Sistemas Informáticos
Curso 2005/2006

Persia

Personalización inteligente del sistema operativo Linux basado en las preferencias y temperamentos de los usuarios

Julio Alberto Martínez Real
Alberto Pedrera Castela
Guillermo Ortiz Fernández

Dirigido por:
Prof. Héctor Gómez Gauchía
Dpto. Sistemas Informáticos y Programación

Facultad de Informática
Universidad Complutense de Madrid

Índice

1. Introducción

1.1 Descripción del Proyecto.

Los ordenadores se han convertido en una herramienta fundamental en nuestra vida, son nuestra principal herramienta de trabajo, entretenimiento, información, etc… Según se van sofisticando y complicando los sistemas informáticos se ha ido creando una necesidad real de personalización. Esta personalización es más evidente en los sistemas operativos, donde el mismo usuario se conecta todos los días.

Cada usuario presenta una relación diferente con el ordenador según su temperamento. Además este usuario cambia de estados de ánimo, humor, gustos, urgencia, etc… a lo largo de cada sesión de trabajo.

El proyecto consiste en construir una herramienta de fácil instalación, aprendizaje y uso que dé a los usuarios una forma de personalización de GUIs para que se adapten a sus gustos. Nos centraremos en personalizar escritorios Linux, más concretamente en escritorios Debian.

1.1 Project description

Computers play a fundamental role in our lives. They have become an important tool when it comes to work, to look for information or just to play. As computer systems get more and more complex a real necessity for customizing them arises. This necessity seems more evident in the case of operating systems since every particular user makes an intensive use of them day by day.

Every user has a different relationship with computers according to his personality. Moreover, a particular user is never the same. His mood, tastes and necessities may be subject to change.

This project aims to build a tool that is easy to install, learn and use and that provides users with a formula to customize GUIs so they can be adapted to their preferences. We shall focus here on Linux desktops customisation; on Debian desktops to be precise.

Descripción de las funciones generales.

El proyecto incluye un módulo para captar el tipo de usuario que tenemos.

Siguiendo la teoría de temperamentos de Keirsey, que consiste en hacer un simple test para determinar el perfil del usuario actual. Una parte de este perfil es el temperamento, que nos indica cómo el usuario maneja el lenguaje, cómo estructura el concepto tiempo y el espacio, cómo maneja las matemáticas, etc...

Basándose en este perfil de usuario los diferentes módulos que componen el programa harán los cambios pertinentes para ajustarse al usuario actual.

Para todo ello se usan sistemas de razonamiento basados en casos que partiendo de unos casos generales que representan personalidades extremas dentro del sistema se adapten a los gustos de la persona que esta utilizando el sistema.

Para representar todo este conocimiento dentro del sistema se hará utilizando ontologías.

1.2 Teorías aplicadas

Ontologías:

Aunque el concepto de ontología ha estado presente desde hace mucho tiempo en la filosofía, recientemente se utiliza en Informática para definir vocabularios que las máquinas puedan entender y que sean especificados con la suficiente precisión como para permitir diferenciar términos y referenciarlos de manera precisa.

El término ontología en informática hace referencia al intento de formular un exhaustivo y riguroso esquema conceptual dentro de un dominio dado, con la finalidad de facilitar la comunicación y la compartición de la información entre diferentes sistemas. Esta es la diferencia con, aunque toma su nombre de una analogía con, el significado filosófico de la palabra ontología.

Un uso común tecnológico actual del concepto de ontología, en este sentido, lo encontramos en la inteligencia artificial y la representación del conocimiento. En algunas aplicaciones, se combinan varios esquemas en una estructura de facto completa de datos, que contiene todas las entidades relevantes y sus relaciones dentro del dominio.

Los programas informáticos pueden utilizar así la ontología para una variedad de propósitos, incluyendo el razonamiento inductivo, la clasificación, y una variedad de técnicas de resolución de problemas.

Típicamente, las ontologías en los ordenadores se relacionan estrechamente con vocabularios fijos, una ontología de fundacional, con cuyos términos debe ser descrito

todo lo demás. Debido a que esto puede ocasionar representaciones pobres para ciertos dominios de problemas, se deben crear esquemas más especializados para convertir en útiles los datos a la hora de tomar decisiones en el mundo real.

De manera más precisa, extraído del Documento de requisitos de OWL una definición de ontología:

Una ontología define los términos a utilizar para describir y representar un área de conocimiento. Las ontologías son utilizadas por las personas, las bases de datos, y las aplicaciones que necesitan compartir un dominio de información (un dominio es simplemente un área de temática específica o un área de conocimiento, tales como medicina, fabricación de herramientas, bienes inmuebles, reparación automovilística, gestión financiera, etc.). Las ontologías incluyen definiciones de conceptos básicos del dominio, y las relaciones entre ellos, que son útiles para los ordenadores. Codifican el conocimiento de un dominio y también el conocimiento que extiende los dominios. En este sentido, hacen el conocimiento reutilizable.

OWL:

Las ontologías requieren de un lenguaje lógico y formal para ser expresadas. En la inteligencia artificial se han desarrollo numerosos lenguajes para este fin, algunos basados en la lógica de predicados, como KIF y Cycl que ofrecen poderosas primitivas de modelado, y otros basados en *frames* (taxonomías de clases y atributos), que tienen un mayor poder expresivo, pero menor poder de inferencia; e incluso existen lenguajes orientados al razonamiento como Description Logic y Classic. Todos estos lenguajes han servido para desarrollar otros lenguajes aplicables a la Web.

Para nuestro proyecto vamos a usar OWL que son las siglas de *Web Ontology Language.* o Lenguaje de Ontologías para la Web es un lenguaje de etiquetado semántico para publicar y compartir ontologías en la Web. Se trata de una recomendación del W3C, y puede usarse para representar ontologías de forma explícita, es decir, permite definir el significado de términos en vocabularios y las relaciones entre aquellos términos (ontologías). En realidad, OWL es una extensión del lenguaje RDF y emplea las tripletas de RDF, aunque es un lenguaje con más poder expresivo que éste.

Se trata de un lenguaje diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos. OWL surge como una revisión al lenguaje DAML-OIL y es mucho más potente que éste. Al igual que OIL, OWL se estructura en capas que difieren en la complejidad y puede ser adaptado a las necesidades de cada usuario, al nivel de expresividad que se precise y a los distintos tipos de aplicaciones existentes (motores de búsqueda, agentes, etc.).

