Universidad de las Ciencias Informáticas

Título:

Serere 3.0. Instalador del sistema operativo Nova GNU/Linux.

Autor: *Juan Guillermo López Castellanos*

Tutores: *Msc. Siovel Rodríguez Morales*

Ing. Jorge Luis Machín Castillo

Ing. Yunier Soler Franco

Presentado en opción del título de Ingeniero en Ciencias Informáticas

Ciudad de La Habana, Cuba.

Junio de 2012.

Declaración de autoría

Declaro por este medio ser el único autor de esta investigación y además reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste, firmo la presente a los ____ días del mes de _____ del año 2012.

_____

Juan Guillermo López Castellanos

_____ _____

Msc. Siovel Rodríguez Morales Ing. Jorge Luis Machín Castillo

_____

Ing. Yunier Soler Franco

A mis padres. Todo por ustedes.

Agradecimientos

- A mi familia, por ayudarme a cumplir mi sueño de ser universitario.
- A Henry, por estar siempre en las buenas y en las malas.
- A Tomás “el curro”, por los chistes infinitos, el refresco gaseado y las integrales múltiples.
- A Ricardo, Héctor y mis demás compañeros del grupo, por convertir mis cinco años de carrera en una experiencia inolvidable.
- A Miranda, Machín, Abel, Dariem, Miguel y “el puro”, por enseñarme lo genial que pueden llegar a ser los jefes.
- A mis profesores.
- A mi tío René, por apoyarme hasta su último día... y después.

Resumen

El uso de sistemas operativos propietarios en Cuba es una tendencia que atenta contra los principios de soberanía del país y que además, compromete sin dudas la seguridad, confiabilidad y sostenibilidad de los procesos nacionales que utilizan las tecnologías de la información. El sistema operativo Nova surge en el año 2006 como una alternativa libre, para satisfacer las necesidades informáticas de instituciones y empresas cubanas. Desde entonces ha permanecido a la vanguardia del proceso de migración que atraviesa la isla.

En este trabajo se realizó el diseño e implementación de la versión 3 de Serere, el software encargado de llevar a cabo el proceso de instalación del sistema operativo Nova, ya que actualmente el mismo no satisfacía las necesidades del proyecto por su carencia de funcionalidades. Para esto se realizó un estudio de los principales instaladores existentes en la actualidad. Se demostró mediante sus deficiencias que no era factible su uso en el proyecto y se propuso un conjunto de sus características más relevantes como adición a la nueva versión del instalador de Nova GNU/Linux.

Se efectuó el proceso de desarrollo mediante la metodología OpenUP y además fueron documentadas las herramientas, bibliotecas y lenguajes de programación utilizados.

Como resultado de la investigación se obtuvo un producto que agrupa el 89% de las funcionalidades de los instaladores modernos y que responde satisfactoriamente los requerimientos de las tres personalizaciones del sistema operativo cubano Nova.

Palabras clave: software libre, instalador, Nova, Serere.

Índice de contenido

Índice de tablas

Índice de ilustraciones

Introducción

Con el fin de lograr que Cuba alcance la independencia tecnológica, el Centro de Software Libre (CESOL) de la Universidad de las Ciencias Informáticas (UCI), media como entidad rectora del proceso de migración a tecnologías de software libre que atraviesa la isla desde hace varios años. Un avance en la sustitución de herramientas y plataformas privativas por otras alternativas libres, es sin dudas necesario para reducir costos en proyectos cubanos por concepto de pago de licencias y patentes, pero además, para lograr la implantación de un modelo que garantice la seguridad, personalización y soporte de las herramientas informáticas empleadas en muchos procesos cotidianos [1]. Dentro de este ámbito, y posterior a la creación en el país del Grupo Técnico Nacional en el año 2005, se ha prestado especial atención al uso de sistemas operativos y otras plataformas libres [2].

Un sistema operativo es un programa o conjunto de programas que en un sistema informático gestionan los recursos de hardware y proveen servicios a los programas de aplicación, y que además se ejecuta en modo privilegiado respecto de los restantes [3]. Como tal, es un elemento indispensable entre la interacción de un usuario con la computadora. Estudios realizados desde el año 2007 [4], indican que nuestro país no es la excepción de la tendencia mundial al uso de sistemas operativos propietarios como Microsoft Windows, lo que provoca dificultades a la hora de realizar algunos procesos fundamentales como la actualización, el soporte y la validación del software mediante el pago de licencias, debido a las presiones económicas que ejerce los Estados Unidos sobre nuestra isla.

A raíz de la existencia de esta necesidad de cambio, surge en el año 2006 en la Universidad de las Ciencias Informáticas la distribución Nova GNU/Linux, la cual fue declarada posteriormente como plataforma operativa oficial del proceso de migración en Cuba. El objetivo principal de Nova es ofrecer una solución personalizada para cubrir las necesidades informáticas de todo centro o institución cubana que

utilice las TIC[1], lo que ofrece como resultado un producto enfocado completamente en el usuario final, con disímiles características funcionales y éticas que lo hacen ideal para el funcionamiento en nuestro entorno [5].

Para desplegar el sistema en una terminal, Nova cuenta desde sus inicios con Serere, un software cuyo propósito es llevar a cabo el proceso de instalación. Dicho procedimiento constituye una fase crítica en cualquier sistema operativo, ya que garantiza en cierta medida el buen funcionamiento y la correcta configuración inicial del mismo [6]. A pesar de que en su versión actual (2.0) Serere cumple con los requerimientos básicos para instalar un sistema operativo, no se adapta a las condiciones y necesidades reales del proyecto.

Actualmente el instalador no permite agregar aplicaciones adicionales desde repositorios externos. El usuario recibe un sistema solo provisto con las aplicaciones incluidas en el CD del producto y debe instalarlas manualmente una vez concluido el proceso. Además, no es posible manejar configuraciones avanzadas de almacenamiento como RAID y LVM desde la aplicación, lo que representa una dificultad crítica para la versión de servidores de Nova.

Desde el punto de vista de su usabilidad, el software tampoco cubre todas las expectativas de los clientes. Entre los problemas de este tipo puede mencionarse que los modos de instalación actuales de Serere no son apropiados para la diversidad de usuarios que utilizan el sistema, existiendo inconformidades tanto por parte de aquellos que buscan un proceso de instalación fácil e intuitivo, como por los que requieren de una mayor personalización.

Teniendo en cuenta la problemática anterior, se puede entonces plantear el siguiente **problema científico**: ¿Cómo mejorar el proceso de instalación del sistema operativo Nova GNU/Linux, en base a las necesidades actuales del proyecto y de los usuarios del sistema?

1 Acrónimo de “Tecnologías de la información y las Comunicaciones”.

Como **objeto de estudio** de esta investigación fueron seleccionadas las aplicaciones instaladoras de las principales distribuciones de GNU/Linux existentes en el período actual, para así enmarcar los resultados a un **campo de acción**, que en este caso se trata de la estructura y funcionalidad del instalador de Nova.

Se trazó como **objetivo general** desarrollar la nueva versión del instalador del sistema operativo Nova GNU/Linux, y como **objetivos específicos**:

1. Sistematizar en el estudio de instaladores de distribuciones GNU/Linux existentes en la actualidad.
2. Realizar el diseño de la nueva versión de Serere.
3. Implementación del software requerido.
4. Realizar pruebas unitarias y de aceptación al sistema desarrollado.

Todo lo anterior defiende la **idea** de que si se desarrolla la nueva versión de Serere, pudiera mejorarse el proceso de instalación del sistema operativo Nova GNU/Linux.

Las tareas que dirigieron la investigación en pos de cumplir los objetivos trazados en la misma fueron:

1. Revisión bibliográfica acerca de los procesos de instalación de sistemas operativos.
2. Revisión bibliográfica sobre la estructura arquitectónica y las funcionalidades de aplicaciones de instalación de distribuciones GNU/Linux.
3. Levantamiento de requisitos de la versión 3.0 de Serere.
4. Concepción de una arquitectura y un diseño eficientes que garanticen la efectividad, reutilización y mejora del producto.
5. Implementación del diseño propuesto.

6. Realización de pruebas unitarias y de aceptación al producto.

Se utilizó el método **analítico-sintético** para realizar un análisis de los instaladores, comprender sus características y adaptarlas a Serere. También de manera alternativa fueron empleadas las técnicas de encuesta y entrevista para la recopilación de información [7].

Capítulo 1 Fundamentación teórica

El presente capítulo desarrolla un estudio sobre los instaladores de las distribuciones de GNU/Linux más utilizadas en la actualidad, con respecto a sus funcionalidades. Se realiza un análisis sobre la factibilidad de cada uno aplicado al proceso de instalación y las necesidades de Nova, lo que permitirá adoptar alguno de los mismos como solución, o en su defecto, mejorar la aplicación existente con un conjunto de sus características más novedosas.

1.1 Antecedentes de Serere

Previo al análisis y las consideraciones de este capítulo, se realiza en este epígrafe una descripción del sistema de instalación actual de Nova GNU/Linux. Además, son mencionadas las características funcionales del software en su última versión y sus deficiencias más significativas.

1.1.1 Descripción general del sistema

La necesidad de un instalador para Nova fue detectada desde los mismos inicios del desarrollo de la distribución, por lo que se decidió estudiar la posibilidad de utilizar alguno de los instaladores de otras distribuciones de GNU/Linux o desarrollar uno. Este estudio arrojó como resultado que se debía desarrollar un instalador, pues los existentes eran inconsistentes en ese entonces con los propósitos del proyecto [8].

Desarrollado principalmente en Python y actualmente con soporte de interfaz en Qt y modo texto, la segunda versión de Serere arribó en el año 2011 y se ha mantenido con pocas variaciones hasta el momento. Posee un total de 8 vistas (**Anexo 1**) mediante las cuales el usuario efectúa la instalación y configuración del sistema.

1.1.2 Características funcionales

Las funcionalidades que en la actualidad soporta Serere en el proceso de instalación, son las siguientes:

- Permite la selección de idioma y variantes de teclado para el sistema.
- Posibilita que el usuario seleccione la zona horaria en que se encuentra.
- Incorpora un particionador propio con soporte para sistemas de ficheros básicos, incluyendo "ext4"[2].
- Permite el inicio automático de sesión.

1.1.3 Características no funcionales

También el análisis estructural de la herramienta posibilitó la mención de sus principales características internas y estilo arquitectónico:

- Arquitectura con estilo llamada-retorno, basado en un patrón estructural de capas [9].
- Utilización de metadatos para algunos aspectos de configuración.
- Reutilización de componentes externos y bibliotecas del lenguaje.
- Está implementado mayormente en Python y su interfaz visual en Qt.

2 Un sistema de archivos transaccional liberado como una mejora compatible de 'ext3'. Soporta volúmenes de hasta 1024 PiB, usa menos CPU y mejora la velocidad de lectura y escritura.

1.1.4 Deficiencias observadas:

- Solo disponible en idioma Español.
- Particionador sin soporte para particiones y dispositivos RAID y LVM.
- Solo 3 tipos estáticos de instalación (todo el disco, espacio libre, y modo personalizado).
- Imposibilidad de actualizar una versión anterior del sistema o importar configuraciones de sistemas existentes.
- No permite la instalación de aplicaciones adicionales ni configuración de fuentes externas para este propósito.
- No permite el cifrado de sistemas de ficheros.
- Alto nivel de acoplamiento entre algunos módulos y paquetes del sistema.
- Uso incorrecto del principio de modularidad en algunos casos.
- Integración del sistema en un paquete único.
- Inexistencia de una arquitectura documentada.
- Ausencia de un estándar de documentación para el código.
- Violación del principio DRY [10][11] en algunos módulos del sistema.

1.2 Estudio de arte sobre distribuciones exitosas

Con el fin de obtener una panorámica de las tendencias que actualmente son utilizadas en el mundo para llevar a cabo los procesos de instalación en distribuciones GNU/Linux, se desarrollará un estudio de algunas de las aplicaciones de este tipo existentes en la actualidad. El análisis de la estructura arquitectónica y funcional de las mismas, permitirá tomar la decisión de si es conveniente utilizar alguna de estas en la instalación de Nova, o en caso de que ninguna cumpliese las expectativas, utilizar sus características más relevantes para complementar la solución actual.

Tomando como criterios principales la innovación y la aplicación práctica de sus funcionalidades, así como su estructura y estilo, fueron seleccionadas tres aplicaciones para su posterior análisis. Las mismas representan una muestra de los programas de instalación que son mantenidos por algunas de las distribuciones de Linux más activas en la actualidad[3]. Los datos ofrecidos en el estudio de cada individuo de la muestra se centraron específicamente en los siguientes aspectos:

- Descripción general: reseña y antecedentes de la aplicación, así como otros detalles generales.
- Estudio de funcionalidad: un análisis detallado de las características innovadoras de cada producto en cuanto a funcionalidad se refiere.
- Entorno de producción: herramientas utilizadas en su desarrollo, bibliotecas, servicios de terceros, y otros aspectos, en caso de que alguno de estos fuese relevante.

