Universidad de las Ciencias Informáticas

Título: Serere. Aplicación de instalación del sistema operativo GNU/Linux Nova.

Autor: Raydel Miranda Gómez.

Tutores: Msc. Héctor Rodríguez Figueredo, Ing. Allan Pierra Fuentes.

Presentado en opción al Título de Ingeniero en Ciencias Informáticas

Ciudad de La Habana, Cuba

abril, 2009

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y 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 _____.

Raydel Miranda Gómez	Msc. Héctor Rodríguez Figueredo	Ing. Allan Pierra Fuentes
_____	_____	_____
Firma del autor	Firma del tutor	Firma del tutor

Agradecimientos:

A mis tutores, a Ángel, por confiarme una tarea tan importante como el instalador de Nova, a todo el equipo del proyecto, por brindarme un ambiento de trabajo ameno, cordial y amistoso, a todas las personas que me rodean a diario en la Universidad de las Ciencias informáticas, de todos he aprendido mucho.

Dedicatoria

A mi madre que luchó tanto como yo en estos cinco años.

A mi abuelo, al que he podido visitar muy poco, pero siempre confió en mi ciegamente.

A la negrita más bonita de Cuba mi hermana.

A mi primo loco y amigo de San Agustín, Ledis.

A todo mi familia.

Resumen

En esta investigación se presenta el diseño e implementación de la base de un instalador (Serere) y el framework para la gestión de sus vistas (Arare), que constituyen una solución única para la instalación y personalización de cualquier línea de desarrollo del sistema operativo GNU/Linux Nova. Se utilizó la metodología ágil conocida como Programación Extrema y Eclipse como entorno de desarrollo para el lenguaje de programación *Python*. Este trabajo constituye un aporte al desarrollo de los sistemas operativos particularmente en el área de los instaladores personalizables.

PALABRAS CLAVE

Soluciones personalizadas, instalador, Serere, Arare, framework.

Tabla de contenido

Resumen **5**

Tabla de contenido **6**

Introducción **8**

Capítulo 1: Fundamentación del tema **12**

Introducción 12

Ubiquity 13

Anaconda *14*

InstallXS *14*

Yast *15*

Instalador de Gentoo *16*

Instalación de sistemas versus clonación de sistemas *16*

Tendencias tecnológicas *17*

Conclusiones 18

Capítulo 2: Entorno de desarrollo **19**

Introducción 19

Programación Extrema como metodología de desarrollo 21

Programación en parejas *21*

Programación "test-first" *21*

Constante interacción del cliente con el equipo. *21*

Propiedad colectiva del código. *21*

Arquitectura 22

Modelo *22*

Vista *22*

Controlador *22*

Plataforma de desarrollo 23

Lenguajes de programación 24

Gestión 25
Gestión de documentación *25*
Gestión de reportes *26*
Gestión de tareas *26*
Control de versiones *27*
Nova como sistema operativo base 28
Conclusiones 31
Capítulo 3: Descripción de la solución 32
Introducción 32
Base de Serere 33
Historia de usuario número 1: Selección del modo de instalación *33*
Historia de usuario número 2: Selección de zona horaria *36*
Historia de usuario número 3: Administración de discos duros *38*
Historia de usuario número 4: Configuración de red *41*
Historia de usuario número 5: Administración de usuarios *42*
Framework Arare 44
Historia de usuario número 1: Creación de nuevas vistas *44*
Historia de usuario número 2: Adición de nuevas vistas *44*
Historia de usuario número 3: Eliminación de vistas *45*
Conclusiones 46
Bibliografía 47
Anexos 49
Anexo 1: Diagramas de clases 49
Anexo 2: Fragmentos de código 53

Introducción

Cuba es un país consumidor de productos de software, dentro de los que se encuentran los denominados sistemas operativos. Un sistema operativo es, parafraseando a José Antonio Cortés del colegio de Miraflores (Cortés 1999): Un software de sistema, un conjunto de programas de computación destinados a realizar muchas tareas, entre las que destaca la administración eficaz de sus recursos. Se puede ampliar esta definición agregando que es el encargado de gestionar los recursos de la computadora ya sea periféricos u otros programas instalados y garantiza la comunicación hombre-máquina.

En el año 2007 el sistema operativo más usado en Cuba, era Windows XP (Valle 2007). En la actualidad, la mayoría de las computadoras del país funcionan aún con este sistema operativo u otra versión del producto (Windows 98, Windows Me (Millennium Edition), etcétera). El uso de dichos sistemas incurre en gastos elevados, pues los productos que usan son en su extensa mayoría de carácter privativo. Lo que significa, que debe pagarse una licencia para legalizar su uso. El soporte a dicho software es también muy costoso y requiere la implementación de una solución más económica.

Así surge la idea de construir un sistema operativo libre y de distribución gratuita, basado (en sus inicios) en la distribución Gentoo GNU/Linux. Este sistema sería implementado y mantenido en Cuba, logrando cierto grado de soberanía tecnológica y una reducción considerable de los gastos.

Esta idea, dio a luz el proyecto GNU/Linux Nova, una distribución libre que se implementa actualmente en la Universidad de las Ciencias Informáticas (UCI). El objetivo de trabajo de Nova es crear soluciones a la medida según los requerimientos del cliente que la solicita, incorporando al sistema sólo aquellas aplicaciones y características que completan dicho pedido.

Para desplegar las distintas soluciones en las estaciones de trabajo, se lleva a cabo un proceso de instalación. Un proceso de instalación de software, es: El proceso por el cual nuevos programas son transferidos al ordenador y posteriormente configurados para ser usados con el fin para el cual fueron desarrollados. Por lo que se puede definir el proceso de instalación de un sistema operativo como: El proceso por el cual se transfiere al ordenador el software necesario para hacer posible la comunicación

hombre-máquina y además proporcionar una base sobre la cual serán instalados los demás programas.

El programa encargado de guiar este proceso es el que se conoce con el nombre de “Instalador”. Aunque un instalador no es solo el software que guía el proceso. Es, además, todo el conjunto de librerías y archivos de configuración que posteriormente se instalarán en la máquina en conjunto con el ambiente sobre el cual corre el software que guía el proceso.

El proceso de instalación consta de una serie de pasos que no varían de un instalador a otro:

Administración de disco duro.

Paso mediante el cual el usuario particiona o divide el disco duro en las secciones necesarias para realizar la instalación del sistema operativo. Las distintas secciones en las que puede estar dividido el disco son llamadas particiones. Existe la posibilidad de que ya existan algunas particiones. Las mismas deben tener el tamaño requerido para la instalación del sistema, por lo que en este paso el usuario puede redimensionar una partición con dicho propósito.

Selección de zona horaria.

La selección de zona horaria permitirá al reloj del sistema operativo sincronizar con el uso horario de la región en que está teniendo lugar la instalación del sistema.

Administración y creación de usuarios.

En este punto se procede a la creación del usuario inicial. El primer usuario que tendrá nombre y palabra clave para su entrada al sistema operativo. Generalmente también en este momento se cambia o se asigna una contraseña para el usuario administrador o super usuario(root en el caso de los sistemas Unix).

Configuración de la red.

Paso en el que el usuario configura la o las distintas interfaces de red que con las que cuenta el ordenador. Con el objetivo de garantizar una correcta conexión después de instalado el sistema operativo.

La mayoría de lo pasos del proceso de instalación del sistema operativo GNU/Linux NOVA implican configurar y personalizar cada solución, lo que provoca cambios significativos en la interfaz visual y el código del instalador, siendo imprescindible en algunos casos una completa reescritura de la aplicación. Esta situación problémica conduce a plantear el siguiente **problema científico**: ¿Cómo lograr una única solución de instalación y configuración que no dependa de la línea de desarrollo de la distribución GNU/Linux NOVA?

En este trabajo se considera como **objeto de estudio** los *instaladores de algunos sistemas operativos*, sus principales características, ventajas y desventajas, para luego aplicar las experiencias obtenidas en **un campo de acción** limitado a la estructura base del instalador del sistema operativo Nova.

Se trazó como **objetivo principal** desarrollar de una aplicación para la instalación del sistema operativo GNU/Linux Nova.

Partiendo del objetivo principal, se plantean los siguientes **objetivos específicos**:

1. Realizar un estudio del estado del arte de los instaladores de algunos sistemas operativos.
2. Implementar la base del instalador.
3. Implementar el framework para la construcción de vistas de dicho instalador.

Esta investigación tiene como **idea a defender** que una aplicación de instalación del sistema operativo GNU/Linux Nova como solución única para sus diferentes líneas traerá consigo una mayor eficiencia en el proceso de desarrollo.

La **investigación** que se llevó a cabo con ese propósito fue dirigida por las **tareas**:

- Investigación sobre los instaladores más usados.
- Investigación y selección del lenguaje, tecnología y metodología a utilizar.
- Investigación y selección de patrones de arquitectura.
- Investigación y selección de patrones de diseño.

- Investigación acerca del diseño de pruebas de aceptación.

Los **métodos** usados fueron: **analítico-sintético**, que permitió la realización de análisis de los distintos instaladores, para una mejor comprensión de sus características. Se empleó además la **observación,** mediante la cual se detectó de manera directa la necesidad de un instalador flexible y que se pueda adaptar a los posibles cambios a los que se verán sometidas las distintas soluciones personalizadas del proyecto Nova.

Capítulo 1: Fundamentación del tema

Introducción

Existen numerosos instaladores. Se realizó el estudio de una selección, conformada por los que instalan los sistemas operativos más populares. De cada uno de ellos se mencionan las características que los distingue del resto.