OWL está diseñado para usarse cuando la información contenida en los documentos necesita ser procesada por programas o aplicaciones, en oposición a situaciones donde el contenido solamente necesita ser presentado a los seres humanos. Actualmente OWL tiene tres variantes:

- **OWL Lite** da soporte a aquellos usuarios que primordialmente necesitan una clasificación jerárquica y restricciones simples. Por ejemplo, soporta restricciones cardinales, pero solamente permite valores cardinales de 0 ó 1. Así pues, es más simple proveer herramientas de soporte para OWL Lite. OWL Lite ofrece una rápida ruta de migración para tesauros y otras taxonomías. En resumen, OWL Lite tiene una más baja complejidad formal que *OWL DL*.

- ***OWL DL*** da soporte a aquellos usuarios que quieren la máxima expresividad mientras conservan completamente la computacionalidad (todas las conclusiones son garantizadas para ser computables) y resolubilidad (todas las computaciones terminarán en tiempo finito). OWL DL incluye todos los constructores del lenguaje OWL, pero pueden usarse solamente bajo ciertas restricciones (por ejemplo, mientras una clase puede usarse por una subclase de muchas clases, una clase no puede ser una instancia de otra clase). OWL DL se denomina así debido a su correspondencia con las descripciones lógicas (DL), un campo de investigación que han estudiado los lógicos para la fundación formal de OWL.

- ***OWL Full*** da soporte a usuarios que requieren el máximo de expresividad y la libertad sintáctica de RDF sin garantías computacionales. Por ejemplo, en OWL Full una clase puede ser tratada simultáneamente como una colección de individuos y como un individuo por derecho propio. OWL Full permite a una ontología aumentar el significado del vocabulario predefinido (RDF ó OWL). Es poco probable que algún *software* racional pueda soportar por completo el razonamiento para cada característica de OWL Full.

Cada uno de estos sublenguajes es una extensión de su predecesor más simple, en los que ambos pueden ser expresados legalmente y en los que pueden ser válidamente concluidos. El conjunto siguiente de relaciones es correcto, sin embargo, no sus inversas.

- Cada ontología legal *OWL Lite* es una ontología legal *OWL DL*.
- Cada ontología legal *OWL DL* es una ontología legal *OWL Full*.
- Cada conclusión válida *OWL Lite* es una conclusión válida *OWL DL*.
- Cada conclusión válida *OWL DL* es una conclusión válida *OWL Full*.

Temperamentos de Keirsey:

Para averiguar los gustos del usuario que esta trabajando con el sistema hemos usado los test de temperamento de Keirsey. Se ha implementado dos cuestionarios.

- Temperamentos de Keirsey, cuestionario corto. Consiste en dieciséis preguntas con cuatro posibles respuestas cada una, hay que señalar la respuesta que más se

parece al usuario con uno, la que se parece un poco menos con un dos, la siguiente con un tres y la que no se parece la marcamos con un cuatro. Un ejemplo de pregunta sería:

Funcionaría mejor en un trabajo si me ocupara de
a) Herramientas y equipo
b) Desarrollo de recursos humano
c) Materiales y servicios
d) Sistemas y estructuras

Una posible respuesta podría ser marcar con un uno la respuesta c) con dos la respuesta b) con tres la respuesta d) y por último con cuatro la respuesta a).

- Temperamentos de Keirsey, cuestionario largo. Son setenta pregunta con dos posibles respuestas cada uno. Para rellenar el test hay que elegir una de las dos. Un ejemplo de pregunta sería:

  En la mayor parte de las situaciones eres más.
  a) Deliberado que espontáneo
  b) Espontáneo que deliberado

Keirsey no habla de personalidad, sino de temperamentos, y explica que éstos están dados en base a cómo la gente recopila información del medio ambiente (que puede ser con los sentidos (S) o a través de la intuición (N)) y cómo la procesa (que puede ser o de manera Racional (T), a través de los sentimientos (F), emitiendo juicios (J) o a través de la percepción (P)). La combinación de la recopilación y el proceso que se tiene de la información da 4 tipos de temperamentos básicos, que se pueden resumir de la siguiente manera:

- **Temperamento SJ o guardianes:** Son personas leales al sistema, se rigen por el deber, son muy confiables, tienen resistencia al cambio ya que conservan tradiciones, son precisos y son aquellos que dicen: “Si no está roto, no lo arregles”

- **Temperamento SP o artistas:** Se considera gente de espíritu libre, están orientados a procesos, son buenos en situaciones de crisis, son impulsivos, tienen una alta necesidad de libertad y de espacio, son flexibles. Disfrutan el momento y son espontáneos. Aplican la frase: “Las ideas generales y abstractas son la fuente de los más grandes errores de la humanidad”.

- **Temperamento NT o idealistas:** Es gente que logra sus metas, son personas independientes, curiosos intelectualmente, no son conformistas, es gente basada en principios, y son arquitectos de cambio, se preguntan constantemente: “Qué pasaría si....”

- **Temperamento NF o racionales:** Es gente que cuenta con habilidades interpersonales y puede ser considerado un seductor por su capacidad para convencer. Es gente que es un apoyo para otros, son simpáticos y cuentan con una imaginación vívida. Son hipersensitivos al conflicto, están en una constante

búsqueda de sí mismos, necesitan de motivación y reconocimiento y son personas que quieren “Llegar a ser”.

Una vez rellenado el cuestionario corto de Keirsey aparecerá el porcentaje de cada temperamento que compone la personalidad del usuario. Utilizando estos porcentajes el sistema hará los cambios pertinentes en el mismo para que se ajuste a sus gustos.