3 Según datos estadísticos actualizados ofrecidos por el sitio http://www.distrowatch.com

1.2.1 Ubiquity. Instalador de Ubuntu GNU/Linux.

Ubuntu es la segunda distribución más usada en el mundo al momento de realizarse esta investigación, y de igual forma, una de las más estables y enfocadas al trabajo del usuario final [12]. Canónical, la empresa que da soporte a este sistema operativo, se ha propuesto desde sus inicios en convertirlo en la plataforma que rompa los tabúes de baja usabilidad que ostentaban los sistemas basados en Linux desde hace ya varios años, y ciertamente ha alcanzado el éxito en este sentido. Actualmente Ubuntu es usado por millones de usuarios y negocios del mundo, con fines empresariales y personales [13]. Además posee una gran comunidad en Internet que contribuye a su actualización, desarrollo y revisión.

Posiblemente por estos factores Ubiquity puede calificarse como uno de los instaladores más dinámicos y usables existentes en la actualidad [14]. En su versión para la liberación 11.10 del sistema operativo (la más reciente al término de esta investigación), el software posee 7 vistas de instalación básicas, aunque el número de estas varía en dependencia del nivel de configuración que escoja el usuario durante el proceso. Está desarrollado en Python, posee *frontends*[4] para GTK[5] y Qt[6], y ha sido portado a otras distribuciones también basadas en Ubuntu como Linux Mint, Kubuntu y gNewSense.

La observación y reingeniería parcial esta aplicación, permitió el señalamiento de las funcionalidades y características más distintivas de la misma.

- **El instalador como primera pantalla del sistema:** una característica a señalar y que Ubiquity ha mantenido desde su versión 9.4, es el enfoque de utilizar el instalador como primera interacción del

4 Un término en inglés utilizado para referirnos a la "parte visible" de una aplicación, o lo que comúnmente conocemos como la interfaz de usuario.

5 Siglas en inglés de GIMP Tolkit. Biblioteca originalmente diseñada para el trabajo con gráficos del editor de imágenes GIMP, y posteriormente adoptada como suite gráfica por defecto de GNOME, XFCE y otros entornos de escritorio en GNU/Linux.

6 Una plataforma de desarrollo de interfaces gráficas para aplicaciones de GNU/Linux, Windows y MacOS.

usuario con el sistema. La tradicional opción de probar o instalar provista anteriormente por los gestores de arranque en la mayoría de las distribuciones, es invocada ahora en la primera vista del instalador, lo que permite la no interrupción del proceso de carga del sistema. Pero el aspecto más significativo de esta mejora es que al proveer un gestor de ventanas para la ejecución del instalador, Ubiquity ofrece las opciones de gestión de red, accesibilidad, sonido, energía y apagado de forma concurrente, minimizando el número de pasos para la configuración de estos aspectos.

- **Selección dinámica de tipos de instalación:** además de los modos de instalación básicos (instalar en todo el disco e instalar en modo avanzado), Ubiquity es capaz de sugerir otros tipos de instalación de manera dinámica, basándose en el reconocimiento de los sistemas operativos existentes en la computadora. El objetivo de este cambio no es otro que encapsular una serie de operaciones (anteriormente posibles sólo en modo avanzado) en casos de uso más básicos y asequibles al usuario promedio. Reemplazar un sistema operativo existente e instalar sin modificar el sistema, son dos ejemplos que siguen esta filosofía.

- **Copia en segundo plano:** hasta el lanzamiento de la versión 9.4 de Ubuntu, todos los instaladores realizaban la instalación de un sistema en tres fases: recopilación de información básica, configuración adicional y copia e implantación [15]. Cada uno de estos ejecutándose consecutivamente. Más adelante Ubiquity introdujo el concepto de “copia en segundo plano”, donde solo son necesarios un conjunto de datos indispensables para comenzar la copia de archivos. Esta copia es realizada por un subproceso independiente y de esta forma el usuario puede seguir suministrando datos adicionales a la par del proceso. Esta innovadora característica utiliza la concurrencia para romper el mecanismo lineal de los instaladores tradicionales y disminuye el tiempo de espera del usuario final.

- **Cifrado de carpeta personal:** con el fin de proteger la confidencialidad del usuario final, Ubiquity

ofrece la posibilidad de cifrar el contenido del directorio "/home"[7] en la vista de configuración personal. Esta funcionalidad es muy útil sobre todo para ordenadores compartidos, o aquellos en que los usuarios requieran una mayor seguridad de acceso a sus datos.

- **Detección y configuración de interfaces de red:** junto a la posibilidad de reconocer una amplia gama de dispositivos de hardware, Ubiquity ofrece la opción de detectar y configurar cualquier interfaz de red conectada a la computadora. Dicha característica permite la instalación de software adicional a la distribución desde repositorios externos, así como otras posibilidades.

Por otra parte, el hecho de ser Ubuntu una distribución tan comprometida con un tipo particular de usuarios y responder de forma tan drástica a los cambios tecnológicos que ocurren a nivel mundial, usar su aplicación instaladora para los propósitos de Nova pudiese ser un arma de doble filo.

A pesar de sus muchas ventajas funcionales, Ubiquity no ofrece actualmente un modo de instalación en modo texto, o ningún otro que pueda ser utilizado en ordenadores de bajas prestaciones. Esto constituye una desventaja crítica para algunas personalizaciones de Nova cuyo propósito es instalarse en este tipo de computadoras. Tampoco incorpora la posibilidad de manejar formatos especiales de disco, al no ser estas funcionalidades una prioridad para la distribución y su nivel de configuración para tareas específicas como la gestión de red es insuficiente.

A manera de conclusión puede decirse que Ubiquity es una herramienta cuya filosofía y estilo de instalación proporcionan un gran valor a usuarios comunes y avanzados, pero la naturaleza de algunas de sus funcionalidades constituye una dificultad para los propósitos de Nova.

7 Directorio dentro de la raíz de un sistema GNU/Linux que almacena las configuraciones y archivos personales de un usuario.

1.2.2 Anaconda. Instalador de Fedora GNU/Linux.

Si bien muchas distribuciones han logrado hacer más usables los sistemas basados en GNU/Linux para los usuarios inexpertos y con necesidades básicas, pudiera decirse entonces que Fedora ha alcanzado llegar un paso más lejos: demostrar que trabajos y objetivos más complejos como la administración de redes, pueden ser desarrollados de igual forma y con las mismas facilidades [16].

Al ser utilizado por esta y otras distribuciones vanguardias en tópicos de gestión avanzada de redes como RedHat y CentOS, se encuentra en Anaconda uno de los ejemplos de instaladores que pudieran clasificarse como más completos [17]. Aunque mantiene una estructura simple y una arquitectura de información poco elaborada, los niveles de configuración que ofrece son sin dudas superiores a los provistos por una instalación básica.

Además de ser utilizada por Fedora, RedHat y CentOS, Anaconda ha sido portado a otros sistemas operativos entre los que pueden mencionarse a OzLinux, Gentoo, VidaLinux y Sabayon. Fue escrita utilizando los lenguajes Python y C, y posee *frontends* en PyGTK y Python-Newt.

El estudio de su funcionamiento en Fedora 15, permitió la acotación de interesantes características:

- **Instalación semi-supervisada:** es un mecanismo llamado *kick-start por sus creadores,* mediante el cual Anaconda instala y configura el sistema de forma automática. Su funcionamiento se basa en un archivo “anaconda-ks.cfg” que es generado al final de cualquier instalación manual, pero que también puede ser elaborado a mano o utilizando otras herramientas que RedHat facilita a los usuarios de la comunidad, como “system-config-kickstart” y “Clobber”. Básicamente un archivo de este tipo contiene una serie de opciones que serán suministradas al instalador y que describen cómo configurar el sistema. También puede incluir *scripts*[8] para ser ejecutados antes o después de

8 Un término utilizado por los programadores para referirse a un fragmento de código escrito en un lenguaje de programación generalmente estructurado, que se ejecuta con un objetivo específico.

la instalación. Este procedimiento diverge en muchos elementos del tipo de instalación que hasta el momento conocemos y sucede mediante los siguientes pasos:

1. La instalación es iniciada mediante un CD, memoria flash o cualquier otro medio.
2. El archivo *kick-start* es cargado del medio de arranque o descargado de la red.
3. Se ejecuta automáticamente una instalación con Anaconda, que lee el archivo y obtiene la ruta del árbol de instalación.
4. El instalador intenta una instalación desatendida y si alguna información requerida no se encuentra en el archivo o el mismo está configurado de forma incorrecta, Anaconda puede preguntar al usuario por información adicional.

- **Gestión de unidades RAID[9] y LVM[10]:** al especializarse las distribuciones que utilizan Anaconda en temas de administración de servidores y redes, su instalador incluye muchas facilidades para el trabajo con formatos RAID y LVM. En la vista de puntos de montaje de la aplicación pueden crearse particiones y dispositivos lógicos RAID a distintos niveles, así como volúmenes LVM físicos y grupos.
- **Configuración en dos bloques:** no toda la información en este instalador es recopilada en un recorrido ininterrumpido, sino que solo las características básicas (idioma, contraseña de administración, zona horaria y puntos de montaje) son requeridas en el programa de instalación.

9 Redundant Array of Independent Disks (conjunto redundante de discos independientes). Hace referencia a un sistema de almacenamiento que dependiendo de su configuración permite mayor integridad, tolerancia a fallos, rendimiento o capacidad a un disco.

10 Logical Volume Manager (Administrador de Volúmenes Lógicos). Es una implementación de un administrador de volúmenes lógicos para el kernel de Linux, e incluye varias características como redimensionado de grupos lógicos y otras.

Los datos restantes (usuarios y grupos, gestión de paquetes y repositorios, entre otros) se solicitan al iniciar el sistema por primera vez. Este enfoque también conocido como *first boot* o configuración en dos bloques, ya es utilizado por otros programas de instalación [18] y es una manera de separar físicamente los niveles de configuración del proceso.

- **Gestión de autenticación por red:** además de permitir la autenticación mediante cuentas locales, puede utilizarse alternativamente Kerberos, LDAP u otros métodos de autenticación de red. En una sección de la vista de usuarios son solicitados los datos básicos para la realización de esta tarea, como son el servidor de dominio, método de autenticación, servidores de administración y opciones DNS. Se permiten también otras configuraciones avanzadas como la especificación de algoritmos para *hashing*[11] de contraseñas y soporte para autenticación mediante tarjetas inteligentes.

- **Reporte de perfil de hardware:** "Smolt" es un sistema integrado a Fedora que permite generar estadísticas en línea mediante perfiles anónimos de Hardware, enviados por los usuarios de forma opcional en el último paso del asistente. De esta forma, mediante un sitio web[12] pueden obtenerse predicciones de funcionamiento del hardware del usuario en el sistema operativo, así como muchas otras estadísticas que contribuyen al soporte y despliegue del sistema.

A consideración del autor, una de las deficiencias de Anaconda es el hecho de ser un software de instalación que maneja principalmente distribuciones no basadas en Debian. Su uso en el proyecto supondría una cantidad significativa de cambios en la arquitectura de la aplicación y la reimplementación de muchas de sus funcionalidades como la gestión de paquetes. Por otra parte, el no ser un instalador pensado en su totalidad para usuarios con poca experiencia [19], pudiera atentar contra la aceptación del

11 Un "hash" o arreglo enumerativo es una estructura de datos que utiliza un método probabilístico para su funcionamiento interno. Al trabajo de datos con este tipo de estructura se le conoce como "hashing".

12 http://smolt.fedoraproject.org

sistema por parte de estos. También teniendo en cuenta los objetivos y metas de Nova, Anaconda desborda la necesidad funcional del proyecto, pero descuida otros aspectos como la usabilidad y la simplicidad en el proceso de instalación.

1.2.3 Yast. Instalador de OpenSuse GNU/Linux.

El proyecto OpenSuse es un programa comunitario respaldado por la compañía Novell, cuyo objetivo es lograr la promoción de los sistemas operativos GNU/Linux en cualquier lugar [20]. Principalmente con fines de escritorio y desarrollo, siguiendo la estrategia de simplificar dramáticamente el proceso de creación de paquetes y aplicaciones, Novell planea convertir a OpenSuse en la opción preferida de muchos desarrolladores y promotores de software [21].

Yast (anónimo de “Yet another system tool”) no es sólo un software de instalación, sino también una herramienta que facilita la administración del sistema una vez desplegado, así como la instalación de software en el mismo [22]. Es desarrollada principalmente en Perl y C++, con *frontends* en Qt y Curses[13]. Además el agradable diseño de su interfaz propicia una excelente experiencia para el usuario.

El estudio de tan completa herramienta en la versión 12.1 de OpenSuse, nos permitió explorar sus características más innovadoras:

- **Actualizar un sistema anterior:** en los primeros pasos del asistente Yast da la posibilidad al usuario de realizar una instalación nueva o de actualizar una versión anterior del sistema. Esta última es sin dudas una característica por la que todos los instaladores debieran apostar en algún momento, ya que representa una posibilidad de renovación con pérdida mínima de datos y configuraciones.

13 Librería añadida a disímiles lenguajes, que permite la representación de interfaces en terminales de consola como modo alternativo a la representación de solo texto.