También, se arribó a la conclusión que para la distribución de un sistema operativo, no es factible el método de clonación de sistemas. Éste solo es eficiente cuando el ambiente tiene determinadas características.

.

Ubiquity

El instalador de la serie Ubuntu no a sufrido grandes transformaciones en cuanto a su estructura desde sus inicios hasta la versión más actual (Ubuntu Hardy 8.10, en el momento en que se escribe el presente documento) .

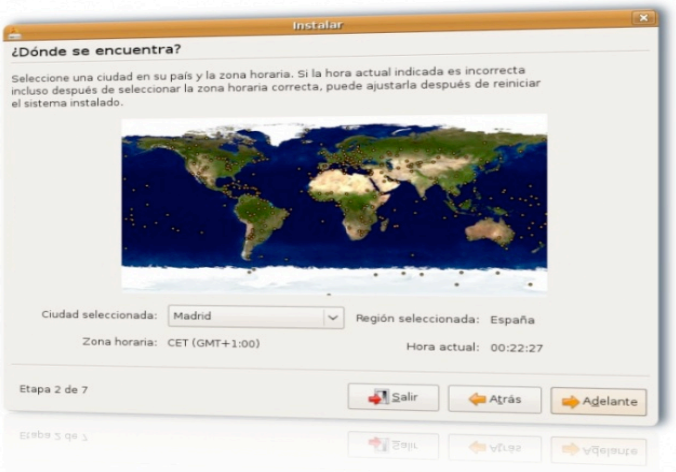

Figura 1: Una de las vistas de Ubiquity, Instalador de Ubuntu.

Dentro de sus características más notables se encuentran:

- **Amplia gama de reconocimiento de hardware y localizaciones.**

  La variedad de configuraciones de idioma que permite Ubiquity es una gran ventaja en cuanto a público se refiere, llegando a la mayoría de éste en su propio lenguaje. Lo mismo ocurre con el hardware, reconociendo las más diversas distribuciones de teclado incluyendo el soporte para teclados con tecnología Machintosh.

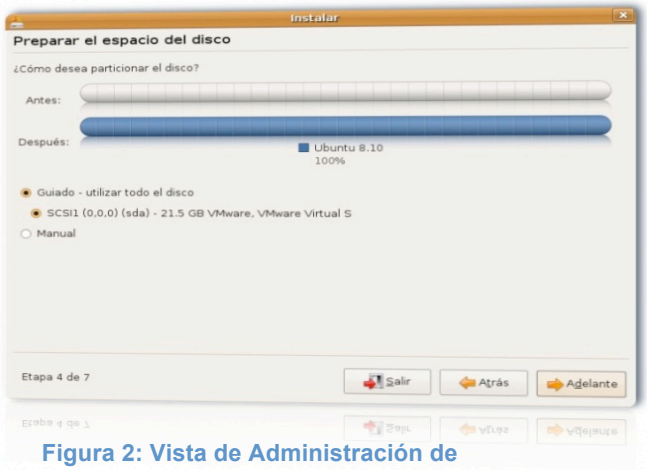

Figura 2: Vista de Administración de disco, Ubiquity.

- **Particionador o gestor de particiones propio.**

  La presencia de un particionador propio (fig 2) brinda a Ubiquity una gran independencia. En consecuencia, no es necesario correr otras aplicaciones para realizar su tarea.

Anaconda

Anaconda es el instalador de Red Hat Linux y Fedora y otros sistemas como Sabayon Linux. Desarrollado en Python y C con interfaz gráfica implementada en PyGtk. Para la interfaz en modo texto se hace uso de python-newt. Un módulo de python para el trabajo con las librerías newt(Package python-newt (0.52.2-11.3) 2009).

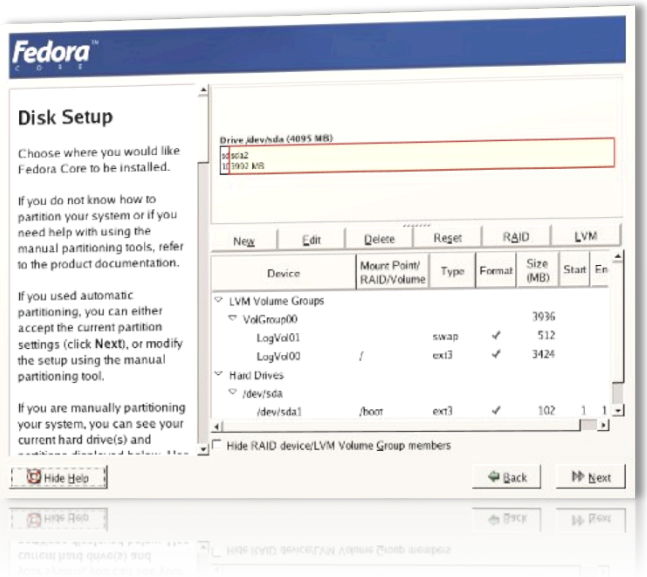

Figura 3: Gestor de particiones de Anaconda.

Sus numerosas opciones lo convierten en un instalador muy funcional. Permitiendo al usuario realizar acciones como: actualizar el sistema instalado, reparar el sistema instalado e incluso reparar o corregir la configuración del gestor de arranque.

Aunque aún no presenta un gestor de particiones visualmente agradable, el existente brinda todas las opciones necesarias para llevar a cabo la gestión de este tipo de dispositivo (fig. 3).

InstallXS

El instalador del sistema operativo GNU Ututo XS. Implementa dos interfaces una en modo texto y otra en modo gráfico. Ésta última no hace uso del servidor gráfico “X” ya que corre sobre framebuffer. Además, posee una interfaz web para comandar remotamente el instalador haciendo posible instalar el sistema desde otra computadora en la red local o Internet.

Una característica importante de este instalador es que está completamente desarrollado usando lenguaje bash. Por lo tanto no es una buena opción para reutilizar pues es complicado integrar código bash con Python o C/C++ que son los lenguajes usados para la desarrollar la mayoría del software de este tipo.

Yast

El instalador de OpenSuse, es uno de los instaladores más completos. Con un meta-configurador que facilita en gran medida, la gestión de la configuración de la mayoría de los archivos existentes en el sistema. Desarrollado mayormente en Perl y C/C++, es una buena opción para reutilizar, pero el tiempo necesario para añadirle nuevas funcionalidades es considerablemente grande y debido a la complejidad del lenguaje C/C++ los recursos humanos para llevar a cabo esta empresa crecerían notablemente. Además, está implementado para un sistema con configuraciones distintas a las de Gentoo, por lo que sería necesario modificar una gran cantidad de código con el fin de hacer compatible Yast con un sistema con Gentoo como base.

Figura 4: Vista de selección del tipo de sistema. Yast, instalador de OpenSuse

Debe destacarse en este instalador la alta disponibilidad de software y elecciones con que cuenta el usuario desde el principio (fig. 4). Siendo posible escoger entre:

1. Sistema con entorno de escritorio Gnome.
2. Sistema con entorno de escritorio KDE.
3. Sistema con entorno de escritorio XFCE.
4. Sistema minimalista con solo X-Windows.
5. Sistema minimalista orientado a servidores, solo modo texto.

Instalador de Gentoo

Conocido por la comunidad de usuarios de sistemas Linux como el más complejo de los instaladores. El instalador de Gentoo , presenta un una interfaz gráfica y otra en modo texto desde las cuales el usuario puede interactuar con los archivos de configuración para configurar elementos como pueden ser:

- Kernel[1]
- Servicios al iniciar.
- Configuración de los stages[2].
- Paquetes adicionales.

Y otra serie de pasos para los cuales es necesario tener un conocimiento avanzado en la compilación de sistemas.

Figura 5: Selección de Zona horaria. Gentoo Installer.

La mayoría de su código escrito en C/C++ y la alta complejidad de los procesos que maneja, lo hacen un instalador difícil de mantener. Por otra parte la filosofía que usa para realizar la instalación (la compilación de todo el sistema) no es la que se quiere utilizar para las soluciones de GNU/Linux Nova.

Instalación de sistemas *versus* clonación de sistemas

La clonación de sistema es una práctica bastante extendida en el mundo. Pero la misma presenta ventajas y desventajas de acuerdo al entorno en el cual se esté utilizando. En muchas instituciones donde el hardware es el mismo para muchas estaciones de trabajo y las prestaciones que debe

[1] *Núcleo del sitema.*

[2] *Niveles de compilación del sistema. Cada uno añade características especificas*

presentar el sistema no varían, tiene mucho sentido aplicar esta técnica. Pero los problemas que aparecerán cuando no es así ameritan el estudio de otras opciones. Es entonces cuando la instalación gana espacio. Al instalar en una estación de trabajo el software es configurado específicamente para el hardware de la misma, además, los paquetes son personalizados de acuerdo al tipo de trabajo que se realiza en esa máquina lo que proporciona un mayor rendimiento y mejor uso de recursos.

Por lo tanto, sólo es recomendable la clonación de sistemas cuando:

1. Las computadoras involucradas cuentan con el mismo hardware o compatible.
2. El sistema cuenta con todas las prestaciones necesarias para el funcionamiento de esa estación de trabajo.
3. El tiempo de adaptar el sistema (instalar y desinstalar programas, librerías o controladores) es menor que el tiempo de instalación del sistema propiamente dicho.
4. El sistema que se está clonando ha sido revisado minuciosamente en cuanto seguridad.

Fuera de estos casos, es preferible instalar. La computadora estará lista en menos tiempo, más segura y las configuraciones se harán desde un inicio para la máquina en cuestión.

