we use the tool **Munin**, however, it was decided finally dismiss it because the idea was comparable to another final degree project.

Analysis of data

A very useful improvement would be the development of a logs storage system and any errors that may occur during the application of a task or installation, rather than display them on the server's terminal from which the application was run. The Kosmos application could also implement a system that would generate monthly or quarterly reports with statistics on how many computers have been installed or the use of computers, and other data that could be valuable for the management of the Laboratory of Computer Science and Engineering Department.

Remote update of systems

It would also be useful to add an additional task that would allow updating computers remotely. Thus it could include the ability to upgrade packages or specific programs with the **apt-get** command of *Linux*, or *chocolatey* tool in *Windows*. The application could also implement remote upgrade of the operating systems, but all of these improvements involve hard work and could not be done for lack of time.

Accessibility

Accessibility is very important for all users to use a tool on an equal basis, regardless of whether they have a disability. For this reason it is very important to assess whether the application meets all accessibility recommendations and standards, and otherwise adapt the system to do it. This improvement could not be done for lack of time.

Mobile app

Another idea would be to develop an application for Android or iOS; users could check the occupation of the classrooms from a mobile device. Because the connection could be carried out from anywhere, this improvement would have to pay close attention to the security mechanisms implemented.

Bibliografía

En este apartado se enumeran las fuentes y referencias consultadas durante la realización del proyecto.

[1] "LogMeIn", *LogMeIn* [en línea]. Disponible en: https://secure.logmein.com/home/es. [Consulta: 26 Febrero 2016].

[2] "LogMeIn Central - Getting Started Guide", *LogMeIn* [en línea]. Disponible en: https://secure.logmein.com/welcome/documentation/EN/pdf/Central/LogMeIn_Central_GettingStarted.pdf. [Consulta: 26 Febrero 2016].

[3] "TeamViewer", *TeamViewer* [en línea]. Disponible en: https://www.teamviewer.com/es/. [Consulta: 29 Febrero 2016].

[4] "RemoteUtilities", *Remote Utilities* [en línea]. Disponible en: http://www.remoteutilities.es/. [Consulta: 30 Febrero 2016].

[5] "Java Vs. Python – Which Programming Language is More Productive? – Infographic", *Perception Blog* [en línea]. Disponible en: http://blogs.perceptionsystem.com/infographic/java-vs-python-programming-language-productive/. [Consulta: 3 Marzo 2016].

[6] "Comparación de Rendimiento entre Python, Java y .Net", *PythonAr*. Disponible en: http://www.python.org.ar/wiki/RendimientoPythonVsJavaVsNet. [Consulta: 3 Marzo 2016].

[7] "Comparing Python to Other Languages", *Python* [en línea]. Disponible en: https://www.python.org/doc/essays/comparisons/. [Consulta: 3 Marzo 2016].

[8] "Teoría de la información", *Textos Científicos* [en línea]. Disponible en: http://www.textoscientificos.com/informacion/teoria. [Consulta: 3 Marzo 2016].

[9] "C++", *cplusplus* [en línea]. Disponible en: http://www.cplusplus.com/. [Consulta: 5 Marzo 2016].

[10] "Java", *Java* [en línea]. Disponible en: https://www.java.com/es/. [Consulta: 5 Marzo 2016].

[11] "Python", *Python* [en línea]. Disponible en: https://www.python.org/. [Consulta: 5 Marzo 2016].

[12] S. Raschka. "The key differences between Python 2.7.x and Python 3.x with examples", *sebastianraschka* [en línea]. Disponible en: http://sebastianraschka.com/Articles/2014_python_2_3_key_diff.html. [Consulta: 5 Marzo 2016].

[13] "Qt", *Qt* [en línea]. Disponible en: http://www.qt.io/. [Consulta: 9 Marzo 2016].

[14] "The GTK+ Project", *The GTK+ Project* [en línea]. Disponible en: http://www.gtk.org/. [Consulta: 9 Marzo 2016].

[15] "Python Editors", *Python* [en línea]. Disponible en: https://wiki.python.org/moin/PythonEditors. [Consulta: 15 Marzo 2016].

[16] "Wingware Python IDE", *Wingware Python IDE* [en línea]. Disponible en: http://www.wingware.com/. [Consulta: 15 Marzo 2016].

[17] "PyCharm", *PyCharm* [en línea]. Disponible en: https://www.jetbrains.com/pycharm/. [Consulta: 16 Marzo 2016].

[18] "Anjuta DevStudio", *Anjuta DevStudio* [en línea]. Disponible en: http://anjuta.org/features/. [Consulta: 16 Marzo 2016].

[19] "GNU General Public License", *GNU Operating System* [en línea]. Disponible en: https://www.gnu.org/copyleft/gpl.html. [Consulta: 14 Abril 2016].

[20] "Software QA - ¿Cuáles son los tipos de pruebas software?", *El Blog de Panel Sistemas* [en línea]. Disponible en: http://blog.panel.es/index.php/software-qa-cuales-son-los-tipos-de-pruebas-software/. [Consulta: 18 Abril 2016].

[21] J. Hernan Abad Londoño "Tipos de pruebas de software", *Ingeniería de Software* [en línea]. Disponible en: http://ing-sw.blogspot.com.es/2005/04/tipos-de-pruebas-de-software.html. [Consulta: 18 Abril 2016].

[22] "Manual del IDE Anjuta", *stuff.mit.edu* [en línea]. Disponible en: https://stuff.mit.edu/afs/athena/system/amd64_deb50/os/usr/share/gnome/help/anjuta-manual/es/anjuta-manual.xml. [Consulta: 4 Mayo 2016].

[23] "Compara tu salario", *Tusalario.es* [en línea]. Disponible en: http://www.tusalario.es/main/salario/comparatusalario. [Consulta: 13 Mayo 2016].

[24] "Seguridad Social: ¿Cuánto pagamos realmente?", *blog.iese.edu* [en línea]. Disponible en: http://blog.iese.edu/martinezabascal/2014/03/12/seguridad-social-cuanto-pagamos-realmente/. [Consulta: 13 Mayo 2016].

— FIN DEL DOCUMENTO —