- **Instalación de software adicional:** con la posibilidad de un medio capaz de almacenar software alternativo además del sistema base, Yast brinda al usuario una amplia gama de aplicaciones adicionales para ser instaladas en el sistema. La aplicación ordena los paquetes por categorías: entornos gráficos, funciones de escritorio, tecnologías de base, funciones de servidor y otras. Esta es una posibilidad más para desplegar sistemas personalizados desde el mismo momento de la instalación.

- **Importar configuración de sistemas:** al ser capaz de reconocer la presencia de otros sistemas operativos instalados en la PC, este instalador ofrece la capacidad de importar configuraciones de los usuarios, mediante un análisis de las particiones ocupadas por los sistemas foráneos. Pueden importarse documentos personales, imágenes y preferencias de escritorio, así como algunos archivos compatibles de configuración.

- **Resumen de instalación:** antes de comenzar el proceso de copia, Yast ofrece un resumen con todas las opciones seleccionadas en la instalación agrupadas por categorías. Lo anterior brinda al usuario una vista global del estado futuro de su sistema y le permite encontrar errores de selección en caso de que existan.

Yast es un sistema que combina alta funcionalidad con una buena usabilidad. Su deficiencia principal para resolver el problema de la presente investigación radica en la mantenibilidad del software. El estar escrito en lenguajes no manejados en el proyecto, constituye una deficiencia crítica para su soporte.

1.3 Consideraciones y análisis de mejoras

Al concluir el estudio de las 3 herramientas y conociendo de antemano las características de la versión más reciente de Serere, es posible afirmar que del total de funcionalidades analizadas, el software cuenta actualmente solo con el 19% de las mismas, tal como se evidencia en la Figura 1.

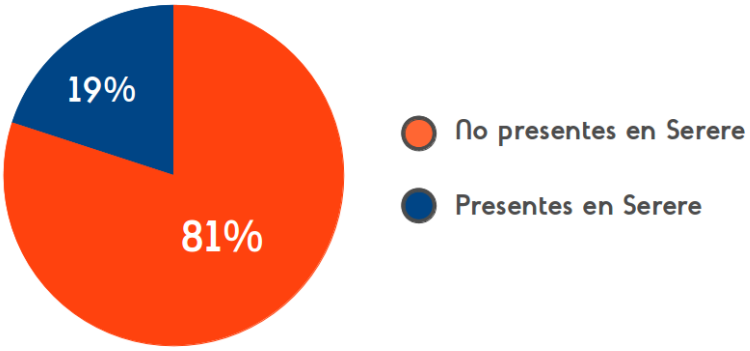


Fig 1. Funcionalidades y características analizadas presentes en Serere.

Esta discapacidad funcional ocasiona que en la actualidad el instalador no cumpla con las expectativas del proyecto Nova y también provoca un impacto desfavorable sobre los usuarios del sistema. Sin embargo, también fue una conclusión importante del estudio que no sería conveniente utilizar en Nova alguno de los instaladores analizados, ya que todos presentan al menos dos de las siguientes dificultades críticas:

- **Debido a sus deficiencias particulares, no son viables para la instalación de alguna de las personalizaciones del sistema operativo Nova.** Por ejemplo, los instaladores que no permiten tratamiento de dispositivos RAID y LVM.

- **Su diferencia de enfoque en algunas funcionalidades y características no son factibles para la situación de nuestro país.** Por ejemplo, aquellas aplicaciones instaladoras que consumen demasiados recursos, no incluyen interfaces ligeras de instalación, o no interactúan con hardware obsoleto.

- **Incumple con políticas establecidas del proyecto.** Por ejemplo, no ofrecen documentación de desarrollo, o utilizan lenguajes y componentes no factibles para Nova.

Las deficiencias anteriores son agravantes que excluyen a cualquier aplicación de ser una alternativa de

instalación viable para el sistema operativo Nova. La aplicación instaladora actual está exenta de las mismas y las deficiencias observadas en esta son de carácter funcional en un 70% como se muestra en la Figura 2.

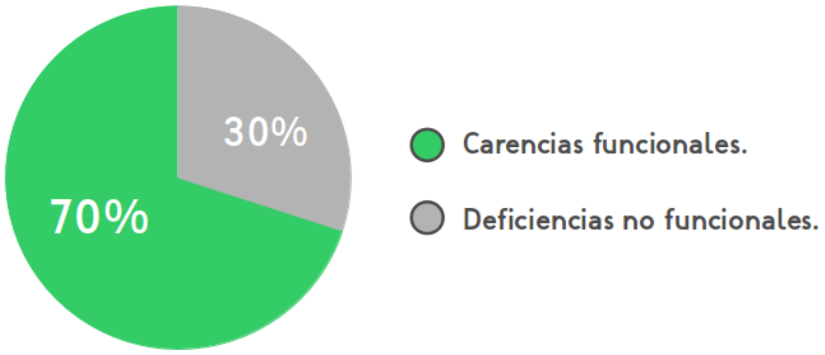


Fig 2. Naturaleza del total de deficiencias encontradas en Serere.

Como propuesta final de esta etapa se recomienda utilizar un conjunto de las funcionalidades más relevantes de las herramientas valoradas, para suplir el déficit funcional de Serere. De esta manera pudiese lograrse que solucione las necesidades que en la actualidad tiene el proyecto y las expectativas de los usuarios.

A continuación se mencionan algunas características candidatas a incluir en la próxima versión del software:

- **Soporte para idioma inglés.** Por la necesidad y posibilidad real de que la distribución sea distribuida y utilizada por la comunidad internacional en algún momento del futuro.
- **Gestión de particiones y unidades RAID y LVM.** Por el uso de Nova para la implementación de servidores y ordenadores dedicados a servicios que requieren este tipo de configuración de disco.

- **Otros tipos dinámicos de instalación**. Reemplazar un sistema operativo existente, instalar sin modificar el sistema, instalar mediante un archivo de configuración y otros.

- **Actualizar un sistema anterior.** Para permitir la actualización de una versión anterior de Nova a la versión actual con un mínimo de pérdida de datos y configuraciones.

- **Importar configuraciones personales de otros sistemas.** Para facilitar a los usuarios el proceso de configuración del nuevo sistema instalado.

- **Gestionar instalación de paquetes.** Para adicionar diversas funcionalidades que contribuirán a la personalización y mejora del instalador, como instalar software adicional y gestionar repositorios desde la aplicación. Esta mejora también requiere la detección y configuración de interfaces de red.

- **Cifrar sistemas de fichero.** Para aumentar la seguridad y confidencialidad de los datos almacenados en las unidades de disco del sistema.

- **Introducir copia en segundo plano.** Para reducir el tiempo de espera del usuario y de esta forma proporcionar una mejor experiencia de instalación.

A lo largo de este capítulo se investigó el estado actual de las aplicaciones instaladoras más utilizadas en el mundo. Además se demostró la necesidad de implementar una nueva versión de Serere, agregando un conjunto de funcionalidades para aumentar el valor de la herramienta en el proceso de instalación de Nova. Por último fueron propuestas como candidatas un conjunto de estas características.

Capítulo 2 Diseño del sistema y características de entorno

En el presente capítulo se realiza el proceso de elaboración y diseño del sistema. Esta fase es de suma importancia, ya que tiene como objetivo principal conocer las necesidades de los clientes y conformar la arquitectura del software. También se ofrece una descripción de las herramientas que serán utilizadas como parte del entorno de desarrollo.

2.1 Características de ambiente y herramientas

En el capítulo anterior se llegó a la conclusión de la necesidad de implementación de una nueva versión de Serere, teniendo en cuenta los resultados del estudio realizado. Para construir sistemas eficientes mediante un proceso de desarrollo eficaz que garantice la calidad de los resultados, es necesario asegurar un entorno de desarrollo que lo permita [23]. En este epígrafe se presentan de manera general las distintas herramientas utilizadas durante el proceso, que permitieron el aseguramiento del ambiente productivo para realizar las distintas tareas del ciclo.

2.1.1 Metodología de Desarrollo

OpenUP/Basic es un Framework de procesos de desarrollo de programas de código abierto, que con el tiempo espera cubrir un amplio conjunto de necesidades en el campo del desarrollo de programas. Permite un abordaje ágil al programa en análisis con sólo proveer un conjunto simplificado de contenidos, fundamentalmente relacionados con orientación, productos de trabajo, roles, y tareas [24]. Es un proceso interactivo de desarrollo de software simplificado, completo y extensible, para pequeños equipos de desarrollo que valoran los beneficios de la colaboración y de los involucrados, por encima de formalidades innecesarias [25].

Es la metodología orientada en el departamento de sistema operativo para el desarrollo de todos sus

productos, ya que permite detectar errores tempranos a través de un ciclo iterativo y también por ser una metodología centrada en el cliente.

2.1.2 Lenguaje de programación y modelado

Python es el lenguaje principal utilizado en el desarrollo de las distintas versiones de Serere. Esto se debe fundamentalmente a que es un lenguaje potente, con una comunidad de desarrollo muy activa que se revierte en una gran cantidad de módulos y bibliotecas muy bien documentadas [26], lo que posibilita el uso de estas en la implementación de funcionalidades del instalador. Para el desarrollo de Serere 3.0 este lenguaje fue nuevamente seleccionado, pues además de las características antes planteadas, es posible reutilizar y mantener con mayor facilidad los módulos, clases y funciones de la versión antecedente del sistema. El uso de este lenguaje también ha sido establecido como política del proyecto.

Para el modelado de los elementos arquitectónicos e ingenieriles del sistema fue empleado el lenguaje UML[14].

2.1.3 Gestión de proyecto

Para la gestión del proyecto en su totalidad fue utilizado Gespro [27], potente herramienta de gestión escrita usando el framework Ruby on Rails. Mediante la misma puede realizarse un seguimiento de las tareas y la planificación de varios proyectos, obtener reportes de los avances actuales sobre la línea base, acceder e interactuar con repositorios de versiones, y otras funciones útiles como administración de noticias, documentos y archivos.

14 El Lenguaje Unificado de Modelado (UML), es el lenguaje estándar para la representación del desarrollo de software y sistemas. Para lo anterior utiliza una serie de estereotipos que ilustran los diferentes procesos, objetos y entidades de un software, así como sus relaciones.

2.1.4 Control de cambios y versiones

Es de suma importancia en cualquier proyecto de desarrollo de software, ejecutar mecanismos de control y configuración de cambios. Mediante los mismos se asegura el acceso a la información importante, la integridad de los datos y el control de estos [28]. Las herramientas de control de versiones pueden ser muy útiles en este punto del entorno. Para el presente se utilizó *Subversion*, una de las aplicaciones de versionado más utilizadas a nivel mundial por las ventajas funcionales que ofrece y por su facilidad de uso e integración con aplicaciones de desarrollo [29].

2.1.5 Plataforma de desarrollo

Para desarrollar Serere 3.0, se decidió utilizar el entorno integrado de desarrollo Eclipse en su versión 3.7.0 (Indigo), que integrado a un conjunto de *Plug-Ins*[15], incluye una gran cantidad de funcionalidades utilizadas tanto para el desarrollo del instalador como para la gestión del mismo. Los componentes adicionados fueron los siguientes:

- **PyDev:** integra las funcionalidades de Eclipse para el desarrollo con Python. Permite la utilización de las bibliotecas de este lenguaje, reconocimiento sintáctico de código, refactorización, *debugger*[16] y consola interactiva.

- **PyUML:** mediante este complemento, Eclipse permite la generación de diagramas UML a partir de código fuente escrito en Python y viceversa.

15 Los "plug-ins" son fragmentos de software que añaden funcionalidad a una aplicación determinada. Generalmente son instalados de manera adicional.

16 Una herramienta que permite monitorizar la ejecución de una aplicación en su totalidad o un fragmento de la misma, observando los valores que toman sus variables, llamadas realizadas, objetos creados y otras acciones, con el objetivo de detectar fallas y observar su comportamiento.

2.1.6 Herramientas CASE

La principal herramienta utilizada para el modelado en el proyecto fue Visual Paradigm en su versión 5.0. Esta es una aplicación que soporta el trabajo con UML y otros lenguajes de descripción de sistemas. Entre sus principales funcionalidades pueden mencionarse la fácil confección y manejo de una amplia gama de diagramas ingenieriles, que permiten la documentación de cualquier software orientado a objetos [30]. Como funciones adicionales, provee la capacidad de generar reportes y posibilita la reingeniería del código.

2.2 Elaboración y diseño del sistema

En el presente epígrafe se documenta el proceso de diseño de la aplicación basado en la metodología OpenUP, específicamente describiendo una síntesis de los artefactos generados en sus dos primeras fases: Concepción y Elaboración, las cuales tienen como objetivo principal capturar las necesidades del cliente y elaborar la arquitectura del sistema.

2.2.1 Levantamiento de requisitos de software

La ingeniería de requisitos facilita el mecanismo apropiado para comprender lo que quiere el cliente, analizando necesidades, confirmando su viabilidad, y negociando una solución que razonablemente cubra las expectativas de los mismos [31].