Tendencias tecnológicas

En la mayoría de los instaladores los lenguajes de programación son en un primer lugar, Python y en un segundo lugar C/C++. Las librerías gráficas más usadas son GTK y QT para las interfaces gráficas mientras para las interfaces modo texto lo más usado es ncurses y python-newt. El lenguaje C/C++ es mayormente utilizado para programación de sistema, mientras que Python es usado para la programación de la interfaz(gráfica o modo texto) de la aplicación.

Conclusiones

La investigación desarrollada ha demostrado que actualmente existen instaladores que podrían ser utilizados para la instalación de la distribución GNU/Linux Nova. Pero ninguno de ellos cuenta con la estructura necesaria para la rápida implementación de nuevas vistas.

En muchos casos sería necesario adaptar el sistema al instalador, lo que puede afectar directa o indirectamente algún requisito en el resultado final de la solución pedida por el cliente.

Por lo que se demostró la necesidad de construir un instalador que cubra las necesidades ya mencionadas. Siendo posible la implementación de nuevas vistas a través de un framework también implementado como parte de la solución.

Capítulo 2: Entorno de desarrollo

Introducción

En el capítulo anterior el estudio del dominio arrojó que todos los instaladores estudiados presentan elementos reutilizables y en teoría, a excepción de el instalador de Gentoo, todos pueden utilizarse para realizar la instalación de GNU/Linux Nova. Pero, a pesar de ello, ninguno está diseñado para ser modificado en cortos períodos de tiempo. La adopción de cualquiera de los instaladores existentes por el proyecto Nova sería contraproducente. Pues en el momento de agregar o eliminar funcionalidades el gasto de recursos tanto humano como tiempo aumentaría considerablemente.

El escenario menos agresivo es aquel en el que se necesite modificar solo una de las vistas para cumplir con uno de los requisitos. Por el contrario, cuando se requiera modificar o agregar vistas en más de uno de los instaladores de las distintas soluciones que brinda Nova, serían necesarios, incluso, más de un equipo para el desarrollo de las nuevas prestaciones.

Además, para utilizar uno ya implementado, en muchos casos habría que adaptar el sistema al instalador y no solo viceversa. Por ejemplo: para utilizar Anaconda es necesario un gestor de paquetes[3] capaz de manejar archivos empaquetados con el formato rpm[4].

Por todo lo expresado anteriormente, se **propuso como solución** implementar Serere[5]. Una aplicación flexible y que cuente con los requisitos antes expuestos. Serere consta de dos partes esenciales: la base de un instalador a la cual se le pueden agregar o eliminar vistas y un framework para el desarrollo de dichas vistas o secciones.

---

[3] *Software para administrar la instalación de paquetes y aplicaciones en el sistema.*

[4] *De las siglas RPM, Red Hat Package Manager.*

[5] *Serere. Latín, plantar, sembrar.*

El entorno de desarrollo como lo muestra este documento, es el conjunto de herramientas o elementos, utilizados para implementar la solución.

Los mas importantes de estos elementos son:

- Metodología
- Arquitectura
- Lenguajes de programación.
- Entorno Integrado de Desarrollo (IDE, por sus siglas en Inglés).
- Herramientas de gestión.

En este capítulo se exponen las consideraciones tomadas en cuenta para la selección de cada uno. Y en casos como son metodología y gestión se profundizará en el uso de las mismas, y cómo es que son implementadas para llevar a cabo el desarrollo de la solución propuesta.

Programación Extrema como metodología de desarrollo

La metodología seleccionada para el desarrollo de Serere es Programación Extrema(XP por sus siglas en Inglés). XP es una metodología que pertenece al grupo de las llamadas ágiles. Y fue elegida porque las prácticas que dicta se ajustan perfectamente a la misión del proyecto Serere.

Programación en parejas.

La programación en parejas reduce en gran medida la introducción de errores en el código, pues éste es escrito y revisado al mismo tiempo. En adición a esto, está el hecho de que un programador está brindando la mayor parte de su atención a la escritura mientras, que el otro se enfoca en el diseño, la refactorización y el chequeo de errores lógicos.

Por otra parte resuelve parcialmente el problema de la capacitación del personal. Formando las parejas de forma que contengan un desarrollador experto y uno con un nivel de experticia inferior.

Programación "test-first".

En la programación "test-first", gran parte las pruebas de caja blanca[6] son automatizadas haciendo uso de las pruebas unitarias[7]. Con la particularidad de que dichas pruebas son implementadas antes de las funciones o módulos que serán probados. En un inicio, todas fallarán. Se considera que se ha terminado de programar cuando el software pasa todas las pruebas.

Esto expuesto facilita el trabajo directo con el código, pues permite refactorizarlo, optimizarlo o actualizarlo y comprobar que sigue funcionando.

Constante interacción del cliente con el equipo.

Esta práctica garantiza una retroalimentación casi instantánea con las no conformidades del cliente, detectando a tiempo los cambios en los requerimientos del software en construcción.

Propiedad colectiva del código.

Código no es de nadie, no tiene dueños y cualquiera debe poder modificarlo.

---

[6] *Pruebas que se realizan al código.*

[7] *Pruebas dirigidas a "unidades" específicas del software, módulos, funciones, etcétera.*

Arquitectura

Metáfora: *El instalador descomprime el sistema configura los archivos necesarios para su funcionamiento y permite la gestión de vistas (crear, incorporar y eliminar)*

Serere está concebido según el patrón de arquitectura: Modelo-Vista-Controlador. El patrón fue seleccionado atendiendo al hecho de que la interfaz gráfica estará cambiando constantemente y es necesario separarla de la lógica del negocio, para ganar flexibilidad y reusabilidad. Así mismo, los archivos de configuración de un sistema Linux, se sustituyen muy a menudo por nuevas versiones, esto trae como consecuencia la aparición de nuevas variables y la desaparición de otras. Este patrón permite cambios en el modelado de la aplicación sin la necesidad de modificar el código de la interfaz, con el objetivo de hacer el software compatible con las variaciones de dichos archivos.

Los tres grupos que componen este patrón, están definidos para Serere de la siguiente manera:

Modelo

En este grupo del patrón, se encuentran reunidos los módulos correspondientes a la modelación de los distintos elementos que en un momento dado serán consultados y/o editados por las vistas del instalador. Ejemplo: datos de usuario, propiedades de red, sistema operativo, configuración de gestor de arranque, etcétera.

Vista

Conjunto de todas las interfaces gráficas utilizadas para la interacción del usuario con los modelos correspondientes.

Controlador

El instalador en si mismo, controla y gestiona todo lo que acontece sobre las vistas y modelos.

Plataforma de desarrollo

El Entorno de Desarrollo Integrado (IDE, por sus siglas en inglés) es Eclipse en su versión 3.3.1. Eclipse es software libre y tiene la capacidad de ser ampliado o extendido con una serie de plug-ins dentro de los cuales se encuentran el conjunto de los utilizados para la implementación de Serere.

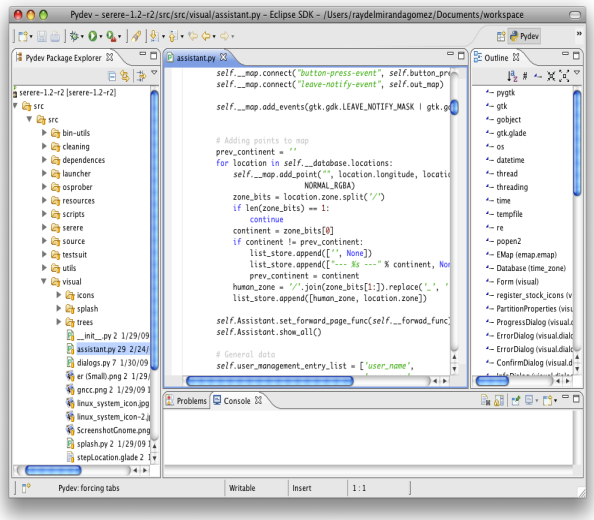

Figura 5: Eclipse

Plugins de Eclipse para el desarrollo de Serere.

Nombre	Versión	Propósito	Compatibilidad con Eclipse
PyDev	1.3.18	Ofrece la perspectiva de desarrollo para el lenguaje de programación Python.	Eclipse >= 3.2
Subclipse	1.2.4	Ofrece la perspectiva para el trabajo y control de versiones haciendo usos de repositorios de subversión a través del protocolo svn.	Eclipse >= 3.2
PyUML	1.0.1	Ofrece la perspectiva para la construcción de diagramas UML y la exportación de los mismos a código Python y viceversa.	Eclipse >= 3.3

CDT-Master	4.0.3	Ofrece la perspectiva para el trabajo con los lenguajes C y C++	Eclipse >= 3.3

Tabla 1 Plug-ins para el desarrollo de Serere con el IDE Eclipse.

Lenguajes de programación

Python, es un lenguaje muy potente y fácil de aprender. Cuenta con una gran popularidad a nivel mundial lo que lo convierte en uno de los lenguajes mas documentados que existen. Usándolo se puede programar usando la mayoría de los paradigmas de programación[8] e implementarse varios de los patrones de diseño dentro de los que destacan: Fábrica abstracta (Abstract Factory), Observador (Observer), Instancia única (Singleton) y Constructor (Builder).

Por ser un lenguaje interpretado, la ejecución de código Python suele ser más lenta que la puesta en marcha de un archivo binario resultado de la compilación de un lenguaje como puede ser C o C++. Para eliminar esta barrera existe Python.h una librería que permite la incorporación de código C o C++ como módulos y funciones Python.