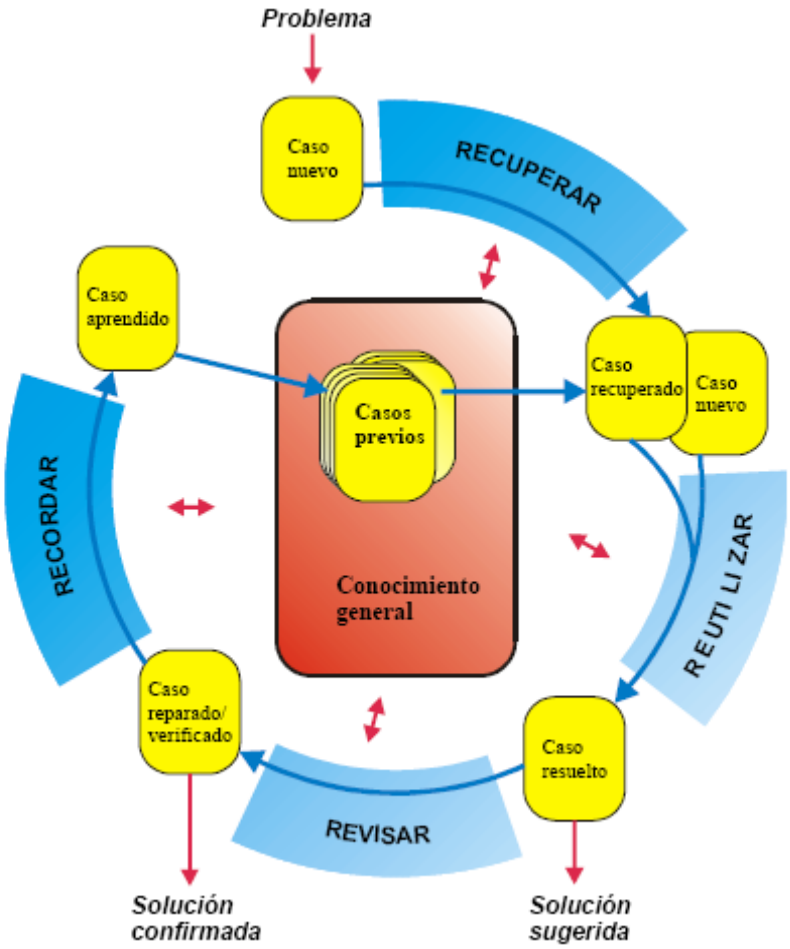
CBR

El CBR (Razonamiento basado en casos) es una tecnología para la construcción de sistemas expertos alternativa a los sistemas basados en reglas, los sistemas basados en reglas se basan en: hechos conocidos sobre el dominio, las reglas son de la forma if-then. El cuello de botella en la construcción de sistemas expertos basados en reglas radica en la obtención del conocimiento, porque es difícil trasladar el conocimiento del experto a reglas aplicables para nuestro programa.

En sistemas CBR es más fácil adquirir conocimientos ya que se pueden proponer soluciones en dominios que no se comprenden del todo, resulta más sencillo adquirir nuevos casos que descubrir reglas y generalizaciones nuevas, es más rápido reutilizar una solución previa que obtener la solución desde cero, los casos ayudan a un razonador a concentrarse en los aspectos importantes de un problema, al identificar las características definitorias, mantenimiento de la base de conocimiento: los usuarios pueden añadir nuevos casos sin ayuda de los expertos.

El CBR es especialmente adecuado en dominios poco formalizados y donde el aprendizaje juega un papel preponderante.

Un razonador basado en casos resuelve nuevos problemas adaptando las soluciones que fueron utilizadas para resolver problemas previos similares.



La figura muestra el esquema general de un ciclo CBR. Que nosotros tomaremos como base para construir el ciclo CBR para nuestro sistema. Para hacer esto hay que tener en cuenta que:

¿Qué es un caso?

"Un caso es un fragmento contextualizado de conocimiento que representa una experiencia y que enseña una lección importante para conseguir los objetivos del razonador" [Kolodner & Leake 97]

No todas las situaciones "enseñan una lección" puede haber casos redundantes o cubiertos por el conocimiento general.

La descripción de la situación o el problema debe estar compuesta por los objetivos, restricciones para la consecución de los objetivos y por las características de la situación.

A la hora de conseguir el resultado hay que comprobar si tuvo éxito o no y en caso de fallo hay que descubrir porque falló y que estrategias de reparación se siguió.

El conocimiento de adaptación incluye lo que hay que cambiar en el caso recuperado para que se llegue a una solución que tenga éxito.

Pasos a seguir en la recuperación

1. Valoración de la situación. Determinar las características que permiten encontrar casos relevantes.

2. Búsqueda en la memoria para encontrar los casos que guardan una similitud por encima de un cierto umbral. Comparación superficial. El procedimiento de búsqueda dependerá de la organización (la estructura de datos de los casos (lista, árbol, redes,....)). En ocasiones los casos se organizan automáticamente utilizando métodos de aprendizaje máquina.

3. Ordenación (ranking) de los casos recuperados. Comparación más elaborada.

4. Selección del caso mejor.

Se utiliza el conocimiento incluido en el caso recuperado para resolver/clasificar el problema/situación actual. Si no se modifica el caso recuperado, la solución es válida o es el usuario quien se encarga de adaptarla o interpretarla. Si se modifica el caso recuperado se adapta la solución (resolución de problemas) o la justificación (interpretación de situaciones) de los casos recuperados.

¿Es correcta la solución propuesta? La evaluación de la solución normalmente se realiza fuera del sistema CBR, respuesta de un experto (o del usuario).

En los sistemas CBR el razonamiento y el aprendizaje están íntimamente ligados, un sistema CBR mejora con el uso al ir adquiriendo nuevas experiencias que integra adecuadamente, mejora la eficiencia del sistema al disponer de más casos a partir de los cuales obtener soluciones.

1.3 Mapa de los trabajos realizados por cada autor

-Investigación previa de linux, kde, GNOME... (Julio, Alberto, Guillermo)
-Codificación del cuestionario corto (Alberto)
-Interfaz gráfico del cuestionario corto (Alberto)
-Codificación del cuestionario largo (Julio)
-Interfaz gráfico del cuestionario largo (Julio)
-Codificación de los cambios internos del programa (Guille)
-Codificación de los cambios en linux (Alberto)
-Codificación de la priorización de procesos (Julio)
-Codificación de la interfaz gráfica (Guillermo)
-Codificación del conector genérico (Alberto)
-Codificación de los conectores particulares para cada ontología (Julio, Alberto)
-Codificación de las clases del User (Julio)
-Codificación del UserType (Alberto)
-Introducción del orden en los elementos de las ontologías (Guillermo)
-Codificación de las clases de las variaciones (Julio)
-Codificación de las demás clases de los conectores (Julio, Alberto)

-Codificación de las clases del ciclo cbr (Guillermo)
-Codificación de los ficheros de configuración (Guillermo)
-Pruebas y remodificación de los conectores (Alberto, Julio)
-Pruebas cbr y remodificación del cbr (Guillermo)
-Pruebas globales (Guillermo, Julio, Alberto)
-Clases que cargan datos en la ontología (Guillermo)
-Codificación del clone en todas las clases (Julio)
-Introducción de los cambios de tema (Guillermo, Alberto)
-Codificación de los toString de todas las clases para facilitar la depuración (Julio)
-Nuevas pruebas y remodificación de conectores a medida que se añaden nuevas prestaciones a la aplicación (Julio, Alberto)
-Introducción de canciones (Guillermo)
-Realización de los javadoc (Cada uno el de sus clases)
-Clases auxiliares de Themes y música (Guillermo)
-Investigaciones, documentarse acerca de todo lo referente al proyecto (Julio, Alberto, Guillermo)