Utilizando la entrevista como técnica principal de recopilación de información, se realizaron dos encuentros con los principales clientes y *stakeholders*[17] del proyecto, para así conocer su perspectiva funcional del sistema. Posteriormente se llevó a cabo un proceso de análisis y correspondencia de los

17 Algunas metodologías de desarrollo definen el término "stakeholder", como cualquier persona interesada en el proyecto de forma general. Por ejemplo, se incluyen en esta categoría: los clientes, usuarios del sistema y otros.

resultados de dicha entrevista y aquellos arrojados por el estudio de arte. El procedimiento anterior permitió obtener las necesidades reales del proyecto Nova en la actualidad y negociar las funcionalidades futuras del software. En la etapa final se elaboró el documento de descripción de requisitos de software, que luego de un período de revisión y cambio, fue aprobado por los clientes e involucrados.

2.2.2 Modelo de casos de uso del sistema

Una vez recopilados los requisitos, fue conformada la primera vista externa del sistema mediante el modelo de casos de uso, como se muestra en la Figura 3. Esta etapa del proceso permitió identificar los principales actores que interactúan con el sistema (ver Tabla 1), además de agrupar los requisitos en fragmentos funcionales de la aplicación que respondan a las peticiones de un actor determinado.

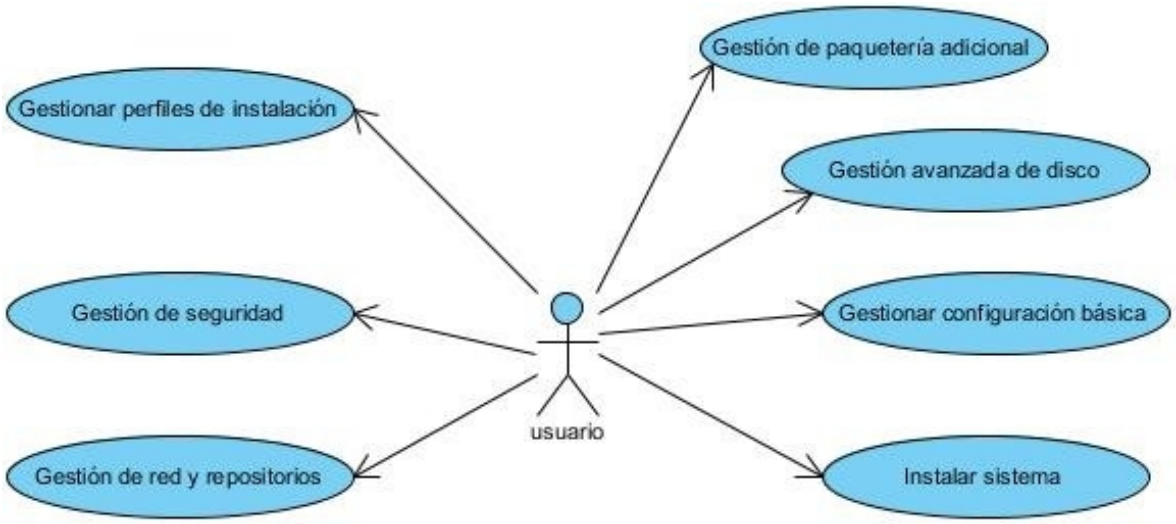


Fig 3: Diagrama de casos de uso del sistema

Actor	Descripción
Usuario	Interactúa con la aplicación para lograr la instalación y configuración del sistema.

Tabla 1: Actores del sistema

2.2.3 Descripción de casos de uso del sistema

La descripción de los casos de uso constituye una actividad fundamental para obtener un mayor entendimiento sobre los procesos que estos representan y a la vez comprender a profundidad la interacción de cada usuario con el sistema.

Cada escenario fue descrito en el documento de especificación de casos de uso del sistema en formato expandido [32]. Sin embargo, para los propósitos del presente documento, es suficiente la descripción de estos en formato de alto nivel [32].

Aunque en este epígrafe son descritos la totalidad de casos de uso del sistema, solo son arquitectónicamente significativos cuatro de los mismos (CUS2, CUS3, CUS4 y CUS6) que por ende, son aquellos que engloban las funcionalidades propuestas al alcance de esta investigación. Los restantes forman parte del flujo básico de configuración e instalación y son heredados de la versión previa de Serere.

CUS 1. Gestionar configuración básica

Efectúa la configuración básica requerida para la instalación satisfactoria del sistema operativo Nova

GNU/Linux. Es iniciado por el actor *Usuario,* que en este caso indica las opciones de su preferencia para la configuración del sistema.

- **Sección 1. Especificar zona horaria.** El sistema muestra una lista de zonas horarias y el actor indica en cuál de ellas se encuentra.

- **Sección 2. Indicar idioma y distribución de teclado.** El sistema muestra una lista de los idiomas disponibles, así como una lista de distribuciones de teclado. El actor indica qué idioma desea para la instalación y qué tipo de teclado está utilizando en la misma.

- **Sección 3. Especificar puntos de montaje.** El sistema muestra las unidades de disco disponibles, el usuario selecciona una de ellas e indica al menos un punto de montaje (raíz) para alguna de las particiones de esa unidad.

- **Sección 4. Indicar información básica de usuario.** El usuario indica su nombre completo, usuario, contraseña y confirmación de contraseña. El sistema valida los datos y los guarda si son correctos. En caso contrario se muestra un error.

CUS 2. Gestionar paquetería adicional

Lleva a cabo la instalación de un conjunto de paquetes marcados desde un repositorio previamente configurado. El caso de uso es inicializado por el actor Usuario. El sistema muestra una lista con la caché de paquetes del repositorio seleccionado agrupados por categorías, permitiendo al usuario la selección de un grupo de los mismos. Al ser instalado el sistema, Nova instalará la selección adicional de paquetes junto con sus dependencias.

CUS 3. Gestionar perfiles de instalación

Manipula los valores de una instalación específica mediante un archivo de perfil XML. El caso de uso es

iniciado por el actor *Usuario,* quien elige cargar un perfil para usarlo en una instalación, o generar uno para su posterior uso.

- **Sección 1. Cargar un perfil de instalación.** El sistema muestra la opción de cargar un perfil y usarlo en la instalación actual. El mismo puede añadirse de los predeterminados del instalador o de otro cualquiera, presente en cualquier dispositivo extraíble. El sistema realiza una verificación de la validez del perfil y en caso de ser correcto prosigue con el proceso. En caso contrario se informa al usuario.

- **Sección 2. Generar un perfil de instalación.** Al final de cualquier instalación, Serere da la opción de generar un perfil en su última vista. Un perfil no es más que un archivo XML con toda la información necesaria para replicar la instalación previa. En caso de indicarlo, el sistema construye el perfil y da la posibilidad de almacenarlo en cualquier medio.

CUS 4. Configuración avanzada de disco

Da la posibilidad al usuario de editar la tabla de particiones de una unidad de disco señalada, realizando varias configuraciones en la misma. El caso de uso es iniciado por el actor Usuario y termina una vez que el mismo hace clic en el botón de aplicar los cambios al tiempo que comienza el sistema a efectuar las acciones indicadas.

Sección 1. Crear una partición primaria. Una vez que el usuario selecciona un dispositivo, si hay espacio disponible en el mismo, el usuario puede crear una nueva partición. Se indica el sistema de archivos, tamaño, etiqueta, formato y cifrado (de manera opcional). El sistema guarda las preferencias y las aplica al terminar el paso.

Sección 2. Eliminar una partición primaria. El usuario selecciona una partición existente para eliminar. El sistema guarda las preferencias para liberar el espacio correspondiente a la partición

luego de terminar el paso.

Sección 3. Editar una partición primaria. El usuario selecciona una partición existente para su edición. El sistema muestra las características de la partición en un cuadro de diálogo con la posibilidad de ser cambiadas. Al terminar el paso el instalador aplica los cambios a la partición previamente seleccionada.

Sección 4. Crear una partición extendida. El sistema muestra el mayor bloque de espacio libre de la unidad seleccionada para crear la partición. El usuario indica el tamaño para la misma. El instalador guarda las preferencias para ser aplicadas al final del paso.

Sección 5. Eliminar una partición extendida. Igual a la *Sección 2*, solo que en este caso se trata de una partición extendida existente.

Sección 6. Editar una partición extendida. Igual a la *Sección 3*, pero en este caso referido a una partición extendida ya existente.

Sección 7. Crear una partición lógica. El usuario selecciona una partición extendida existente y en caso de que haya espacio disponible en la misma, se muestra un diálogo con las características de la nueva partición. El usuario indica el sistema de archivos, tamaño, etiqueta, formato y cifrado (de manera opcional). El sistema verifica las restricciones y si no hay errores, guarda las preferencias para aplicarlas al final del paso.

Sección 8. Eliminar una partición lógica. El usuario selecciona una partición lógica creada previamente en una de las particiones extendidas del disco. El sistema guarda los cambios para liberar el espacio de la partición al final del paso.

Sección 9. Editar una partición lógica. El usuario selecciona una partición lógica creada previamente en una de las particiones extendidas del disco. El sistema muestra un diálogo con las

propiedades de la partición que pueden ser editadas. Luego de que el usuario realice los cambios pertinentes, el sistema guarda los mismos para aplicarlos al final del paso.

Sección 10. Crear un nuevo dispositivo RAID. El usuario elije la creación de un dispositivo RAID nuevo. El sistema muestra un diálogo en el cuál se indica el sistema de ficheros, el nivel, los miembros, y el cifrado (opcional). Al final del paso, el sistema creará un nuevo dispositivo con las propiedades señaladas.

Sección 11. Eliminar un dispositivo RAID. El usuario selecciona un dispositivo RAID existente. El sistema elimina el dispositivo al aplicar las preferencias al concluir el paso.

Sección 12. Editar un dispositivo RAID. El usuario selecciona un dispositivo RAID existente y el sistema despliega sus características en un cuadro de diálogo. Se realizan los cambios necesarios, que son aplicados al terminar el paso.

Sección 13. Crear un grupo de volúmenes LVM. El sistema muestra un diálogo donde el usuario indica el nombre, extensión física y volúmenes físicos a utilizar en el grupo. El sistema guarda los cambios y los aplica al terminar el paso.

Sección 14. Eliminar un grupo de volúmenes LVM. El usuario selecciona un grupo de volúmenes a eliminar. El sistema aplica los cambios al terminar el paso.

Sección 15. Editar un grupo de volúmenes LVM. El sistema muestra un diálogo con las características del grupo, en el cual el usuario realiza los cambios pertinentes. Las preferencias son salvadas por el sistema hasta aplicarlas en la conclusión del paso.

Sección 16. Crear un volumen lógico LVM. El usuario elige la creación de un volumen lógico sobre un grupo de volúmenes LVM existente. Los cambios son aplicados al final del paso.

Sección 17. Eliminar un volumen lógico LVM. El usuario selecciona un volumen lógico existente de los mostrados por el sistema. El instalador almacena los cambios hasta aplicarlos al final del paso.

CUS 5. Gestionar opciones de seguridad

Tramita las características referentes a la seguridad del sistema, para asegurar que el mismo pueda cumplir desde sus inicios con los estándares de seguridad requeridos. El caso de uso es inicializado por el actor Usuario cuando el mismo arriba a la vista de seguridad del sistema. Todas las opciones de configuración son opcionales y conducen a las siguientes secciones.

Sección 1. Seguridad del GRUB. El usuario pulsa el botón de seguridad del GRUB. El sistema despliega un diálogo en el que el usuario introduce una contraseña que es validada por el instalador y si es correcta, es aplicada como contraseña del GRUB al terminar el paso. En caso de no cumplir con los estándares, se notifica al usuario.

Sección 2. Instalación de firewall. El sistema muestra un mensaje preguntando al usuario si desea instalar el firewall por defecto en el sistema y guarda la respuesta del usuario para aplicarla al concluir este paso.

Sección 3. Integración a un dominio. El usuario pulsa el botón de esta característica. El sistema muestra un diálogo con los requerimientos pertinentes para esta tarea. Luego de ser introducidos por el usuario, son validados por el instalador y almacenados para su aplicación al terminar el paso.

CUS 6. Gestión de red y repositorios

Efectuar la configuración básica de red y repositorios que garantice el acceso a fuentes de paquetería alternativos para usar en el proceso de instalación.

- **Sección 1. Configuración de red.** El sistema muestra un diálogo donde se pide información necesaria para una configuración básica inicial de red. El usuario provee entonces la dirección IP, máscara de subred y pasarela y acepta los cambios, que en caso de ser correctos serán aplicados al cerrar el diálogo.
- **Sección 2. Agregar un repositorio.** El usuario introduce una nueva dirección HTTP que es analizada por el sistema. Luego de terminado el proceso, se muestra una lista con los repositorios de Nova encontrados en dicha dirección. El usuario debe marcar al menos uno de ellos para agregarlo al sistema. Este cambio será aplicado al concluir el paso.

CUS 7. Instalar el sistema

En este caso de uso ocurre la descompresión, copia y posteriormente la configuración del sistema mediante los parámetros obtenidos a lo largo de toda la instalación. El caso de uso es iniciado por el actor *Usuario* cuando el mismo presiona el botón de finalizar la instalación y es direccionado a la vista de progreso del asistente.

2.2.4 Modelo de diseño del sistema

El siguiente paso una vez definido el alcance y realizado el análisis de los casos de uso del sistema, es conformar el modelo de clases de diseño, utilizando una arquitectura robusta que garantice la efectividad de la solución y otros elementos importantes como la reutilización y la flexibilidad.