Por todas estas características y ser un lenguaje de fácil aprendizaje, se convierte en el ideal para el uso del desarrollo de Serere. Pues en el marco donde se desarrolla (Universidad de las Ciencias Informáticas) una de las premisas de los proyectos productivos, es la capacitación del nuevo personal (en su mayoría estudiantes). Python facilita llevar a cabo este proceso sin perder el foco en el desarrollo.

También, el hecho de que sea el lenguaje más utilizado para llevar a cabo la construcción de software de este tipo, brinda al equipo de desarrollo grandes posibilidades de poder reutilizar código. Aspecto que disminuiría aún más el tiempo de respuesta de los desarrolladores para cada una de las soluciones personalizadas de la distribución GNU/Linux Nova.

---

[8] *Enfoque particular o filosofía para la construcción de un software determinado. Ej. Programación orientada a objetos (POO).*

Los módulos PyGtk y PyQt, interfaces de python para Gtk [9] y Qt[10] respectivamente, facilitan la implementación de interfaces visuales, lo que completa el set de herramientas necesarias para hacer de este lenguaje el elegido para la implementación de la solución propuesta en este documento.

Gestión

Para la gestión del proyecto se escogió la herramienta Gforge[11]. Dentro de los usos que se le dan a la misma se pueden encontrar: gestión de documentación, reportes y tareas así como control de versiones.

Gestión de documentación

Con este propósito se usa la vista de Gforge de mismo nombre. En la cual se pueden crear y editar grupos para la organización de los distintos documentos.

Figura 6 Vista de gestión de documentos en Gforge.

Desde esta sección el usuario puede leer y publicar nuevos documentos previa autorización del administrador del proyecto. Quien tiene a su vez la opción de administrar la documentación del proyecto, creando, editando, moviendo o eliminando los distintos documentos y categorías.

---

[9] *Gtk (Gimp Tool Kit) Librería gráfica para construir los controles visuales de Gnome.*

[10] *QT Librería gráfica para construir los controles visuales de KDE.*

[11] *Herramienta web para la administración de proyectos. Herramienta colaborativa.*

Gestión de reportes

La gestión de reportes se divide en 4 subgrupos: reporte de bugs, soporte, parches y finalmente pedidos de requisitos o prestaciones. Los más importantes son los apartados de bugs y prestaciones, pues los mismos revelan lo que está mal en el software y lo que debe cambiar o añadírsele al mismo.

Cada reporte de bug, soporte, parche o pedido de prestación se denomina registro. Un registro contiene la información necesaria para su seguimiento ordenada dentro de seis campos: resumen, fecha de envío, prioridad, responsable, remitente y estado. Este último puede tomar dos valores (cerrado o abierto), que reflejan si se a resuelto o no lo planteado en el reporte.

Gestión de tareas

La gestión de tareas es una actividad que se encuentra estrechamente relacionada con la planificación del proyecto. Esta vista de Gforge es usada con ambos fines. Es aquí donde se asignan a los integrantes del equipo de trabajo las distintas tareas a cumplir en un tiempo determinado.

Los campos que figuran dentro de una tarea son: descripción de la tarea, fecha de inicio, fecha de final, porcentaje completado e id. Una tarea puede ser creada, eliminada o modificada de acuerdo a las necesidades del administrador del proyecto para con el mismo, teniendo además la opción de generarlas a partir de un reporte.

Como herramienta auxiliar, desde esta vista de Gforge se puede generar un diagrama de Gantt[12], que permite al administrador del proyecto tener una vista rápida de los objetivos inmediatos del equipo.

[12] *Diagrama que muestra la disposición de las tareas en el tiempo y las dependencias entre las mismas.*

Figura 7 Vista de gestión de tareas del Gforge

Control de versiones

El control de versiones se lleva a cabo utilizando dos de las prestaciones de Gforge. La primera es el servicio de subversion [13](fig. 9) y la segunda es la vista de ficheros de liberación del proyecto, donde se publican las distintas versiones del producto liberado.

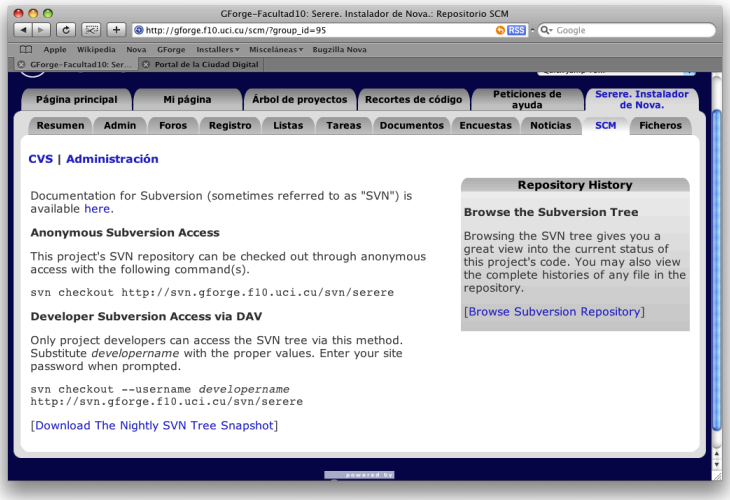

Figura 8: Vista de subversión de Gforge. Proyecto

Eclipse integra el servicio de control de versiones brindado por Gforge, permitiendo una interacción casi transparente al desarrollador. Brindando prestaciones como son: comparación de versiones de código, actualización, sincronización , historial de versiones, etcétera.

[13] *Subversion. Uno de los protocolos utilizados para el control de versiones.*

Nova como sistema operativo base

Para el desarrollo de Serere se usa como sistema operativo GNU/Linux Nova ya que es el sistema sobre el cual "corre".

El instalador depende de programas, librerías y archivos. Dentro del conjunto de programas (tabla 2) destacan: Unsquashfs y os-probe, el primero, es un programa utilizado para la descompresión de archivos con formato squashfs, se caracteriza por su gran velocidad de descompresión, logrando descomprimir 700 Mb de información en sólo 10 minutos para ordenadores con 256 Mb de memoria RAM[14]. El segundo, usado por Debian y Ubuntu para el reconocimiento de otros sistemas operativos instalados en la computadora, con el objetivo de configurar posteriormente el gestor de arranque. Usado por Serere, además, con el propósito de mostrar dichos sistemas en la vista de administración de discos duros.

Tabla de programas

Nombre	Versión	Propósito	Comentarios
unsquashfs	1.5	Descompresión de archivos con formato squashfs.	---
os-probe	1.23	Detección de sistemas operativos.	---
Gparted	0.3.5	Administración de disco duro.	Para esta versión fue necesario aplicarle un parche, en la forma en que se formatea con el sistema de archivos "ext3". Se redujo el tamaño de los nodos de 256 a 128 bytes.

[14] *Memoria de acceso aleatorio. RAM por sus siglas en Inglés: Random Access Memory*

			Para garantizar la compatibilidad con Grub.
Grub	0.97	Gestor de arranque.	Versión que no soporta la lectura a sistemas de archivos “ext3” con nodos de una longitud mayor a 128 bytes.
Bash	3.2.33	Intérprete de comandos	
Python	2.5	Intérprete de para el lenguaje Python	El código de Serere, utiliza sintaxis que no son compatibles con versiones menores a 2.5
Gettext	0.17	Herramienta para internacionalización	---

Tabla 2: Programas que constituyen dependencias para Serere.

Tabla de librerías

Nombre	Versión	Comentario
libparted	1.8.8	---
libgtk	2.0	---
os-probe	1.23	---

libglade	2.0	---
Nota: *Las demás librerías usadas, están incluidas por defecto en cualquier sistema Linux, pues son necesarias para su funcionamiento.*		

Tabla 3: Librerías utilizadas para el funcionamiento y desarrollo de Serere

Los archivos con los que interactúa Serere son en su totalidad archivos de configuración

Tabla de archivos

Dirección	Generado/Editado	Propósito
"/etc/conf.d/net"	G	Configuración de red.
"/etc/fstab**"**	G	Configuración del montaje[15] de los dispositivos
"/etc/X11/xorg.conf**"**	G	Configuración del servidor X.
"/etc/X11/gdm/custom.conf"	E	Configuración de la ventana de entrada el sistema.
"/etc/init.d/xdm"	E	Script de inicio del xdm[16]
"/etc/sudoers"	E	Configuración de permisos de usuario.
"/etc/inittab"	G	Configuración de las terminales.
"/etc/passwd"	E	

[15] Los distintos dispositivos o particiones son montadas en distintas direcciones con diversas opciones.

[16] Administración de pantalla de X (XDM por sus siglas en Inglés) usado para el manejo de sesiones.

“/etc/shadow”	E	Enmascaramiento de los datos de “/etc/passwd”
“/boot/grub/menu.lst”	G	Configuración del gestor de arranque Grub.

Tabla 4: Archivos de configuración involucrados con Serere.

Conclusiones

Las herramientas y la metodología que en su conjunto conforman el entorno de desarrollo de Serere, responden adecuadamente a las necesidades de desarrollo y gestión de dicho proyecto.

La metodología escogida (XP), brinda la organización y los procesos necesarios para llevar a cabo la construcción de la solución. Serere es un proyecto mediano y no son necesarios tantos artefactos como podría generar cualquiera de las metodologías tradicionales.