MEMORIA
-Punto uno (Alberto)
-Punto 1.3 (Julio)
-Punto 2 (Guillermo)
-Punto 3 (Alberto)
-Punto 4.1 (Julio)
-Punto 4.2 (ciclo global) (Guillermo)
-Punto 4.2 (componentes y clases principales)(Julio)
-Punto 4.2 (esquema) (Alberto)
-Punto 5 (Realización de las encuestas) (Julio)
-Punto 5 (Extraer conclusiones, realizar gráficas...) (Julio)
-Punto 6.1 Manual de usuario (Guillermo)
-Punto 6.2 Manual del sistema y pruebas (Alberto)
-Anexo Cuestionario corto (Alberto)
-Anexo Cuestionario largo (Julio)

1.4 Herramientas usadas para el desarrollo

Eclipse

El sistema esta programado en Java y utilizamos Eclipse como IDE de programación.

Eclipse es una estructura (workbench) que puede soportar distintas herramientas de desarrollo y para cualquier lenguaje. Fue desarrollado por IBM bajo licencia Common Public License Version 1.0 ("CPL"), es totalmente extensible, pudiendo hacer tus propios plug-ins (un plug-in es un desarrollo a parte de la plataforma de eclipse que aportan una determinada funcionalidad) o utilizar los que ya están desarrollados.

Elegimos eclipse por su facilidad de uso y por adaptarse perfectamente a nuestras necesidades.

CVS

Para poder trabajar todos los integrantes del grupo a la vez usamos un CVS.
Un CVS (Concurrent Versions System) es un sistema de control de versiones: mantiene el registro de todo el trabajo y los cambios en la implementación de un proyecto (de programa) y permite que distintos desarrolladores (potencialmente situados a gran distancia) colaboren. CVS se ha hecho popular en el mundo del software libre. Sus desarrolladores difunden el sistema bajo la licencia GPL.

CVS utiliza una arquitectura cliente-servidor: un servidor guarda la(s) versión(es) actual(es) del proyecto y su historia, y los clientes conectan al servidor para sacar una copia completa del proyecto, trabajar en esa copia y entonces ingresar sus cambios. Típicamente, cliente y servidor conectan utilizando Internet, pero cliente y servidor pueden estar en la misma máquina si CVS tiene la tarea de mantener el registro de la historia de las versiones del programa de un proyecto solamente con desarrolladores locales. El servidor normalmente utiliza un sistema operativo similar a Unix, mientras que los clientes CVS pueden funcionar en cualquier de los sistemas operativos más difundidos.

Varios clientes pueden sacar copias del proyecto al mismo tiempo. Posteriormente, cuando ingresan sus modificaciones, el servidor trata de fundirlas. Si esto falla, por ejemplo debido a que dos clientes tratan de cambiar la misma línea en un archivo en particular, entonces el servidor deniega el segundo ingreso e informa al cliente sobre el conflicto, que el usuario deberá resolver manualmente. Si la operación de ingreso tiene éxito, entonces los números de versión de todos los archivos implicados se incrementan automáticamente, y el servidor CVS escribe una línea de descripción suministrada por el usuario, la fecha y el nombre del autor y sus archivos log.

Los clientes pueden también comparar diferentes versiones de archivos, solicitar una historia completa de los cambios, o sacar una "foto" histórica del proyecto tal como se encontraba en una fecha determinada o en un número de revisión determinado.

Muchos proyectos de código abierto permiten el "acceso de lectura anónimo", significando que los clientes pueden sacar y comparar versiones sin necesidad de teclear una contraseña; solamente el ingreso de cambios requiere una contraseña en estos escenarios.

Los clientes también pueden utilizar la orden de actualización con el fin de tener sus copias al día con la última versión que se encuentra en el servidor. Esto elimina la necesidad de repetir las descargas del proyecto completo.

CVS también puede mantener distintas "ramas" de un proyecto. Por ejemplo, una versión difundida de un proyecto de programa puede formar una rama, utilizada para corregir errores, mientras que una versión actualmente en desarrollo, con cambios mayores y nuevas características, pueden formar una rama separada.

Nosotros nos decidimos a usar un repositorio CVS gratuito de 2MB que ofrece cvsdude.org sus datos son: host: cvsdude.org, path: /cvs/stdc, tipo conexion: pserver. Como 2MB es muy poco espacio solo subimos el código otras partes del sistema de más tamaño como librerías, ontologías, etc. se descargaban aparte.

Reuniones

Para repartirnos las tareas a realizar entre los distintos miembros del grupo, analizar los problemas y las posibles soluciones en el desarrollo del sistema, las reuniones entre los integrantes tenían una periodicidad de una semana. En dichas reuniones se llegaba a un acuerdo para la asignación las tareas y se analizaba lo hecho hasta ahora.

Con el profesor que ha dirigido el proyecto las reuniones eran también cada semana hasta que se hubo asentado la base sobre la que trabajar, una vez hecho esto las reuniones se fueron espaciando en el tiempo.

2. Investigación

Para el desarrollo del proyecto se han realizado investigaciones en diferentes ramas que se explicarán a lo largo de este apartado.

2.1 Investigaciones de Linux

El primer contacto que hemos tenido con este proyecto ha sido ampliar nuestro conocimiento de Linux. Actualmente, Linux posee gran variedad de distribuciones, aunque todas ellas basadas en el mismo núcleo. Por ello las mayores diferencias que podemos encontrar son las utilidades y el entorno gráfico que cada distribución nos aporta.

Las distribuciones que hemos probado para ver cual era la que mejor se adaptaba a nuestro propósito han sido: Suse, Mandrake, Fedora y Debian. Dependiendo de la distribución suelen tener uno de los entornos gráficos GNOME, KDE o ambos. Cada uno de esos GUI posee una manera diferente de guardar la configuración, aunque los dos se basan en ficheros XML o texto plano de configuración. Más tarde se comentará algo respecto a estos ficheros de configuración.

El primer paso es hacer una lista con los posibles cambios que podemos considerar interesantes como: cambio de color de fondo, cambio de escritorio, música, etc. Una vez detectadas las variaciones habrá que detectar que ficheros de configuración se usan para definir estos parámetros o con que programas podemos cambiar estos valores de una forma rápida y sencilla.