En su versión anterior, Serere utilizó una arquitectura basada en capas, que garantizó el bajo acoplamiento de las clases y la posibilidad de extensión de la aplicación de manera eficaz. Para esta nueva liberación del software, se mantiene con toda intención este diseño estructural basado en un estilo de llamada-retorno [33]. Sin embargo, a fin de corregir algunas de las deficiencias encontradas en las etapas de la investigación, se implementarán los siguientes cambios al diseño original:

- Inclusión de una clase que gestione la interacción del instalador con procesos que se ejecuten en segundo plano, mediante un patrón Observador [34].

- Rediseñar el concepto de la clase “Wizard”, implementando un TDA Grafo [35] como estructura de datos principal, así como hacerla totalmente independiente de las interfaces de usuario.

El modelo de diseño del sistema recoge las realizaciones de la totalidad de los casos de uso reflejados en el epígrafe 2.2.3. Serán reflejadas en este documento sólo las realizaciones de los casos de uso críticos que dan paso al cumplimiento de los objetivos de la investigación, prescindiendo de las funcionalidades del sistema cuyos cambios no hayan sido arquitectónicamente significativos para esta versión de Serere.

Gestión de red y repositorios

El cumplimiento de las funcionalidades descritas en la segunda sección del CUS 6, se logra mediante la colaboración de las clases de la Figura 4.

La interfaz principal del módulo hacia el usuario se provee a través de la clase “RepoManager”. La misma permite la interacción con el archivo “sources.list” del sistema, cuya dirección es almacenada en la variable estática “R_SOURCE”. Además, esta clase brinda la posibilidad de añadir nuevos repositorios de forma manual mediante su método “add_new”, o utilizando la función pública “explore” de su atributo “__finder”. Siendo este último una instancia de la clase “RepoFinder”, cuyo objetivo principal es realizar una búsqueda en profundidad en una o varias direcciones suministradas como parámetros, y extraer de las mismas los repositorios cuyo origen se corresponda con el contenido de la variable “ORIGIN_MATCH”. En todos los casos (excluyendo el momento de la escritura en el archivo “sources.list”), los resultados son devueltos y tratados en términos de objetos de la clase “Repository”, que provee un objeto contenedor simple que de manera abstracta almacena los datos de un repositorio.

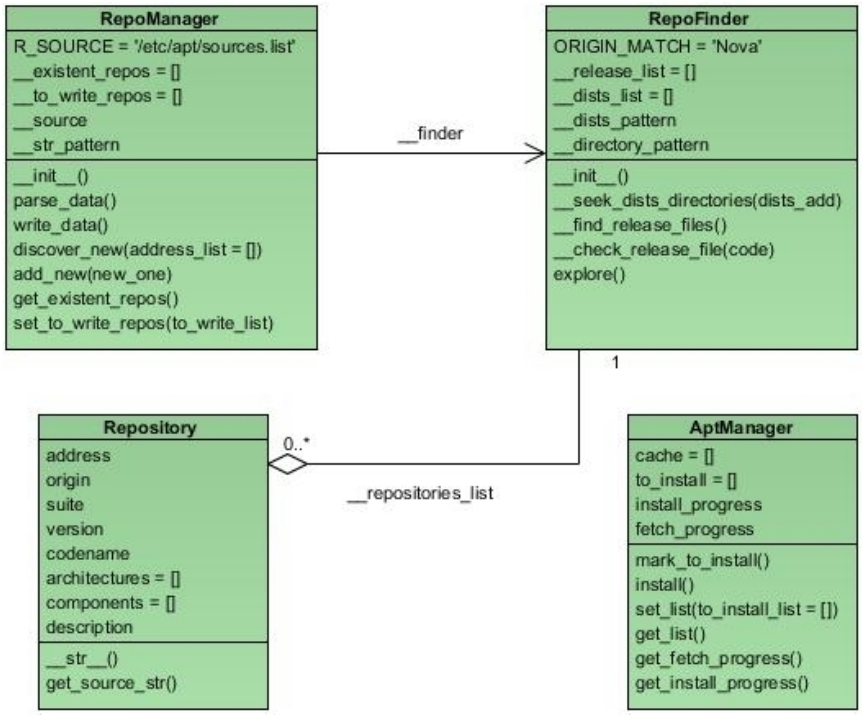


Fig 4. Colaboración de clases. CUS 6. Sección 2.

Para la sección 1 del mismo caso de uso, se encapsularon las funcionalidades necesarias en la clase “Interface” mostrada en la Figura 5. Al momento de crear un objeto de la misma es necesario suministrar de manera obligatoria el nombre del dispositivo de red (eth0, eth1, wlan0, etc), y de manera opcional los restantes parámetros (dirección IP, máscara de subred y pasarela). La función privada “__apply_config”, aplica los cambios al sistema en dos momentos: cada vez que una instancia de la clase es creada y cuando se realizan llamadas a las funciones “set_ip4_add”, “set_netmask” y “add_gateway”. Se utilizará el servicio DHCP cuando se suministre el valor “None” al campo “__ip4_add” en el constructor, o se utilice el método “set_ip4_add” con un valor nulo como argumento.

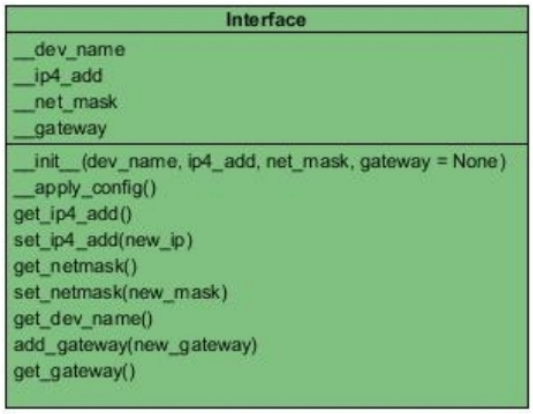


Fig 5.Colaboración de clases.CUS 6.Sección 1.

Gestión de paquetería adicional

Dadas las muchas facilidades de las distribuciones basadas en Debian para la gestión de paquetes, la clase “AptManager” mostrada en la Figura 6 funciona como fachada entre APT [36] y Serere.

Para satisfacer los requerimientos incluidos en este caso de uso, un objeto de esta clase posee el atributo “to_install”, una lista donde se almacenará la selección de paquetes candidatos a instalar. Dicho atributo puede ser modificado a través del método “set_list”. Una vez seleccionados los paquetes, el sistema debe indicar a APT que estos sean marcados para instalación, para lo que “AptManager” suministra el método “mark_to_install”, cuya función principal es chequear si los paquetes seleccionados no se encuentran instalados en el sistema y en caso de suceder, marcar también sus dependencias. Finalmente la función “install” ejecuta el proceso de descarga e instalación de los paquetes, que puede ser consultado mediante una llamada a los métodos “get_fetch_progress” y “get_install_progress” respectivamente.

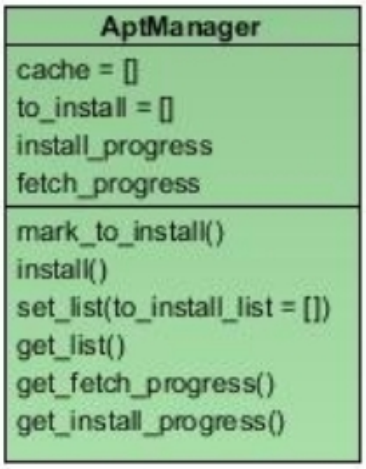


Fig 6. Colaboración de clases.

CUS 2.

Gestión de perfiles de instalación

La colaboración de clases de la Figura 7, resuelve los requisitos contenidos en la sección 1 del caso de uso número tres. La clase “ProfileLexer” es la encargada de una vez cargado un perfil de instalación empleando la función “load_profile”, y además establecidos todos los *tokens* posibles mediante una llamada a “set_tokens_struture”, reconocer todo el conjunto de tokens presentes en el archivo XML del perfil. Por su parte la clase “ProfileParser”, obtiene la lista de tokens presentes en el archivo de perfil mediante la función “scan” de su variable de instancia “__scanner”. Dicha clase posee además los atributos privados “__wiz_reference”, “__widgets_map” y “__grammar”. La referencia a todo el asistente, un diccionario para enlazar valores de los tokens a componentes visuales del instalador, y la gramática de un archivo XML de perfil respectivamente. El método “parse_profile” hace corresponder el listado de tokens devueltos por el objeto “__scanner” y la gramática definida para un perfil XML, devolviendo verdadero o falso en dependencia de la correctitud del perfil analizado. Una vez que el perfil ha sido

comprobado, la entidad cliente finamente puede realizar una llamada a la función “map_profile”, que introduce todos los valores predeterminados en el perfil directamente al asistente.

Para las operaciones de la sección 2 del caso de uso, solo es necesaria una llamada a la función “unmap_profile”, que devuelve un archivo XML con los valores actuales de los componentes visuales del instalador.

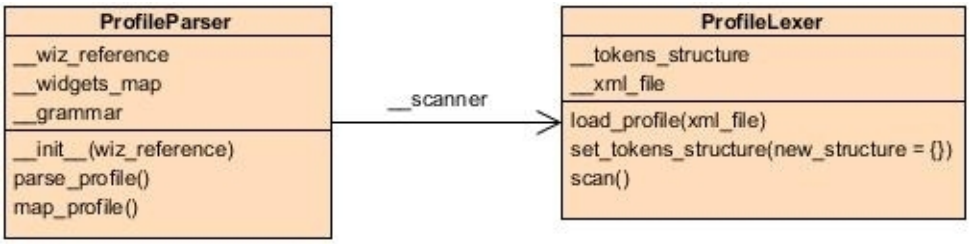


Fig 7. Colaboración de clases. CUS 3.

Gestión avanzada de discos

Es el caso de uso más complejo arquitectónicamente del software, ya que para lograr su objetivo requiere la colaboración de varias clases y bibliotecas del sistema. Su funcionamiento se centra en la colaboración de clases mostrada en la Figura 8, donde la clase “HDManager” mediante la función “load”, realiza un reconocimiento de las unidades físicas de disco conectadas al sistema, y ofrece una serie de funcionalidades para el trabajo con las mismas. Las clases “Disk” y “Partition”, son fachadas que simplifican y facilitan las funcionalidades del subsistema “python-parted”, una librería utilizada por el lenguaje para el manejo de unidades de disco y sus particiones. Entre su amplia gama de operaciones se encuentran las relacionadas con la gestión de dispositivos RAID y LVM.

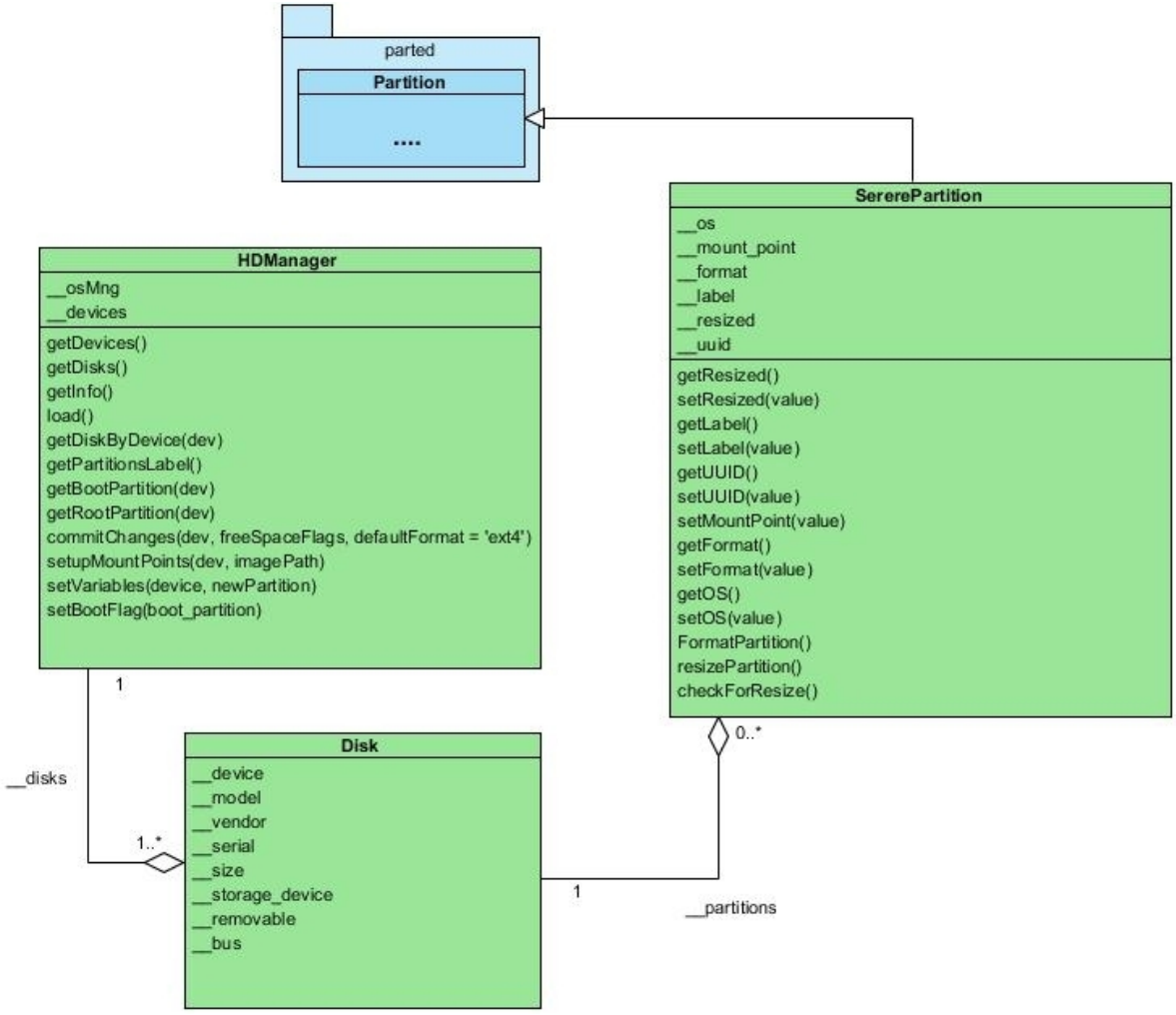


Fig 8. Colaboración de clases. CUS 4.