Por otra parte, Gforge es una excelente opción para la gestión así como para la administración del hacer de las distintas partes del equipo de desarrollo. Con Gforge la información del proyecto está centralizada y es accesible para los integrantes del equipo aunque no estén ubicados en un mismo lugar.

Eclipse como plataforma de desarrollo integrado. Ofrece a los desarrolladores un entorno de desarrollo cómodo con todas las herramientas necesarias. Sus extensiones para el uso de otros lenguajes además de Java, como pueden ser C, C++ y Python en adición con las que permiten la integración con un servidor subversión lo convierten en el IDE ideal para el proyecto en cuestión.

Por último, el lenguaje de programación escogido y el sistema operativo sobre el cual se desarrolla y prueba el instalador son Python y GNU/Linux Nova respectivamente. Python, un lenguaje fácil de aprender que genera un código que puede ser mantenido sin mucho esfuerzo. Es un lenguaje interpretado, pero cuando se requiere de velocidad puede extenderse usando C o C++. Nova, es el sistema operativo sobre el cual “corre” la aplicación, por lo que es el que se debe utilizar para el desarrollo de la misma.

Capítulo 3: Descripción de la solución

Introducción

La solución completa fue dividida, según la metodología aplicada, en distintas historias de usuarios. Las mismas serían desarrolladas en un proceso iterativo e incremental integrándose posteriormente cada una de ellas al conjunto de historias ya desarrolladas. Logrando una liberación contínua de mini-versiones de Serere.

La solución final se divide además en dos grupos: la base de Serere y el framework para agregar vistas a dicha base, Arare. Para la implementación de la interfaz de la base se utiliza el control “Assistant”, de la librería Gtk-2.0. Este control proporciona un ambiente dentro del cual se puede controlar el flujo de trabajo de una manera relativamente sencilla.

Los diagramas de clase generados por las tareas de ingeniería aplicadas a cada historia de usuario, pueden ser consultados en el **anexo 1** del documento.

Base de Serere

Historia de usuario número 1: Selección del modo de instalación

Historia de Usuario	
No: 1	**Nombre:** Selección del modo de instalación
Usuario: Usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Puntos estimados: 3/5	**Iteración asignada:** 1
Descripción: El usuario escoge el tipo de instalación que va a desarrollar: automática o manual.	
Observaciones:	

Historia de usuario 1: Selección del modo de instalación.

La interfaz gráfica o de usuario(UI por sus siglas en Inglés) fue diseñada siguiendo los principios de diseño dictados por el documento: "Human Interface Guideline (HIG)" en su versión 2.0, el cual representa el proyecto de usabilidad de Gnome[17].

Todas las interfaces gráficas de la solución fueron diseñadas siguiendo esos principios.

---

[17] *Proyecto dedicado a la construcción del entorno de escritorio del mismo nombre.*

En esta vista el usuario tiene la opción de escoger, en cual de los modos va a realizar la instalación, automático o manual. Para cada modo escogido, el instalador se comportará de un modo diferente.

Por la vía automática, Serere determinara si es necesario la existencia de una partición de intercambio swap[18] y en caso de que sea necesaria, calculará el tamaño apropiado para la misma. Acto seguido, procede al formateo de la partición donde estará ubicado el sistema.

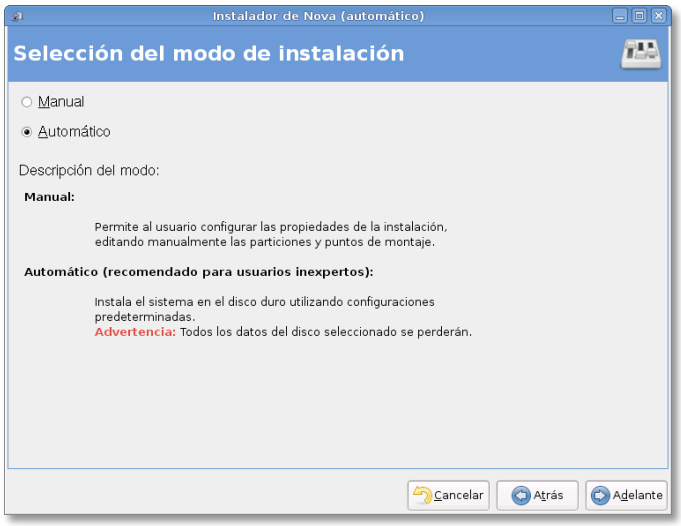

Figura 9: Selección del modo de instalación.

Por otra parte, si el modo escogido es el manual, se añade un paso al instalador en el cual el usuario procede a la edición manual de las particiones.

Serere, una vez que se completa el proceso de configuración y el usuario decide comenzar la instalación, realiza una llamada al método encargado de instalar el sistema.

Python permite asignar un método a una variable en tiempo de ejecución, propiedad que es utilizada para asignar el método de instalación según el modo escogido por el usuario.

Para probar esta historia de usuario se diseñaron dos escenarios:

Escenario 1. Selección de modo de instalación.	
Descripción: El usuario selecciona modo de instalación automática.	**Resultados esperados:** 1. La vista de edición de particiones será inaccesible por el usuario. 2. La función a ejecutar debe ser

[18] *Partición de intercambio para sopesar la insuficiencia de la cantidad de memoria RAM.*

	“on_Assistant_auto_apply”
Resultados obtenidos: 1. No implementada en este punto. 2. Correcto.	
Observaciones: La vista de edición de particiones es una historia de usuario que no ha sido implementada aún.	

Tabla 5: Escenario de prueba #1, Selección del modo de instalación.

Escenario 2. Selección de modo de instalación.	
Descripción: El usuario selecciona modo de instalación manual.	**Resultados esperados:** 1. La vista de edición de particiones será accesible por el usuario. 2. La función a ejecutar debe ser “on_Assistant_apply”
Resultados obtenidos: 1. No implementada en este punto. 2. Correcto.	
Observaciones: La vista de edición de particiones es una historia de usuario que no ha sido implementada aún.	

Tabla 6: Escenario de prueba #2, Selección del modo de instalación.

Historia de usuario número 2: Selección de zona horaria

Historia de Usuario	
No: 2	**Nombre:** Selección de zona horaria
Usuario: Usuario	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Puntos estimados: 3/5	**Iteración asignada:** 1
Descripción: El usuario selecciona la zona horaria por la que se regirá el sistema.	
Observaciones: Para esta historia de usuario son necesarias sólo algunas modificaciones.	

Historia de usuario 2: Selección de zona horaria.

Para esta interfaz se reutiliza la vista de mismo propósito del instalador de Ubuntu[19]. Pero la manera en que la muestra Ubuntu presenta elementos que atentan en contra uno de los principios dictados por la HIG.

[19] *Distribución de Linux.*

Ubiquity[20] proporciona la opción de escoger una ciudad de un grupo, pero este grupo es mostrado a través de un Comobobox[21](fig. 12, izquierda). HIG dicta que cuando las opciones son más de 9, debe usarse una lista(fig. 12, derecha). Los elementos son más fáciles de encontrar y no cubren otras partes de la aplicación.

Usar el control del mapa es posible importando el módulo de python “emap”. Con el objetivo de agregar dicho control al asistente se procede a crear un objeto de tipo “Emap” y se carga la base de datos de las localidades para situar los puntos.

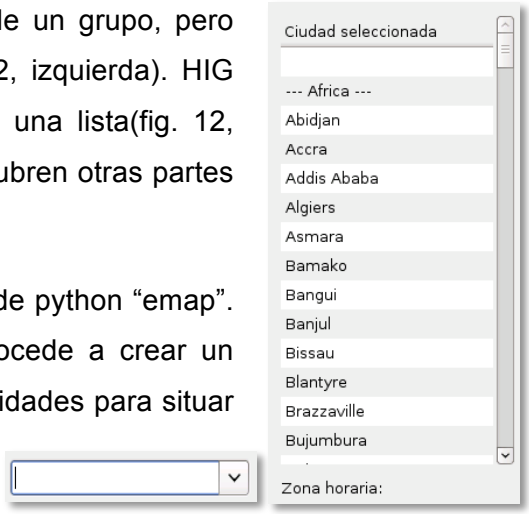

Figura 10: Combobox y lista de izquierda a derecha.

Para probar esta historia de usuario se virtualizó la estructura de carpetas del sistema a instalar y se comprobó que se modificara correctamente el archivo “hwclock” que es el encargado de portar la configuración del reloj y la zona horaria.

Escenario 1. Selección de zona horaria.	
Descripción: El usuario selecciona una zona horaria.	**Resultados esperados:** 1. El archivo “hwclock” de no existir se genera. 2. El archivo está bien configurado.
Resultados obtenidos: Correcto.	

[20] *Instalador de Ubuntu*

[21] *Control visual compuesto de un cuadro de texto y una lista desplegable.*

Observaciones: Este escenario es repetitivo. Es decir, se repite varias veces cada vez con una zona distinta.

Tabla 7: Escenario de prueba #1. Selección de zona horaria.

Historia de usuario número 3: Administración de discos duros

Historia de Usuario	
No: 3	**Nombre:** Administración de discos duros.
Usuario: Usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alto
Puntos estimados: 25/5	**Iteración asignada:** 1
Descripción: El usuario crea, edita y elimina las particiones que estime.	
Observaciones: ---	

Historia de usuario 3: Administración de discos duros.

La interfaz diseñada para esta historia de usuario, es la más compleja de la toda la solución y por tanto es a la que más tiempo se le dedicó. En esta vista, se muestra información al usuario acerca del los discos duros existentes y sus respectivas particiones.