El número de cambios es prácticamente ilimitado y no se tiene porque basar solamente en cambios del entorno gráfico. Gracias a la arquitectura de la aplicación podremos lanzar prácticamente cualquier tipo de aplicación. Por ello la investigación de las posibles variaciones no se ha basado solamente en cambios visuales. Los cambios podrían ser simplemente lanzar programas Java generados específicamente u otras aplicaciones.

Una lista de los posibles cambios que se barajaron fueron:

Lista de posibilidades
Cambiar fondo de pantalla Cambiar icono del ratón
Cambiar iconos de los programas a un tema que le guste al usuario
Cambiar la velocidad del ratón según la experiencia y edad del usuario
Poner una música al inicio de sesión que le guste al usuario
Sacar cada un tiempo aleatorio una cita, chiste, comentario, ayuda, etc.

Cambiar el tipo de letra según si el usuario tiene tendencias autísticas o más tecnológicas.
Personalizar los colores de las ventanas según los gustos del usuario.
Aumentar la prioridad de los programas que más pudiera usar un usuario.
Cambiar fondo de pantalla periódicamente según los gustos.
Personalizar los mensajes de inicio de sesión según el tipo de usuario.
Personalizar la barra de herramientas según el tipo de usuario.
Mostrar solo los iconos de los programas más usados
Mostrar solo los iconos de los programas que creamos que le podrían ser útiles a un usuario determinado.
Que el ratón tenga efectos Avise del tiempo que va a hacer hoy Nos diga el santo que es hoy Nos felicite en nuestro cumpleaños Se despida cuando lo vayamos a apagar Nos avise en fechas importantes
Sugerencias de que comer hoy según el tiempo que va a hacer
Sugerencias de que comer hoy según la temporada del año
Sugerencias de que hacer hoy (cine, teatro, tv)
Recomendar una película según los gustos de una base de datos.
Recordar fechas como martes 13, día de los inocentes.

Si nos fijamos en todas las variaciones prácticamente todas se pueden realizar con el uso de programas externos Java o no y modificando ficheros de configuración de Linux.

2.1.1 Configuración de KDE

KDE mantiene las propiedades en ficheros de propiedades de texto plano. El primer punto importante es que no son tan sencillos como los típicos ficheros de propiedades ya que poseen ciertas peculiaridades, esto es un impedimento ya que no se pueden modelar directamente con la clase Property que nos proporciona Java.

Uno de los rasgos más significativos es que se permiten repeticiones de los pares clase/valor y se usan etiquetas especiales para agrupar los atributos en secciones. Esto hace que se pueda encontrar la clave COLOR en el mismo fichero de propiedades refiriéndose al fondo de pantalla o al fondo de las ventanas, esta es la principal razón de que no se pueda usar la clase Property al haber repeticiones.

Nota: Se desarrolló una pequeña aplicación capaz de tener en cuenta estos detalles.

Los ficheros de propiedades más significativos son:

- Kdeglobals: dentro de este fichero podemos encontrar pares clave/valor para cambiar el color de fondo del escritorio, pantallas, tamaño de iconos, color de botones, etc.
- Kdesktoprc: desde aquí podremos gestionar opciones de salvapantallas, de escritorio, etc.

Con estos dos ficheros cubrimos prácticamente todas las posibles variaciones gráficas. En la misma localización donde se encuentran estos ficheros podemos encontrar otros, siempre con el prefijo “KDE”. Todos estos ficheros cambian diferentes aspectos del entorno gráfico, aunque los cambios soy demasiado concretos y no se aprecian claramente.

2.1.2 Configuración de GNOME

Gnome es otro entorno gráfico ampliamente utilizado, viniendo conjuntamente en casi todas las distribuciones de linux junto a KDE. También es importante en GNOME es mucho menos pesado que KDE por lo que no consume tantos recursos como el anterior.

Las versiones posteriores de GNOME usaban ficheros de configuración .INI, este sistema es válido para prácticamente todos los entornos. Pero en la actualidad se ha cambiado este formato para pasar a uno que aporte mayor flexibilidad y que puedan soportar aplicaciones ejecutadas en red.

Para solventar esta situación, era necesario que el sistema de configuración de las aplicaciones fuera fácilmente accesible desde cualquier máquina dentro de una red, de forma que un usuario tenga la misma configuración tanto en su ordenador personal, como cuando abre una sesión con su usuario en el servidor de la oficina. Esto sólo se podía solucionar con una herramienta de configuración distribuida, y eso es lo que hace GConf.

Según la propia definición del autor, GConf "es un sistema para almacenar información de configuración, lo que comúnmente se conoce por parejas clave/valor". Además, destaca también la flexibilidad de su arquitectura, que permite cambiar fácilmente el almacén de datos (es decir, la manera y el lugar en que se almacena la información), o incluso usar varios a la vez, o la posibilidad de que cada entrada en la

base de datos (o sea, cada clave), tenga asociado un texto descriptivo, lo que facilita enormemente la labor del usuario, pues cada entrada está perfectamente documentada.

GConf forma parte del proyecto GNOME; está basado en CORBA, lo cual permite el acceso totalmente transparente a la configuración de las aplicaciones, tanto en entornos locales como distribuidos. Su arquitectura está basada en tres elementos principalmente:

- "backends": son librerías dinámicas que se insertan en GConf (plugins) para permitir el almacenamiento de datos en distintos formatos.
- gconfd: este el demonio de GConf, que se activa cuando un cliente se conecta al sistema.
- Clientes: son aplicaciones que hacen uso de las librerías provistas por GConf para acceder al demonio gconfd.

Finalmente para realizar cambios en los ficheros de configuración bajo GNOME bajo consola se puede utilizar el comando gconftool. Este comando es de gran interés, ya que nos permite modificar cualquier par clave/valor con un único comando. Este comando, es el que usamos para realizar todas las modificaciones de GNOME.

Un ejemplo de este comando para cambiar el color del fondo del escritorio sería:

gconftool --set "/desktop/gnome/background/primary_color" --type string “#000000"

2.2 Investigaciones de Ontologías

Las ontologías nos permite representar conocimiento y realizar razonamientos y clasificaciones una vez estén definidas. Las ontologías nos dan muchas más posibilidades como: rehusar el conocimiento, compartir conocimiento entre aplicaciones heterogéneas, analizar el conocimiento del dominio, etc.

Actualmente las ontologías se están usando en diversos campos como el comercio electrónico, portales semánticos, indexación de páginas web, procesamiento de lenguaje natural, etc.

Toda la información se ha almacenado en ontologías por lo que hemos necesito investigar la manera de leer/escribir/modificar de una manera sencilla.