2.2.5 Patrones de diseño utilizados

Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo del software. Los patrones proponen una forma de reutilizar la experiencia de los desarrolladores, y para esto clasifican y explican formas de resolver situaciones que ocurren de manera frecuente a lo largo el ciclo de desarrollo [38].

Durante la realización del análisis y diseño de Serere, fueron identificadas algunas áreas del modelo en las cuales la aplicación de patrones contribuyó a optimizar su funcionamiento, y además, mitigar fallas que pudieron traducirse en factores que atentasen contra una arquitectura eficiente.

Patrón Observador

Un efecto lateral muy frecuente en aquellos sistemas que se fragmentan en un conjunto de clases que cooperan es la necesidad de mantener la consistencia entre los distintos objetos interrelacionados. Para no recurrir a soluciones fuertemente acopladas (que reducen la posibilidad de reutilización), este patrón define una dependencia “uno-a- muchos” entre objetos, para que, cuando uno de ellos cambie su estado, todos los que dependan de él sean avisados y puedan actualizarse convenientemente [39].

Patrón Fachada

Proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, simplifica su uso. La “fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas. Se utiliza principalmente cuando se pretende proporcionar una interfaz simple para un subsistema complejo [39].

Patrón Intérprete

Dado un lenguaje, define una representación para su gramática junto con un intérprete que usa dicha representación para interpretar sentencias en ese lenguaje. Se utiliza principalmente cuando hay un lenguaje que interpretar y sus diferentes construcciones pueden representarse mediante árboles sintácticos abstractos. Da sus mejores resultados cuando la gramática es simple. [39]

Capítulo 3 Implementación y prueba del sistema

En la implementación se comienza a desarrollar el sistema en términos de componentes, tomando como punto de partida los resultados obtenidos del diseño. Es objetivo fundamental de este flujo convertir las clases y subsistemas identificados en el diseño en ficheros y componentes que contengan código fuente y que luego puedan ser integrados y probados.

3.1 Diagrama de componentes del sistema

El presente epígrafe ilustra la organización de los componentes del sistema de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación, así como las dependencias existentes entre estos. Con respecto a la versión anterior de Serere, esta taxonomía ha sido mejorada en cuanto al nivel de acoplamiento de los módulos y la reestructuración de los mismos, ofreciendo mayor claridad en la representación física del sistema y aumentando considerablemente la cohesión entre los componentes.

Componentes principales de negocio

A fin de mantener consistencia entre la arquitectura y representación del sistema, fueron situados los componentes principales de la capa de negocio en el paquete *control,* así como se muestra en la Figura 9. De esta manera, son contenidos en el mismo los siguientes subsistemas:

- **system_manager:** aloja los módulos referentes a las operaciones del negocio que tienen correspondencia directa con el sistema. Entre ellas la gestión de discos, el manejo de hilos y procesos, gestión de red, etc.

- **config_manager:** módulos que procesan la información de configuración del instalador. Brinda soporte al trabajo con archivos de configuración y trazas de Serere.

- **package_manager:** módulos relacionados con la gestión de paquetería adicional del instalador y en correspondencia, la gestión de repositorios.

- **install_manager:** manejo de la copia e instauración del sistema en el ordenador. Se incluye en este el módulo de gestión de perfiles.

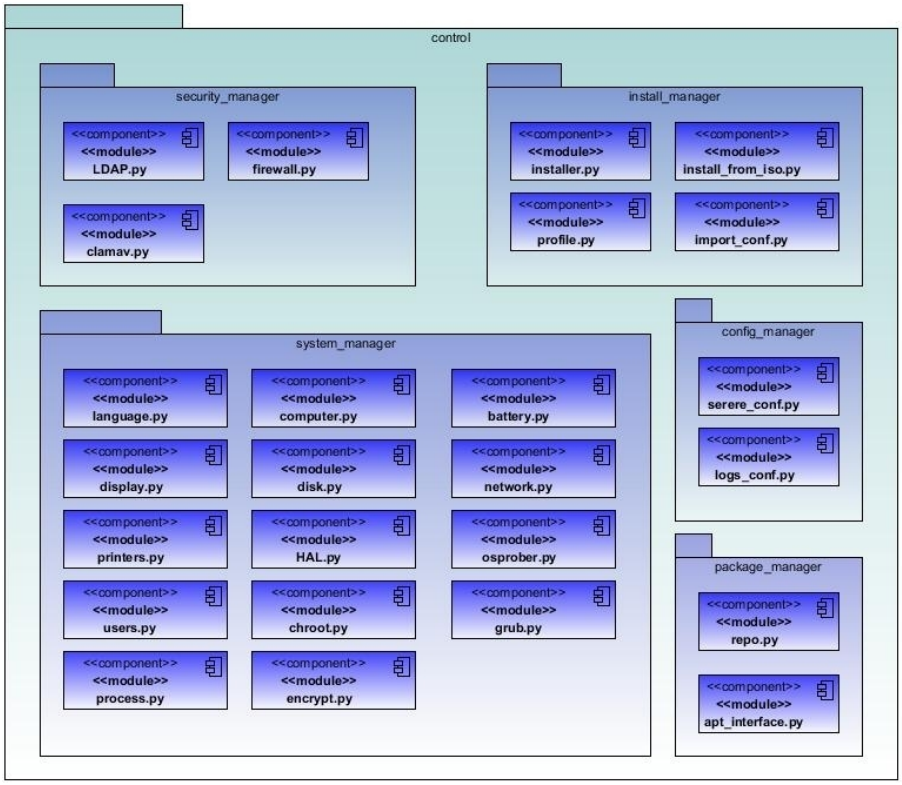


Fig 9. Diagrama de componentes. Nivel de negocio.

Componentes de interfaz

Fueron reunidos en el paquete 'ui' (Figura 10) los componentes relacionados con la interfaz visual de la aplicación. Al ser Serere una solución utilizada para varias personalizaciones de un mismo sistema operativo, necesita ser funcional en varios entornos de desarrollo y bajo ciertas condiciones de rendimiento. Por ello es necesario un alto grado de independencia entre los componentes de interfaz para evitar ambigüedades y facilitar la integración de los mismos con los subsistemas de negocio.

- **Base:** ofrece los componentes básicos para realizar una instalación en modo texto.
- **Curses:** un kit de elementos visuales en python-curses para desplegar un asistente de instalación ligero, rápido y más amigable que el modo Base.
- **Gnome:** los componentes visuales de la interfaz por defecto de Serere. Contiene los módulos visuales, recursos gráficos, plantillas, etc.

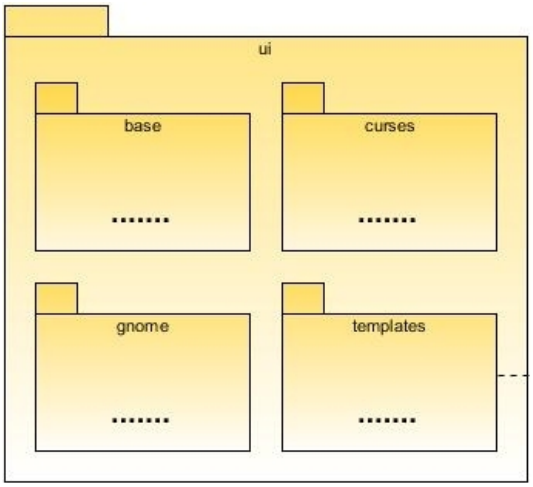


Fig 10. Diagrama de componentes. Nivel de interfaz de usuario.

Dependencias generales

De manera general, las dependencias entre los componentes del sistema fluyen de un modo unidireccional desde el nivel de negocio (paquete “control”) hacia los perfiles de instalación, y de manera bidireccional desde los componentes de interfaz hacia el nivel de negocio y otros recursos gráficos. Dicha colaboración es mostrada en la Figura 11.

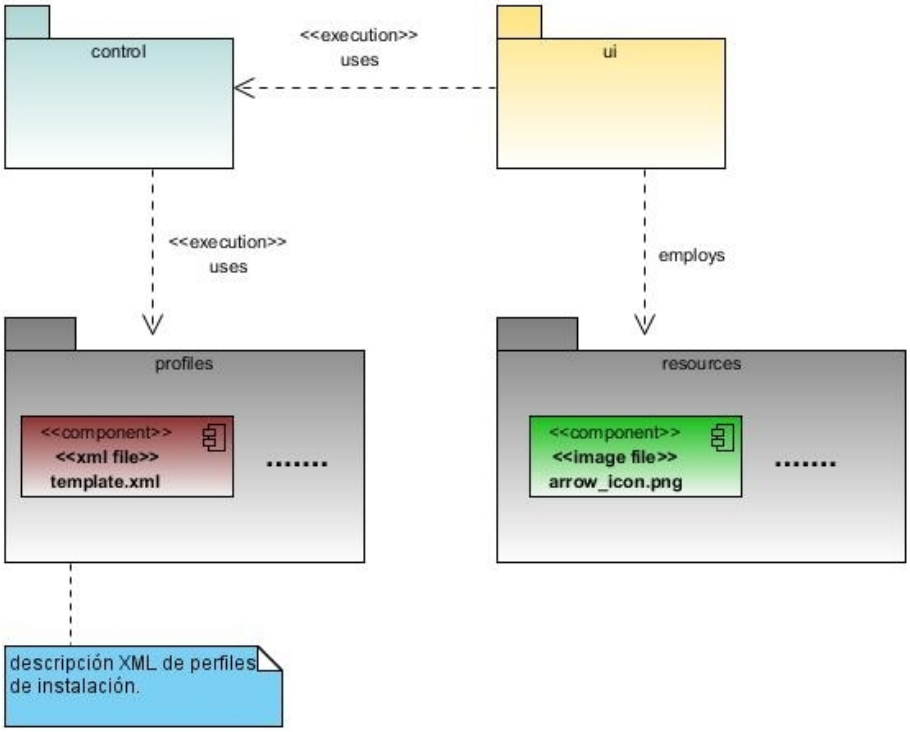


Fig 11. Representación general de componentes del sistema.

3.2 Dependencias externas del sistema

El uso de dependencias externas para complementar el trabajo de los componentes del sistema representados anteriormente, es un factor tenido en cuenta desde el análisis del estado del arte de esta investigación. El mismo propicia un nivel alto de reutilización y además minimiza de manera considerable el tiempo de desarrollo por concepto de implementación.

Dada la adición de nuevos módulos con responsabilidades que satisfagan los requerimientos de esta versión, ha sido necesario incluir otras librerías externas como dependencias del sistema, así como cambiar otras a favor de mejoras de rendimiento o soporte.

1. **Python-parted:** es una biblioteca del lenguaje python para el trabajo con *GNU Parted*, una herramienta empleada para el particionado de unidades físicas de disco con un sin número de funcionalidades. Este componente es utilizado por Serere desde su versión anterior, y en el caso de la actual, se extiende su uso para el tratamiento de unidades RAID y LVM.

2. **Python-apt:** una facilidad brindada por el lenguaje para el trabajo con *APT*. Esta potente biblioteca permite una interacción rápida y eficiente con la suite de gestión de paquetes por excelencia en distribuciones basadas en Debian y además, ofrece posibilidades adicionales como el monitoreo de progresos de descarga.

3. **Python-netifaces:** un componente sencillo que permite la obtención y el trabajo parcial de las interfaces de red conectadas al ordenador. Es utilizada en conjunto con otras bibliotecas del sistema para ofrecer un soporte de red básico desde el instalador.

3.3 Pruebas del sistema

Las pruebas de software son un elemento crítico para la garantía de calidad del software y representan una revisión final de las especificaciones, del diseño y de la codificación. Es el proceso de ejecución de un programa con la intención de descubrir errores [38]. Todo software debe probarse desde dos perspectivas diferentes: examinando la lógica interna del programa, y comprobando el cumplimiento de los requisitos del sistema. Para lograr este objetivo se llevan a cabo técnicas de diseño de casos de prueba de “caja blanca” y “caja negra” respectivamente.