Información acerca de los discos duros

El uso de el módulo pyparted es esencial para poder brindar toda la información necesaria para un correcto desempeño del usuario en el momento de enfrentar la aplicación. Con el mismo se

resuelven datos como son: distribuidor, tipo de conexión, modelo, tamaño, etcétera. En adición, es posible observar también la cantidad de discos existentes y su respectiva ruta[22].

Árbol de particiones del disco duro seleccionado

Esta sección de la vista muestra al usuario información relativa a las particiones de un disco duro en específico. Para el usuario que instala usando Serere, una partición tiene: nombre y sistema de fichero, además, opcionalmente puede tener: punto de montaje, tamaño y sistema operativo instalado. Si el disco presenta espacio libre, éste es también reflejado en el árbol de particiones. Las particiones que se están usando en el momento de mostrar la vista, son resaltadas con el ícono de Gtk que representa restricción de acceso(fig. 14).

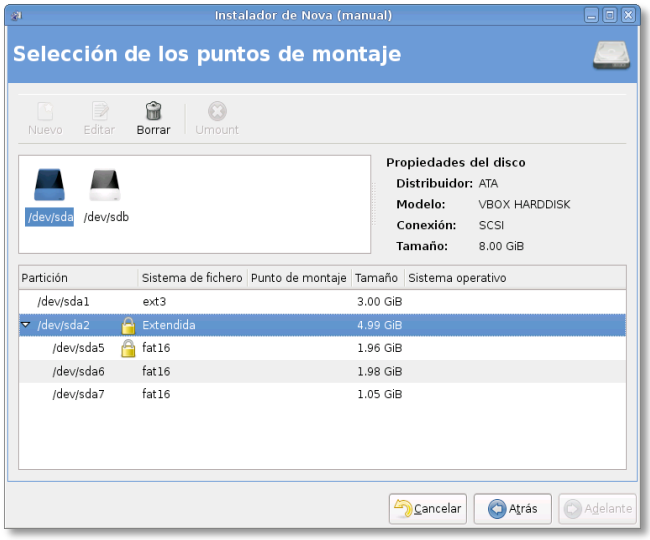

Figura 12: Vista del árbol de particiones.

Para acceder a las opciones de edición, el usuario cuanta con una barra de herramientas desde la cual puede realizar las operaciones pertinentes.

Desde esta barra es posible desmontar[23] , editar y crear particiones. Otra vía para llevar a cabo cualquiera de estas tres acciones es a través de un “click derecho” en la partición que se quiere modificar o en el espacio vacío en el cual se quiere crear una nueva. La acción de editar partición se completa mediante un diálogo en el que el usuario proporciona los datos necesarios, tales como: formato[24] y punto de montaje[25] .

Figura 13: GTK_STOCK_DIALOG_AUTHENTICATION

---

[22] *Ruta en formato Unix, Ej. “/dev/sda”*

[23] *Montar una partición es asignar un directorio a la misma, desde la cual será accesible. Desmontarla es liberar ese directorio.*

Las pruebas realizadas a esta historia de usuario fueron enfocadas en la edición de particiones y la validación de errores introducidos por el usuario en momentos como la especificación de formato o punto de montaje.

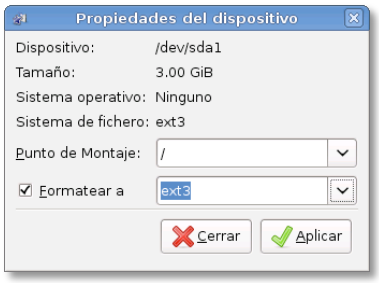

Figura 14: Diálogo de edición de particiones.

Escenario 1. Administración de particiones.	
Descripción: El usuario selecciona un punto de montaje para una partición determinada. El punto de montaje es un camino valido y existe.	**Resultados esperados:** 1. En el sistema instalado, aparece configurado debidamente el archivo "/etc/fstab".
Resultados obtenidos: Correcto.	
Observaciones: ---	

Tabla 8: Escenario de prueba #1. Administración de particiones.

Escenario 2. Administración de particiones.	
Descripción: El usuario selecciona un punto de montaje para una partición determinada. El punto de montaje no es un camino valido.	**Resultados esperados:** 1. Mensaje de error visualizado por el usuario.

[24] *FAT, ext3, reisefs, etc ..*

[25] *Directorio en el cual se monta una partición.*

Resultados obtenidos: Correcto.
Observaciones: ---

Tabla 9: Escenario de prueba #2. Administración de particiones.

Historia de usuario número 4: Configuración de red

Historia de Usuario	
No: 4	**Nombre:** Configuración de red
Usuario: Usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Bajo
Puntos estimados: 1/5	**Iteración asignada:** 1
Descripción: El usuario define el nombre de la computadora para la red y configura la red con los datos: dirección IP, máscara de red, y puerta de enlace.	
Observaciones: ---	

Historia de usuario 4: Configuración de red.

La vista de configuración de red es sencilla, con pocos controles y de fácil familiarización. En esta sección de Serere, el usuario introduce los datos necesarios para una correcta configuración. Por la complejidad de dichos datos y la frecuencia con que son mal formulados es una de las vistas donde mas validación de errores se implementaron. En un intento

Figura 15: Ejemplo de dato incompleto.

por reducir los errores cometidos por el usuario, se usa color rojo para el texto de un dato con mal formato (fig. 16).

El usuario, además, tiene la posibilidad de configurar todas las interfaces de red existentes en el ordenador donde se está llevando a cabo el proceso de instalación.

Las pruebas realizadas para esta historia de usuario, estuvieron centradas en la correcta configuración del archivo "/etc/conf.d/net". Archivo contenedor de los datos de las interfaces de red.

Escenario 1. Configuración de red.	
Descripción: El usuario realiza una configuración.	**Resultados esperados:** 1. El archivo involucrado está configurado debidamente.
Resultados obtenidos: Correcto.	
Observaciones: El formato de los datos nunca estará mal escrito, pues de ser así la vista no estaría completa y por tanto no se puede avanzar.	

Tabla 10: Escenario de prueba #1. Configuración de red.

Historia de usuario número 5: Administración de usuarios

Historia de Usuario	
No: 4	**Nombre:** Administración de usuarios
Usuario: Usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio

Puntos estimados: 3/5	**Iteración asignada:** 1
Descripción: El usuario establece el nombre y las contraseñas para al acceso al sistema.	
Observaciones: ---	

Historia de usuario 5: Administración de usuarios.

La última de las historias de usuario de la base del instalador y la más sencilla después de la vista de selección del modo de instalación. Sólo cuenta con 5 cajas de texto, 3 para el usuario inicial 2 para del super usuario o root.

Figura 16: Contraseñas que no concuerdan.

La filosofía para el control de errores es la misma que la historia de usuario anterior, mientras los datos no estén correctos, el texto se mostrará rojo (fig. 17). Tampoco será posible escribir el nombre de usuario con mayúsculas o con caracteres como: "@", "/", ":", "\" que generalmente son usados en sistemas Linux para la confección de direcciones.

Todas estas consideraciones, deja solo un escenario para las pruebas:

Escenario 1. Configuración de usuario.	
Descripción: Se introduce el nombre y la contraseña del usuario inicial. Además de la contraseña de root.	**Resultados esperados:** 1. Se crea el usuario con su contraseña. 2. La contraseña de root se actualiza.
Resultados obtenidos:	

Correcto.
Observaciones: ---

Tabla 11: Escenario #1, Configuración de usuario.

Framework Arare

Historia de usuario número 1: Creación de nuevas vistas

Historia de Usuario	
No: 1	**Nombre:** Creación de nuevas vistas.
Usuario: Desarrollador de Serere	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 5/5	**Iteración asignada: 2**
Descripción: A partir de las clases bases y plantillas, el desarrollador crea nuevas vistas.	
Observaciones: ---	

Historia de usuario número 2: Adición de nuevas vistas

Historia de Usuario	
No: 2	**Nombre:** Adición de nuevas vistas.

Usuario: Desarrollador de Serere	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Medio
Puntos estimados: 3/5	**Iteración asignada: 2**
Descripción: El desarrollador puede agregar las vistas creadas a la base del instalador.	
Observaciones: La base del instalador debe estar desarrollada con la librería para el asistente que brinda el framework.	

Historia de usuario número 3: Eliminación de vistas

Historia de Usuario	
No: 3	**Nombre:** Eliminación de vistas.
Usuario: Desarrollador de Serere	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Medio
Puntos estimados: 2/5	**Iteración asignada: 2**
Descripción: El desarrollador puede eliminar vistas añadidas anteriormente a la base del instalador.	
Observaciones: La base del instalador debe estar desarrollada con la librería para el asistente que brinda el framework.	

Conclusiones

Al terminar el estudio de los instaladores, quedó demostrado que es contraproducente el uso de cualquiera de los mismos como instalador del sistema operativo GNU/Linux Nova. Sería necesario adaptar el sistema al instalador y no viceversa.

Por lo tanto, se implementó una solución única y adaptable a las distintas líneas de desarrollo del proyecto Nova. Dicha solución se implementó usando Python como principal lenguaje de programación y la metodología XP.

Después de desplegada cada historia de usuario dentro de la solución se obtuvo como resultado un instalador intuitivo y fácil de usar. Que permite al usuario realizar las operaciones básicas para la instalación de un sistema operativo.

Desde el punto de vista del desarrollador, Serere es una aplicación flexible, que puede modificarse consumiendo poco recurso “tiempo” y empleando poco esfuerzo. Gracias a la implementación de Arare, también son necesarios menos recursos humanos.