Existen herramientas específicas que han salido en los últimos años para el desarrollo específico de ontologías entre los que se encuentran:

- Ontology Server
- WebODE
- Protegé

- WordNet
- …

Cada uno de estos tiene sus ventajas y desventajas. Se ha optado por Protegé ya que teníamos conocimientos previos. Este editor tiene varias ventajas que hemos creído importantes. Posee soporte tanto para Linux como para Windows. El proyecto es libre y se actualice con bastante regularidad y poseen un foro con bastante movimiento donde poder solucionar las dudas. Con este programa se han generado las ontologías necesarias.

Otro punto importante es que lenguaje usar para el almacenamiento de las ontologías. Protegé permite guardar las ontologías en varios formatos. El formato a elegir es un punto fundamental, ya que nos ligará fuertemente al software que podremos usar para manejar estos ficheros, ya sea en programas para razonamiento, desarrollo, framework desde Java, etc.

Algunos de los formatos que existen actualmente son:

- EXPRESS
- CML
- KIF
- Loom
- XML
- OWL
- …

Cada uno de estos lenguajes tienes sus ventajas, nosotros hemos elegido OWL ya que teníamos un pequeño conocimiento del mismo.

Los ficheros OWL tienen una estructura etiquetada de la misma forma que los XML, y aunque se pueden modificar manualmente o ver su contenido con un editor de textos, la cantidad de datos adicionales que guardan es bastante grande, por lo que se hace necesario el uso de un editor específico como Protegé.

Teniendo ya definido el estándar en el que vamos a almacenar las ontologías y un editor con el que modificarlos fácilmente, nos falta un framework con el que manejar fácilmente las ontologías desde Java. En este caso hemos optado por Jena.

Jena es un framework que nos da las clases necesarias para la modificación de las mismas desde un entorno Java. Además la comunidad de Jena se mantiene bastante activa en la actualidad habiendo foros y material con el que documentarse.

Otras herramientas en las que nos hemos interesado en menor medida del desarrollo han sido Junit, JColibri, Pellet, CVS.

3. Casos de uso

CASO DE USO Nº 1

Nombre: Adaptación de interfaz automática de usuario.
Objetivos: Adaptar aspectos del interfaz de Linux para un usuario existente según su temperamento.

Entrada: Usuario de la aplicación
Precondiciones: El usuario existe en el sistema y ha rellenado en otra sesión el formulario 1.

Salida: Adaptación del sistema
Postcondiciones si éxito: El sistema se adaptará al usuario.
Postcondiciones si fracaso: Mensaje informativo del problema que ha ocurrido.

Actor: Usuario que se conecta.

Secuencia de ejecución normal:

1. Crear "modelo" de la aplicación (AccesoModeloDatos).
2. Leer los archivos de configuración del sistema.
3. Conectar con las ontologías.
4. Crear GuiPrincipal
5. Insertar usuario que se va a conectar al sistema.
6. Insertar estado de ánimo y personalización "Automática".
7. Ejecutar ciclo CBR, para el usuario actual. (Más detallado en otro Caso de Uso).
8. Preguntar si usuario satisfecho, si está satisfecho FIN. En otro caso Paso9
9. Automatización manual del usuario
10. Ejecutar ciclo CBR, para usuario actual y tipo personalización "Manual"

Secuencia alternativa:

- Si se produce algún error en el proceso se sacará un mensaje de error explicativo de lo que ha ocurrido. Los pasos más sensibles a la aparición de errores son 7, 8, 9, 10.

Nota: En un principio da igual si el usuario existe previamente en la aplicación o no. Esto se tratará internamente dentro del ciclo CBR.

CASO DE USO Nº2

Nombre: Ciclo CBR para usuario existente
Objetivos: Realizar la similitud de un usuario, adaptación y ejecución de los cambios. Finalmente se actualizará el usuario.

Entrada: Usuario sobre el que realizar la adaptación y tipo de adaptación (Automática, manual, etc...)
Precondiciones: El usuario existe en el sistema.

Salida: Adaptación del sistema
Postcondiciones si éxito: El sistema se adaptará al usuario.
Postcondiciones si fracaso: Mensaje informativo del problema que ha ocurrido.

Actor: Usuario que se conecta.

Secuencia de ejecución normal:

11. Se saca el usuario de la ontología.
12. Ejecución completa de ciclo CBR para usuario existente (ejecutarCicloCBRUsuarioExistente)
    - Se realiza una adaptación de los cambios que posee su UserType. (adaptacionVariacionesUsuario)
    - Se ejecutan los cambios adaptados para el usuario (ejecutarAdaptaciones)
    - Se devuelve un booleano indicando si el proceso ha tenido éxito.

Secuencia alternativa:
- Si se produce algún error se sacará información explicativa de lo que ha ocurrido.

CASO DE USO Nº3

Nombre: Ciclo CBR para usuario NO existente
Objetivos: Crear un nuevo usuario. Realizar el ciclo CBR y preguntar al usuario si está satisfecho. En caso de que no, realizar una realimentación del sistema con las opciones que nos da el usuario.

Entrada: Usuario sobre el que realizar la adaptación y tipo de adaptación (Automática, manual, etc...)
Precondiciones: El usuario NO existe en el sistema y acaba de rellenar el test1.

Salida: Adaptación del sistema
Postcondiciones si éxito: El sistema se adaptará al usuario. Se creará un nuevo usuario en el sistema.
Postcondiciones si fracaso: Mensaje informativo del problema que ha ocurrido.

Actor: Usuario que se conecta.

Secuencia de ejecución normal:

13. Ejecución para ciclo CBR de un usuario inexistente. (ejecutarCicloCBRUsuarioNoExistente)
    - Ver que el usuario no existe en el sistema

- Crear un nuevo User, con el nombre y valores del test. (crearNuevoUser)
- Crear un nuevo UserType por similitud (getUserTypePorSimilitud)
    i. Comprobar que hay un UserType suficientemente similiar y devolverlo.
    ii. En caso contrario crear uno nuevo y devolverlo.
- Asignar el UserType al User actual.
- Adaptar el User al UserType (adaptacionVariacionesUsuario)
- Ejecutar las adaptaciones (ejecutarAdaptaciones)
- Preguntar si usuario satisfecho, si está satisfecho FIN en caso contrario..
- …. USER realiza sus variaciones manualmente…..
- Reparar caso de uso con la opinión del usuario (repararCasoConOpinionUsuario)
- Guardar caso de uso final.

Secuencia alternativa:

- Si se produce algún error se sacará información explicativa de lo que ha ocurrido.