En el presente epígrafe, presentamos los procedimientos empleados para la realización de pruebas a la versión cuarta de Serere una vez concluido su ciclo de desarrollo.

3.2.1 Pruebas unitarias

Las pruebas unitarias centran en proceso de verificación en la menor unidad del diseño del software: las clases y funcionalidades de los módulos. Es un tipo de prueba comúnmente sugerida como buena práctica, por su simplicidad de realización y además por brindar información de valor sobre el manejo de datos a este nivel [38]. Al ser una técnica de “caja blanca”, el objetivo de una prueba unitaria no es otro que garantizar que las funciones y objetos devuelvan salidas correctas bajo condiciones y conjuntos de entrada específicos. La comprobación del resultado esperado garantiza la falla o éxito de la prueba.

El módulo Unitest de python, permitió implementar pruebas unitarias a los métodos y clases de los caminos básicos de orden crítico de esta versión de Serere.

3.2.2 Partición equivalente

La partición equivalente es un método de “caja negra” que divide el campo de entrada de un programa en

clases de datos, de los que se pueden derivar casos de prueba. Un caso de prueba ideal en este método, descubre de forma inmediata una clase de errores. De esta forma reducimos el número total de casos de prueba a desarrollar [38].

El diseño de casos de prueba para este método se basa en una evaluación de las clases de equivalencia[18] para una condición de entrada. Típicamente esta última es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica.

En el documento descripción de casos de prueba, se muestra el diseño final de los casos de prueba generados a partir de este método para la validación y prueba final de los requisitos de Serere.

3.2.3 Ejecución y resultados

Luego de diseñadas las pruebas del producto se procedió a la implementación y ejecución de las mismas. Los resultados fueron obtenidos luego de dos iteraciones de ejecución guiadas por las actividades del flujo de Prueba de la metodología OpenUP.

La primera iteración arrojó como resultado un fallo del 31% del total de pruebas diseñadas, equivalente a un levantamiento de 13 no conformidades funcionales en el sistema. Luego de trabajar durante un período en la solución de las mismas y efectuar la segunda iteración, se logró el éxito de un 19% más de pruebas con respecto a la fase anterior, permaneciendo solo 6 no conformidades que fueron resueltas desde entonces hasta la terminación de la fase.

Tal como se muestra en la Figura 12, la resolución de los errores detectados en esta etapa permitió el aseguramiento de una mayor calidad en el producto.

18 Representa un conjunto de estados válidos o no válidos para condiciones de entrada.

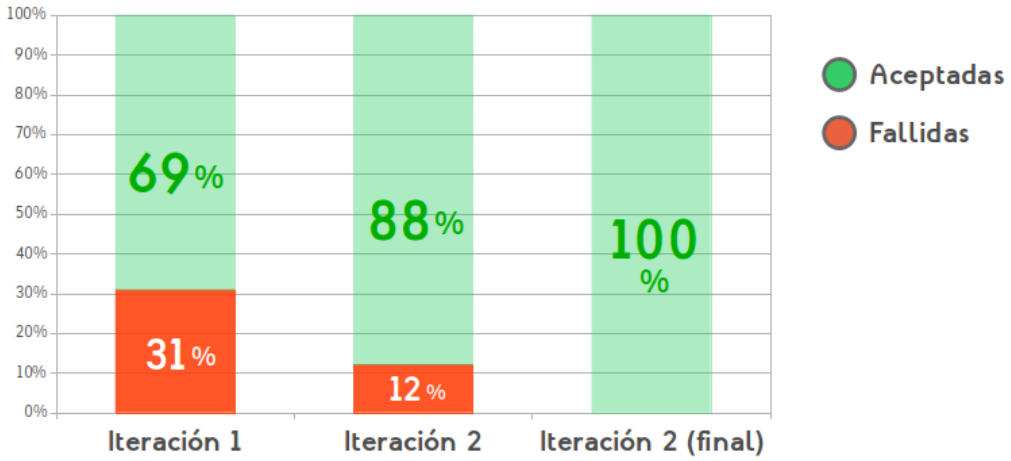


Fig 12. Porcentaje de aceptación y fallas de pruebas de Serere 3.0 por iteración

En la presente fase se logró la construcción de un producto funcional basado en el diseño y la arquitectura propuestos. Además fueron identificadas y corregidas las fallas existentes en el sistema.

Conclusiones

A la culminación de esta investigación y el cumplimiento de los objetivos planteados desde sus inicios, se arribaron a las siguientes conclusiones:

- El estudio de los instaladores pertenecientes a las distribuciones GNU/Linux más utilizadas, permitió identificar la necesidad existente de implementar una nueva versión de Serere que contara con un número mayor de funcionalidades.

- El cambio de la arquitectura del sistema y la adición de nuevas funcionalidades requeridas por el departamento, garantizaron la obtención de una herramienta que satisface el proceso de instalación de la distribución cubana Nova GNU/Linux.

- Las pruebas realizadas al producto implementado, demostraron que el mismo cumplía con los requerimientos definidos al inicio de la investigación y permitieron la corrección de errores encontrados.

Recomendaciones

- Implementar en próximas versiones una fábrica abstracta de interfaces para reducir el tiempo de desarrollo por concepto de diseño de las mismas.
- Incluir en próximas versiones el resto de las funcionalidades sugeridas en el estudio del arte que no fueron implementadas en esta versión.

Referencias Bibliográficas

1 Allan Pierra Fuentes. Nova, distribución cubana de GNU/Linux. Reestructuración estratégica de su proceso. octubre 2011. [8 noviembre 2011].

2 Yoandy Pérez Villazón. Metodología para la migración de software libre de las universidades del ministerio de educación superior. mayo 2008. [21 octubre 2011].

3 William Stallings. Operating Systems. Internals and design principles. 5 [EEUU]: ITec Books, febrero 2010 [8 noviembre 2011].

4 Raydel Miranda Gómez. Serere. Aplicación de instalación del sistema operativo GNU/Linux Nova. marzo 2000. [7 septiembre 2011]. Disponible en la Web: <http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=7917>.

5 Michel Hernández Rodríguez. GNU/NOVA: una distribución de software libre para Cuba. *Revista Giga.* febrero 2007. [12 octubre 2011].

6 Christopher Negus. *The Linux Bible*. 2009 Edition. Wiley Publishing, marzo 2009 [9 enero 2012].

7 Roberto Hernández Sampieri, Carlos Fernández Collado, y Pilar Baptista Lucio. *Metodología de la investigación*. 2 [México]: McGrawHill, año 1998 [21 agosto 2011].

8 Yunier Soler Franco. Serere 2.0. Instalador del sistema operativo GNU/Linux Nova. mayo 2010. [10 septiembre 2011]. Disponible en la Web: <http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=9347>.

9 Carlos Billy Reynoso. Introducción a la arquitectura de software. marzo 2004. [17 enero 2012].

Disponible en la Web: <http://willy.net>.

10 Andrew Hunt, y David Thomas. *The pragmatic programmer*. 1 Addison Wesley, octubre 1999 [18 enero 2012].

11 Brett D. MCLauglhin, Gary Pollice, y David West. *Head First. Object-oriented analysis and design*. 1 O'REILLY, noviembre 2006 [4 enero 2012].

12 Karel Riverón Escobar. La mejor distribución de Linux. *humanOS.uci.cu* julio 2010. [4 enero 2012]. Disponible en la Web: <http://blogs.prod.uci.cu/humanOS/2010/07/10/la-mejor-distribucion-de-linux/>.

13 Mark Spyce. *Ubuntu 11.04 Essentials*. 5 Payload Media, agosto 2011 [5 febrero 2012].

14 Anón. Step-by-step beginner's guide to installing Ubuntu 11.10 - TechSpot OpenBoards. *TechSpot* octubre 2007. [20 octubre 2011]. Disponible en la Web: <http://www.techspot.com/vb/newreply.php?do=newreply&noquote=1&p=1096690>.

15 Gerard Beekmans. *Linux From Scratch*. 2009 [9 enero 2012].

16 Andrew Hudson, y Paul Hudson. *Fedora Unleashed*. Pearson Education, marzo 2008 [17 febrero 2012] pág 7.

17 Marius Nestor. Installing Fedora 10. *Installing Fedora 10* diciembre 2010. [12 marzo 2012]. Dispoible en la Web: <http://news.softpedia.com/news/Installing-Fedora-10/>.

18 Narciso Bravo Tejeiras. Diversos enfoques para una instalación de GNU/Linux. *Revista Begins* febrero 2009, 68. [10 marzo 2012].

19 Andrew Hudson, y Paul Hudson. *Fedora Unleashed*. Pearson Education, marzo 2008 [17 febrero

2012] pág 24 y 25.

20 Novell team. Novell Software Solutions. [12 marzo 2012]. Disponible en la Web: <http://developer.novell.com/who-are-we/index.html>.

21 OpenSuse team. OpenSuse Project. *OSP*. [16 marzo 2012]. Disponible en la Web: <http://www.opensuse.org/en/>.

22 Eduardo Adolfo Sotomayor G. *OpenSuse 11.2. Guía ilustrada*. TekillerBooks, noviembre 2009 [14 marzo 2012].

23 PMI. *Project Management Book of Knolledge*. 4 [EUA]: Project Management Institute, marzo 2009 [12 febrero 2012].

24 James Shore. *The art of agile development*. O'REILLY, 2008 [15 enero 2011].

25 Alan Cameron Wills. Modeling in an Agile Context. *The Architecture Journal* septiembre 2011, 47. [3 marzo 2012]. Disponible en la Web: <http://www.architecturejournal.net> .

26 Dave Brueck, y Stephen Tanner. *Python 7. Bible.* 2 [New York]: Hungry Minds, febrero 2009 [2 febrero 2012].

27 Anón. 1. Generalidades — Inicio. [21 octubre 2011]. Disponible en la Web: <http://gespro-help.prod.uci.cu/introduction/#introduccion>.

28 Roger S. Pressman. *Ingeniería del Software: un enfoque práctico.* 5 [Madrid]: Félix Varela. La Habana., abril 2005 [4 diciembre 2011]. Disponible en la Web: <http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=3143>. Pag 151 y 152.

29 Milton Mazzarri. Herramientas libres para el apoyo en el proceso de desarrollo de software.

noviembre 2008. [12 abril 2012].

30 Leonard Jessen. *UML. A practice aproach.* 2 [EUA]: O'REILLY, marzo 2007 [16 febrero 2012].

31 Rick Lutowsky. *Software requirements*. 3 [Londres]: Auerbach Publications, abril 2009 [20 septiembre 2011].

32 Craig Larman. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* 1era, 1999. [México]: PEARSON, 1999 [2 octubre 2011]. Disponible en la Web: <http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=3222>.

33 Diomidis Spinellis, y Georgio Gousios. *Beautiful Architecture*. 1 [USA]: OREILLY, enero 2009 [6 marzo 2012].

34 Erick Freeman, y Elisabeth Freeman. *Head First. Design Patterns*. 1 [EEUU]: OREILLY, junio 2007 [6 enero 2012].

35 Peter Brass. *Data Structures*. [New York. USA.]: Cambridge University Press, 2008 [11 octubre 2011].

36 Martin F. Krafft. *Debian system concepts and techniques*. OpenSource Press, febrero 2005 [24 abril 2012]. Disponible en la Web: .

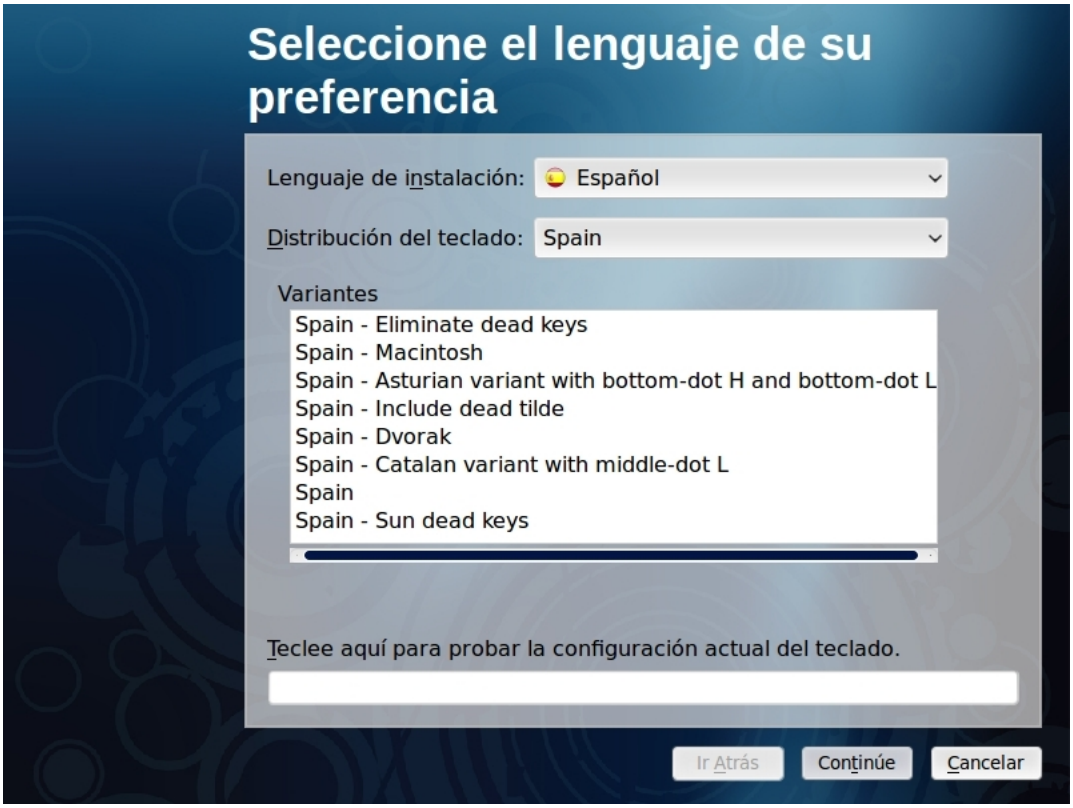
37 Marius Nestor. Installing Fedora 10 - Softpedia. *Softtpedia* diciembre 2010. [20 mayo 2012]. Disponible en la Web: <http://news.softpedia.com/newsTag/install+Fedora>.