Hasta la fecha, la solución ha sido desplegada en dos de las versiones liberadas de GNU/Linux Nova. La última de ellas, Nova Baire-1.1.2 liberada en el marco de la Feria Internacional InfomáticaHabana 2009, fue instalada en varios stands y en la sala de navegación del Palacio de Convenciones.

Bibliografía

Anaconda team. *Anaconda.* 13 de Enero de 2009. http://fedoraproject.org/wiki/Anaconda (Consultado 17 de Febrero de 2009).

Benson, Calum. *GNOME Human Interface Guidelines.* Free software fundation. 2004.(Consultado 10 de Enero de 2009)

Cody, Preston. *Gentoo Linux Installer.* 13 de Enero de 2009. http://www.gentoo.org/proj/en/releng/installer/ (Consultado 17 de Febrero de 2009).

Comunidad de Ubuntu. *Instalación estándar.* 31 de Enero de 2009. http://guia-ubuntu.org/index.php?title=Portada (Consultado 17 de Febrero de 2009).

Cortés, José Antonio. *Conceptos de Sistema Operativo.* 1 de Enero de 1999. http://www.mflor.mx/materias/comp/cursoso/sisope0.htm.(Consultado 7 de Abril 2009)

Documentadores de Ututo. *UTUTO XS 2007 - Manual.* 28 de Mayo de 2007. http://ututo.org/www/modules/documents/xml/documents.php?xmlPath=UTUTOXS2007_Manual (Consultado 3 de Enero de 2009).

Fusco, Jhon. *The Linux Programmer's Toolbox.* Prentice Hall, 2007.

Khorell, David. *Using RUP to manage small projects and teams.* 15 de Julio de 2005. http://www.ibm.com/developerworks/rational/library/jul05/kohrell/index.html (revisado 17 de Febrero de 2009).

Medinas, Ricardo Cárdenas. *Inmersión en Python.* Free Software Foundation, 2005.(Consultado 23 de Fefrero de 2009)

Package python-newt (0.52.2-11.3). 2009. http://packages.debian.org/sid/python/python-newt.

Pop, Frans. *Debian Installer Internals.* 1 de Enero de 2006. http://d-i.alioth.debian.org/doc/internals/ (Consultado 3 de Febrero de 2009).

Shore, James. *The Art of Agile Development.* O' Reilly, 2008.(Consultado 2 Enero de 2009)

Valle, Amaury E. del. «Refuerza Microsoft medidas contra uso de versiones ilegales de sus sistemas operativos.» *Juventud rebelde*, 22 de Febrero de 2007.

Anexos

Anexo 1: Diagramas de clases

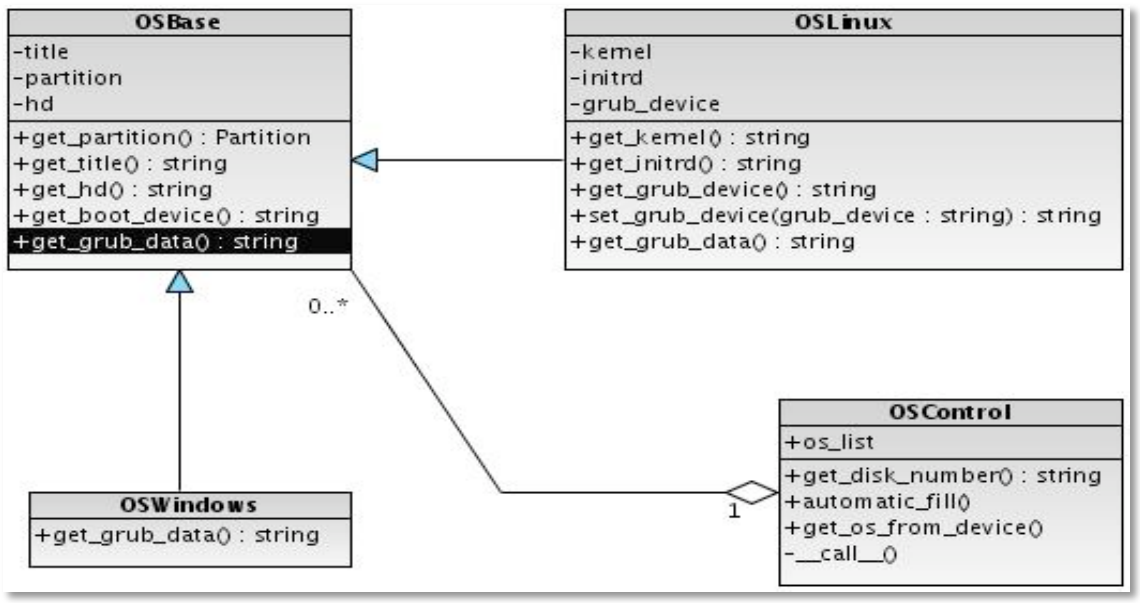

Diagrama de clases 1: Modelado de sistemas operativos para configuración de grub.

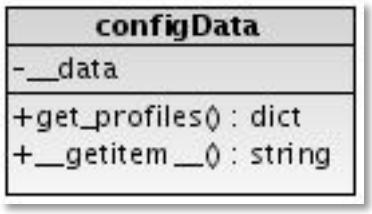

Diagrama de clases 2: Clase para la lectura de la

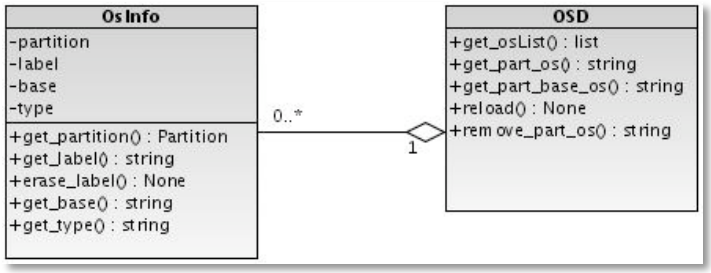

Diagrama de clases 3: Modelado de la información para los sistemas operativos detectados.

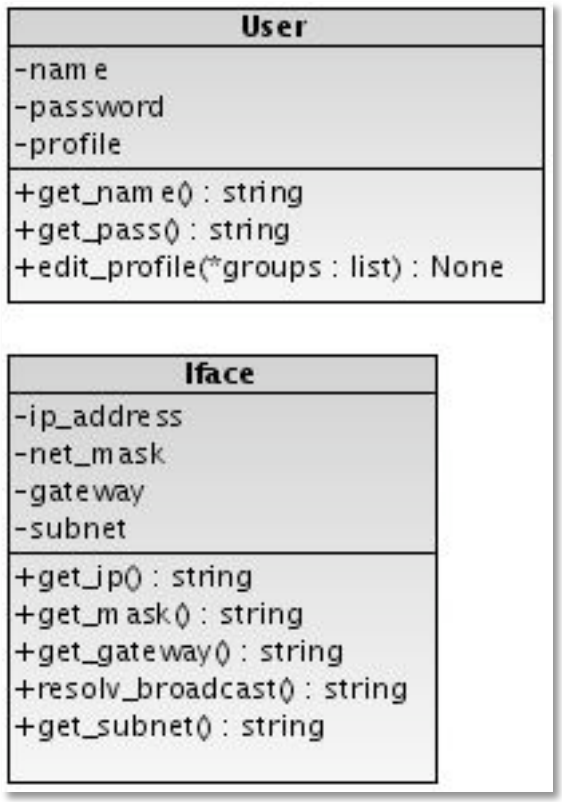

Diagrama de clases 3: Modelado de un usuario e interfaz de red.

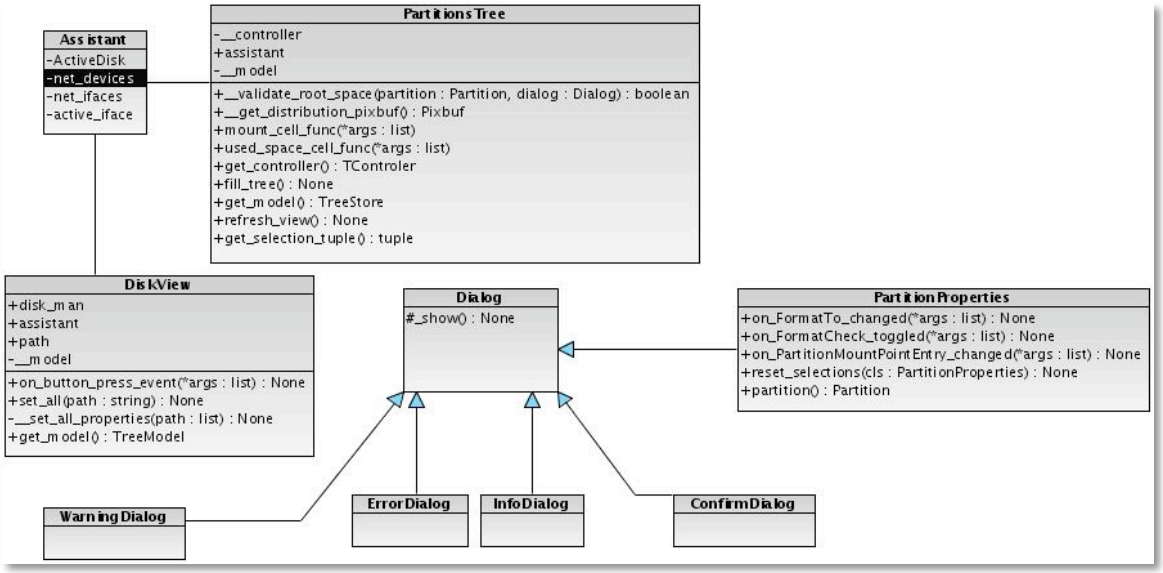

Diagrama de clases 5: Elementos visuales.