CASO DE USO Nº4

Nombre: Calculo de similitud. (getUserTypePorSimilitud)
Objetivos: Dado un usuario con sus porcentajes correspondientes. Encontrar el usertype que más se ajusta. Si no hay uno suficientemente bueno se creará uno nuevo y se insetará en la ontología.

Entrada: User sobre el que comprobar la similitud
Precondiciones: El User de entrada debe de tener nombre y porcentajes correctos.

Salida: UserType que mejor se adapta al User.
Postcondiciones si éxito: UserType que mejor se adapta al User actual.
Postcondiciones si fracaso: Devuelve false.

Actor: ---

Secuencia de ejecución normal:

14. Obtenemos todos los UserType del sistema
15. Recorremos todos los UserType del sistema
    - Para cada uno calculamos en valor absoluto el que mejor se adapta
    - Guardamos el que mejor se adapta de los procesados
16. Si el mejor UserType cumple el “umbral” lo devolvemos.
17. Si el mejor UserType NO cumple el umbral se crea uno nuevo
    - Se crea un nuevo UserType con los porcentajes del User
    - Se inserta en la ontología.
18. Se devuelve el UserType creado.

Secuencia alternativa:

- Si se produce algún error se sacará información explicativa de lo que ha ocurrido.

CASO DE USO Nº5

Nombre: Adaptar UserType a User (adaptacionVariacionesUsuario)
Objetivos: Dado un usuario con un UserType, realizar la adaptación del User teniendo en cuenta la diferencia de porcentajes que hay en cada campo.

Entrada : User sobre el que adaptar. UserType para el que adaptar
Precondiciones: Tanto el User como el UserType deben estar correctamente formados.

Salida: Vector con las adaptaciones.
Postcondiciones si éxito: Tenemos un Vector con las adaptaciones del User al contrastarlo con un UserType.
Postcondiciones si fracaso: Devuelve null.

Actor: ---

Secuencia de ejecución normal:

19. Obtenemos todas las variaciones del UserType
20. Calculamos las diferencias de porcentajes entre el User y el UserType
21. Recorremos todas las adaptaciones
    - Según la diferencia damos un valor u otro entre los posibles valores que tiene una adaptación
    - Se inserta la adaptación en el Vector.

Secuencia alternativa:

- Si se produce algún error se sacará información explicativa de lo que ha ocurrido. El Vector contentra null.

Nota: Como entrada podríamos recibir sólo un User. Se consideraría el UserType que contiene el usuario. Actualmente recibe como parámetro de entrada un User y un UserType, esto permite realizar adaptaciones de User sobre UserType que no les corresponderían en un principio por similitud. Como valor devuelto, da un Vector de adaptaciones. Estas adaptaciones deberían ser String con los comandos a ejecutar u objetos fácilmente convertibles para formar los comandos finales.

CASO DE USO Nº 6

Nombre: Realizar adaptación manual.
Objetivos: Realizar adaptaciones manuales en el sistema.

Entrada: Adaptación a realizar.
Precondiciones:

Salida: Adaptación del sistema
Postcondiciones si éxito: Se realizará la adaptación solicitada.
Postcondiciones si fracaso: Mensaje informativo de lo ocurrido.
Actor: Usuario que realiza la petición de la adaptación.

Secuencia de ejecución normal:

22. El usuario accede a la pestaña de configuración manual.
23. Selecciona la variación que desee que se realice.
24. Realiza la ejecución de la variación que se desee.
25. Se realiza un modificación de las variaciones que tiene predefinidas el usuario en el ciclo CBR
26. Se guarda el usuario en la ontología con las nuevas variaciones.

Secuencia alternativa:

- Si se produce algún error en el proceso se sacará un mensaje de error explicativo de lo que ha ocurrido.

4. Diseño e Implementación

4.1 Tecnologías

Los criterios para la selección de las tecnologías a utilizar han sido variados y se exponen a continuación.

Como nuestra intención desde el principio no es crear una aplicación terminada, sino una base para futuros desarrolladores, hemos elegido tecnologías de programación que sean lo más comunes que se pudiera. Así elegimos Java que es uno de los lenguajes más utilizados en este momento, orientado totalmente a objetos y gratuito. Éste último punto es importante ya que en casos de dudas entre dos tecnologías siempre preferíamos la gratuita para no obligar a futuros desarrolladores a depender de licencias.

Así pues las tecnologías utilizadas, con una breve introducción de sus características, son las que siguen:

Java

Según define SUN, creador de Java a este lenguaje de programación Java es: "simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico".

Aparte de ser estas características la lógica lista de piropos que un padre siempre daría a uno de sus hijos mas aplicados, efectivamente describen bastante bien a Java.

Para escribir un programa en Java primeramente se usa el lenguaje de programación Java. A continuación se compila el código (con javac), para generar un código, que ni es código fuente ni código máquina, conocido como bytecode, que es lo que finalmente ejecutará una Máquina Virtual Java (JVM) residente en el dispositivo de destino.

La sintaxis de Java toma prestado mucho de C y C++, pero elimina herramientas de muy bajo nivel como los punteros, y cuenta con un modelo de objetos muy simple. Cada objeto se encuentra dentro de la zona de memoria que se le asigna, y todas las variables de tipo objetos son referencias.

Java tiene algunos tipos primitivos que no son objetos, pero cualquier otra cosa debe ser un objeto. Sin embargo las librerías estándar de Java proporcionan un objeto envoltura por cada tipo primitivo, y desde la versión 1.5 el compilador de Java inserta de forma automática una conversión entre el objeto envolvente y su correspondiente tipo primitivo. La gestión de memoria es realizada por el recolector de basura (garbage collector) en tiempo de ejecución de forma automática, siendo una de las grandes bazas a favor, y en contra de Java.

Por ser un lenguaje multiplataforma, conocido por todos los miembros del equipo de desarrollo del proyecto y orientado a objetos éste ha sido el lenguaje de programación utilizado para desarrollar el programa.

Java, además, nos ha provisto de distintas clases y herramientas muy útiles a la hora de programar y realizar tareas rutinarias.

También un punto a favor es el javadoc, que ha sido utilizado para generar una especie de API de las clases de la aplicación.

Pellet

Pellet es un razonador de OWL DL basado en Java. Puede ser usado con Jena, API's de OWL y también proporciona interfaces DIG. La API de Pellet facilita funcionalidades para validar las consistencias de las ontologías, clasificar las taxonomías y responder a un conjunto de consultas RDQL.

Pellet esta basado en algoritmos de tableaux para expresar lógicas descriptivas. Proporciona una expresividad completa para OWL DL incluyendo razonamiento sobre nominales (clases enumeradas). Por tanto, las constructoras OWL owl:oneOf and owl:hasValue pueden ser usadas libremente. En la actualidad, Pellet es el primero y único razonador DL completo que puede capturar esta expresividad. Pellet asegura solidez y completitud.