38 Roger S. Pressman. *Ingeniería del Software: un enfoque práctico.* 5 [Madrid]: Félix Varela. La Habana., abril 2005 [4 diciembre 2011]. Disponible en la Web: <http://catalogoenlinea.uci.cu/cgi-bin/koha/opac-detail.pl?biblionumber=3143>.

39 Erich Gamma, Richard Helm, Ralph Johnson, y John Vlissides. *Design Patterns. Elements of reusable software-oriented software.* [USA]: Chinebolt Pubs. [12 noviembre 2011].

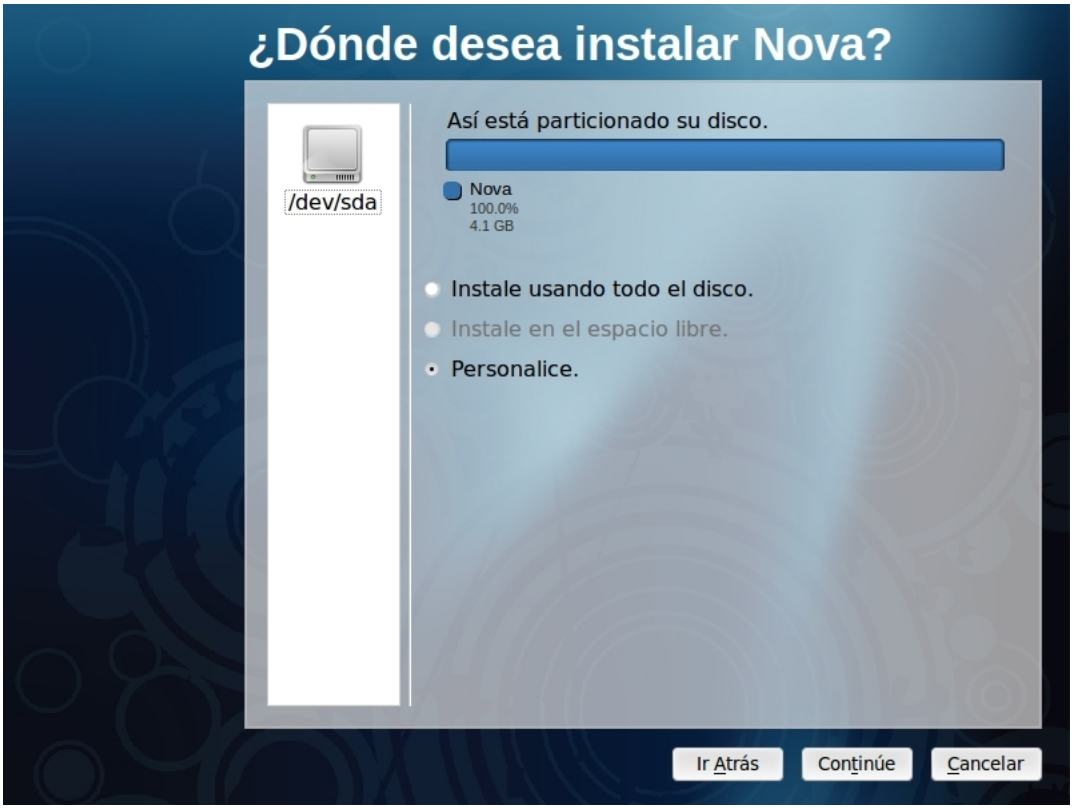
Anexos

Anexo 1. Vistas de Serere 2.0

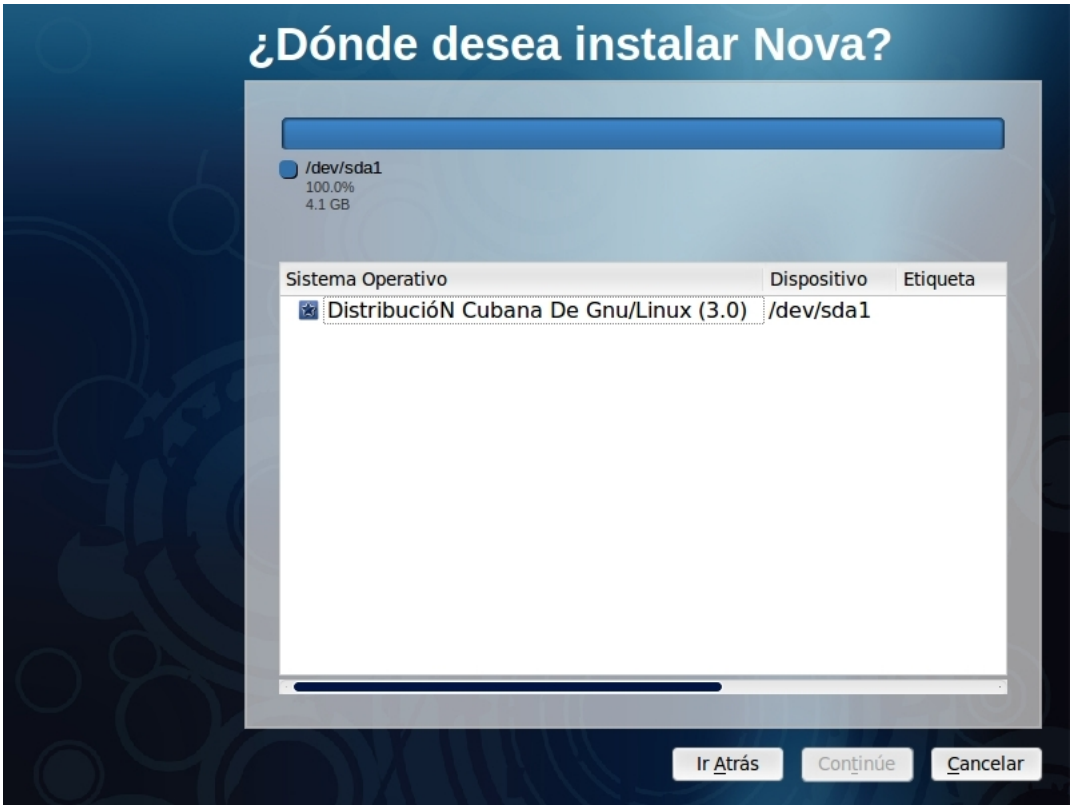


Vista 1. Selección de idioma y distribución de teclado.

Vista 2. Selección de zona horaria.

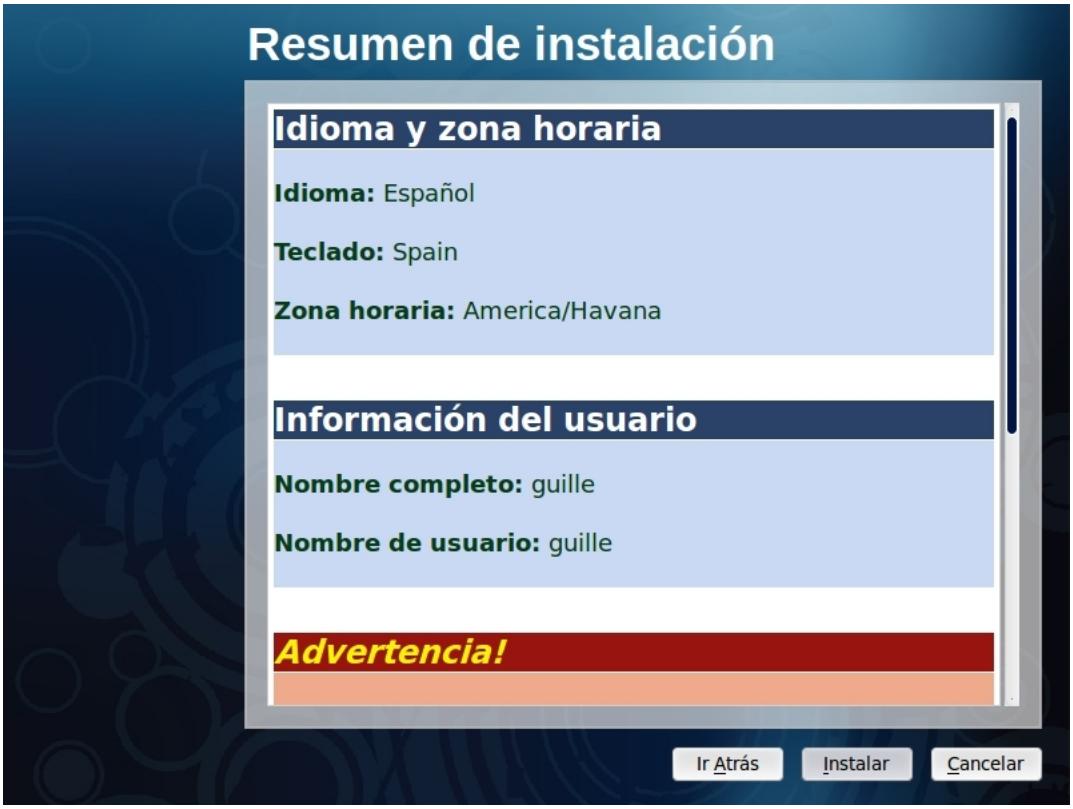


Vista 3. Seleccionar modo de instalación.

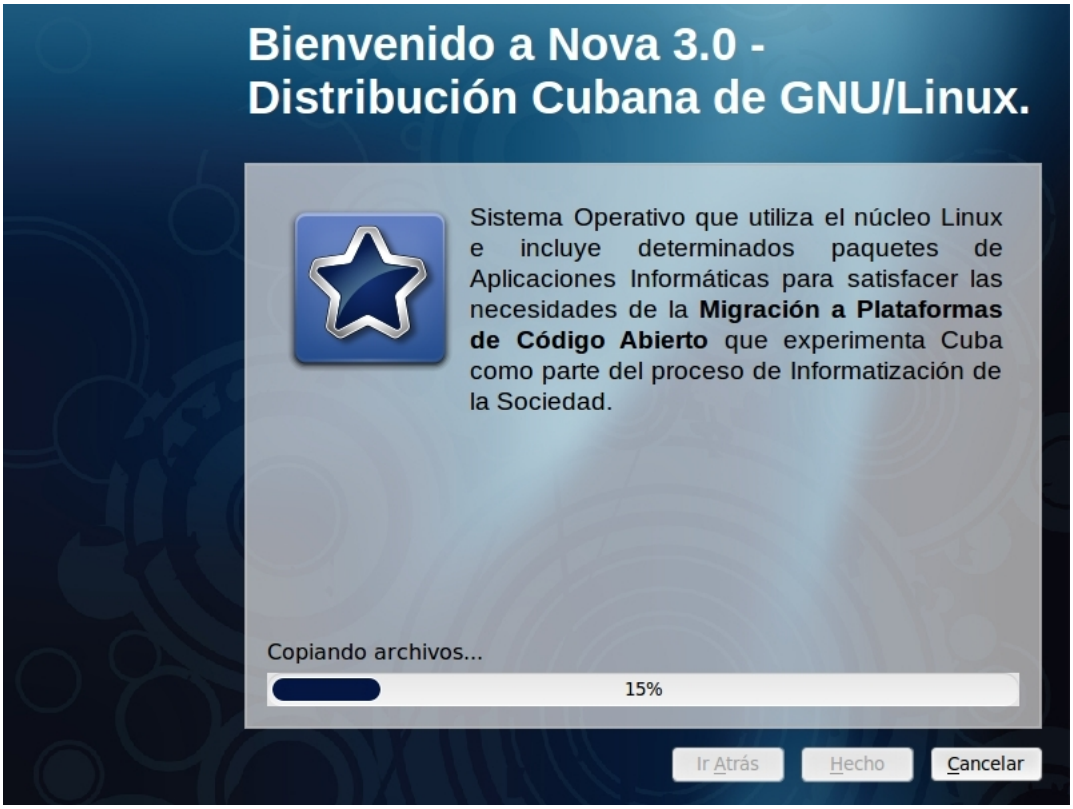


Vista 4. Establecer puntos de montaje.

Vista 5. Vista de preferencias personales.



Vista 6. Resumen de instalación.



Vista 8. Progreso de instalación.