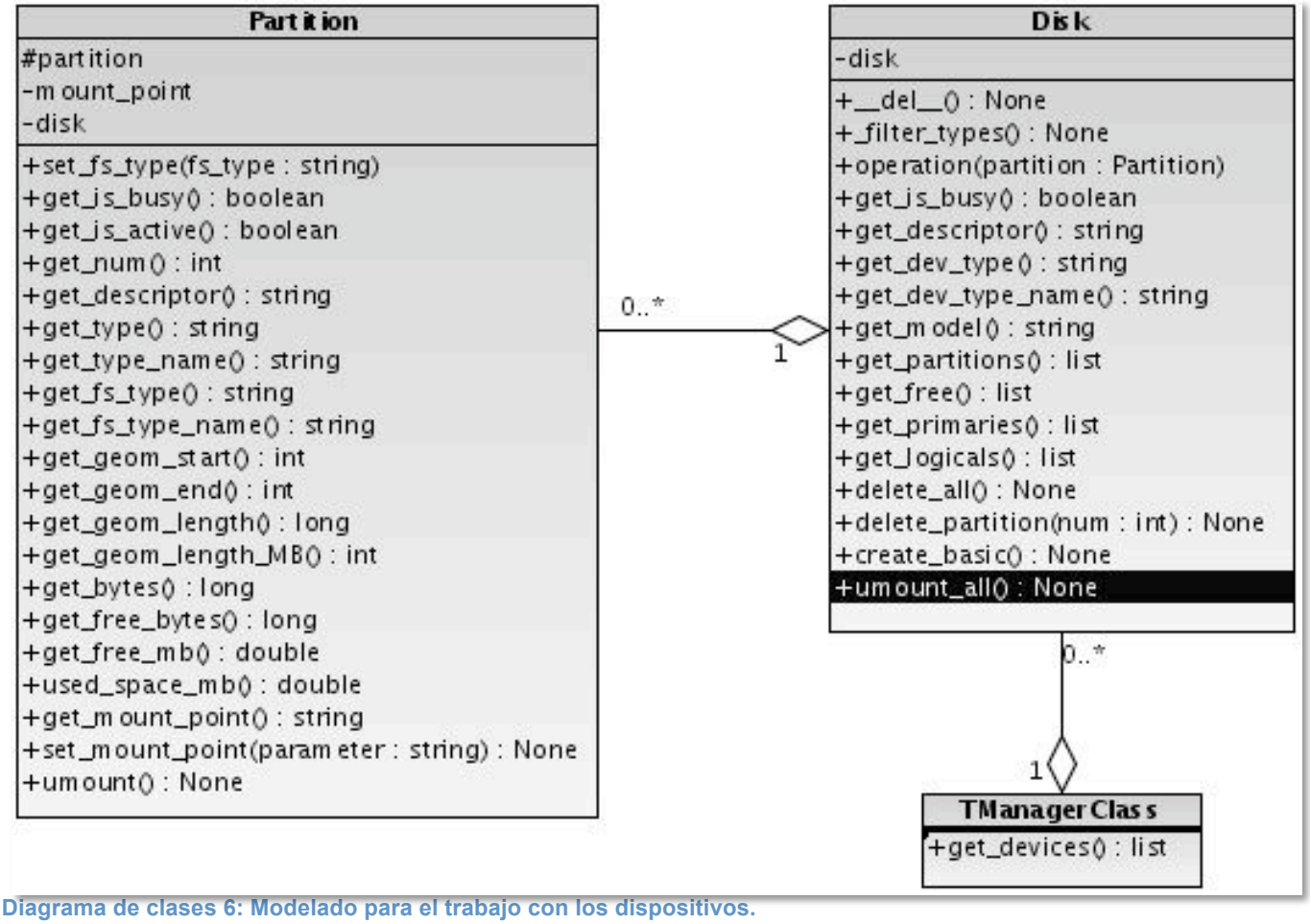

Diagrama de clases 6: Modelado para el trabajo con los dispositivos.

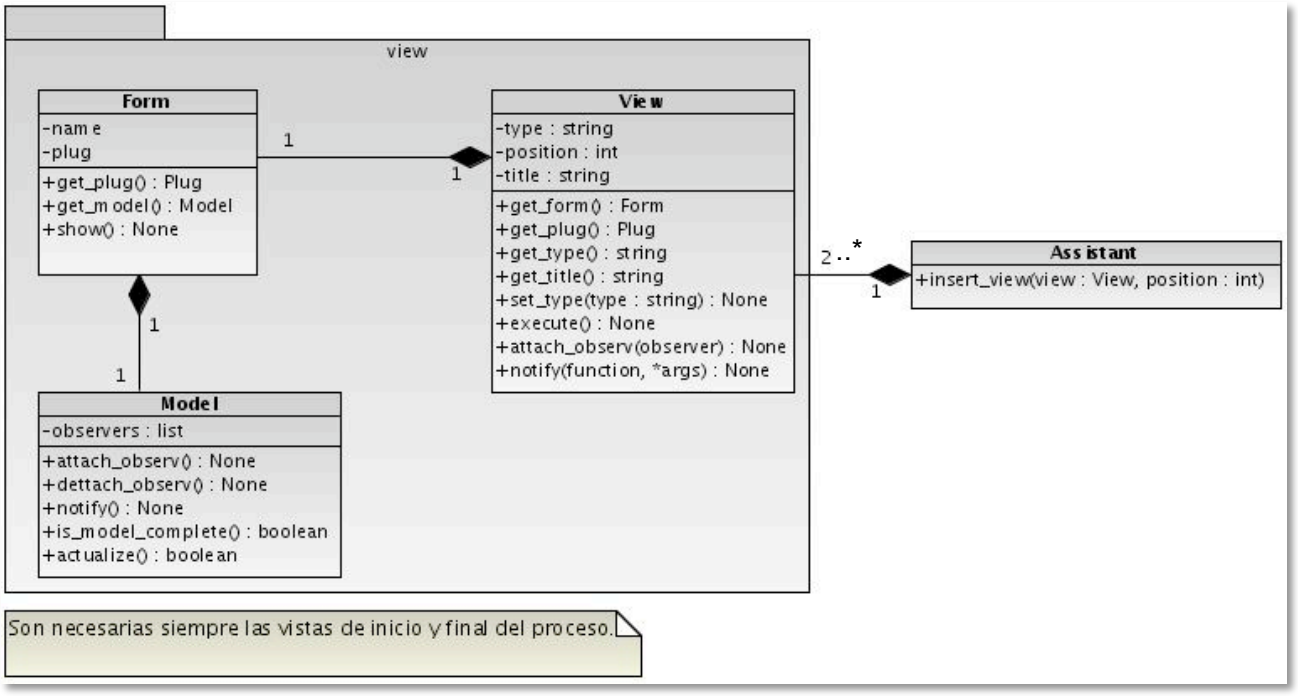

Diagrama de clases 7: Diseño de clases de Arare.

Anexo 2: Fragmentos de código.

```
@__change_model_signal_emit
def actualize(self, **kwargs):
    for key, value in kwargs.items():
        if hasattr(self, key):
            setattr(self, key, value)
        else:
            raise AttributeError, "%s has not attribute %s" % (self.__class__, key)
        self.is_model_complete()
    return True
```

Ejemplo de código 1: Función que actualiza el modelo y emite una señal especificando si está completo o no.

```
@__change_model_completion_emit
def is_model_complete(self):
    """ This function returns must return True is model are complete
    False if not. """
    raise NotImplementedError
```

Ejemplo de código 2: Función que encargada de responder si el modelo está completo o no. Si está completo, emite una señal indicándolo.

```
@requires(lambda self, obsever: isinstance(observer, Observer),
          "First argument must be an instance of Observer class")
def dettach_observ(self, observer):
    """ Attach an observer to model. """
    if observer in self.__obvs:
        self.__obvs.remove(observer)
```

Ejemplo de código 3: Función que añade un observador a un modelo.

```
def requires(predicate, help=""):
    def decorator(wrapped):
        def wrapper(*args, **kwargs):
            if predicate(*args, **kwargs):
                return wrapped(*args, **kwargs)
            else:
                aux = help or predicate.__doc__ or wrapper.__doc__
                raise PreconditionViolationError, "%s, (%s)" % (wrapper.__name__, help)
        return wrap(wrapper, wrapped)
    return decorator
```

Ejemplo de código 4: Implementación de la premisa “requires” del paradigma de programación por contrato.

```
def __set_attr(self, widget):
    setattr(self, widget.get_name(), widget)
    children = None
    try:
        children = widget.get_children()
    except:
        #If no children do nothing
        pass
    if children is not None:
        for child in children:
            self.__set_attr(child)
        return
    return
```

Ejemplo de código 5: Inicialización dinámica de los controles de un formulario.

```
def __initialize(self, glade_path):
    """ Setting all controls as attributes,
    connecting signals, etc... """
    self.__glade = gtk.glade.XML(glade_path, self.__name)
    self.__glade.signal_autoconnect(self)
    self.__set_attr(self.__glade.get_widget(self.__name))
```

Ejemplo de código 6: Inicialización de un formulario.

```
def execute(self):
    """ Here you must implement the vehaviour of your view. """
    raise NotImplementedError
```

Ejemplo de código 7: Función “abstracta” para la implementación del comportamiento de una vista determinada.