Jena

Jena es un framework desarrollado por HP Labs para implementar aplicaciones en Java que manipulen metadata y ontologías. Jena proporciona clases Java para representar modelos, recursos, propiedades y literales. Por tal motivo, las interfases principales de la API son: Model, Resource, Property y Literal.

Jena proporciona métodos para crear, leer y escribir modelos RDF en formato XML.

Hemos usado los métodos que proporciona Jena para que las ontologías creadas con un editor externo al sistema (protege) puedan ser cargadas y manipuladas en clases Java. Por lo tanto Jena ha sido un mediador entre las clases Java que contienen la base de conocimiento del sistema y las ontologías que van a almacenar ese conocimiento.

Protégé

Protégé es un editor de ontologías y de bases de conocimiento de código abierto y libre.

Protégé utiliza estructuras que modelan conocimiento y acciones que apoyan la creación, la visualización, y la manipulación de ontologías en varios formatos de representación.

La plataforma Protégé proporciona dos modos principales de modelar ontologías:

- El editor Protégé-Frames permite a usuarios construir y rellenar las ontologías que son basadas en frame conforme al protocolo de Conectividad de Base de Conocimiento Abierto (OKBC). En este modelo, una ontología consiste en un conjunto de clases organizadas en una jerarquía de subsuposición para representar los conceptos sobresalientes de un dominio, un conjunto de ranuras asociadas a clases para describir sus propiedades y relaciones, y un conjunto de instancias de estas clases - los ejemplares individuales de los conceptos que sostienen valores específicos para sus propiedades.

- El editor de Protégé-OWL permite a usuarios construir ontologías para la Web Semántica en particular en el lenguaje de ontologías web (OWL) del W3C. Una ontología OWL puede incluir las descripciones de clases, propiedades y sus casos. Considerando tal ontología la semántica forma de OWL especifica como sacar las consecuencias lógicas, por ejemplo hechos no literalmente presentes en la ontología.

Linux/Debian

El sistema operativo elegido para la realización del proyecto ha sido una distribución basada en el núcleo linux.

Linux es la denominación de un sistema operativo y el nombre de un núcleo. Es uno de los paradigmas del desarrollo de software libre (y de código abierto), donde el código fuente está disponible públicamente y cualquier persona, con los conocimientos informáticos adecuados, puede libremente estudiarlo, usarlo, modificarlo y redistribuirlo.

El término Linux estrictamente se refiere al núcleo Linux, pero es más comúnmente utilizado para describir al sistema operativo tipo Unix (que implementa el estándar POSIX), que utiliza primordialmente filosofía y metodologías libres (también conocido como GNU/Linux) y que está formado mediante la combinación del núcleo Linux con las bibliotecas y herramientas del proyecto GNU y de muchos otros proyectos/grupos de software (libre o no libre). El núcleo no es parte oficial del proyecto GNU (el cual posee su propio núcleo en desarrollo, llamado Hurd), pero es distribuido bajo los términos de la licencia GPL (GNU General Public License).

La elección de este sistema operativo no ha sido casual. A parte de la política de código abierto bastante atractivo para nosotros, está el hecho de ser gratuito. Además los equipos de la facultad de informática contaban con la distribución Debian lo que nos facilitó el trabajo.

Al ser nuestro proyecto gratuito y de libre distribución parecía lógico que corriera sobre un sistema de iguales características.

No debemos olvidar que, aunque está programado sobre un sistema operativo en concreto, al estar realizado en Java es multiplataforma y, por lo tanto, con pequeñas variaciones puede ser utilizado en cualquier otro equipo con cualquier otro sistema operativo.

La distribución de linux escogida, como se comentó anteriormente, es Debian o, más exactamente, Debian GNU/Linux que es una distribución Linux, que basa sus principios y fin en el software libre.

Creada por el proyecto Debian en el año 1993, la organización responsable de la creación y mantenimiento de la misma distribución, centrado en GNU/Linux y utilidades GNU. Éste también mantiene y desarrolla sistemas GNU basados en otros núcleos.

Nace como una apuesta por separar en sus versiones el software libre del software no libre. El modelo de desarrollo es independiente a empresas, creado por los propios usuarios, sin depender de ninguna manera de necesidades comerciales. Debian no vende directamente su software, lo pone a disposición de cualquiera en Internet, aunque sí permite a personas o empresas distribuir comercialmente este software mientras se respete su licencia.

Como se puede entrever la política Debian es también muy atractiva para jóvenes desarrolladores.

GNOME

GNOME o Gnome es un entorno de escritorio para sistemas operativos de tipo Unix bajo tecnología X Window, se encuentra disponible actualmente en más de 35 idiomas. Forma parte oficial del proyecto GNU.

Como con la mayoría de los programas GNU, GNOME ha sido diseñado para ejecutarse en toda la gama de sistemas operativos de tipo Unix con X Window, y especialmente pensado para GNU/Linux. Desde sus inicios se ha utilizado la biblioteca de controles gráficos GTK, originalmente desarrollada para el programa The GIMP.

Se sopesó la idea de trabajar con KDE o GNOME pero una vez estudiadas ambas opciones se escogió la de Gnomo ya que vimos más sencillo el sobrescribir claves de los valores de los parámetros del escritorio y su configuración. Además este entorno viene por defecto con Debian.

4.2 Arquitectura Software:

4.2.1Esquema

Debido al elevado número de clases que componen la aplicación en el esquema de las relaciones entre las clases solo mostramos las más importantes.

Cada clase se relaciona con las otras a través de su correspondiente interfaz.

La clase GuiPrincipal se encarga de crear todas las clases gráficas de la aplicación, como el sistema se compone de pestañas cada pestaña es una funcionalidad. PanelCambiosLinux y PanelCambiosInternos van a recoger los cambios que quiera efectuar el usuario al interfaz de Linux o al del propio programa respectivamente.

GuiUsuarioSistema es la entrada de los datos de la persona que trabaja con PERSIA.

PanelCuestionarios se encarga de crear los dos paneles que se corresponden con los dos test de personalidad de Keirsey.

A la vez que se crea la GuiPrincipal se crea el AccesoModeloDatos que es la clase que rige como se va a trabajar con la información contenida en las ontologías y a ejecutar los cambios en el sistema.

Ciclo CBR ejecuta un ciclo completo en el sistema, recuperando las variaciones que se deducen del cuestionario corto de Keirsey.

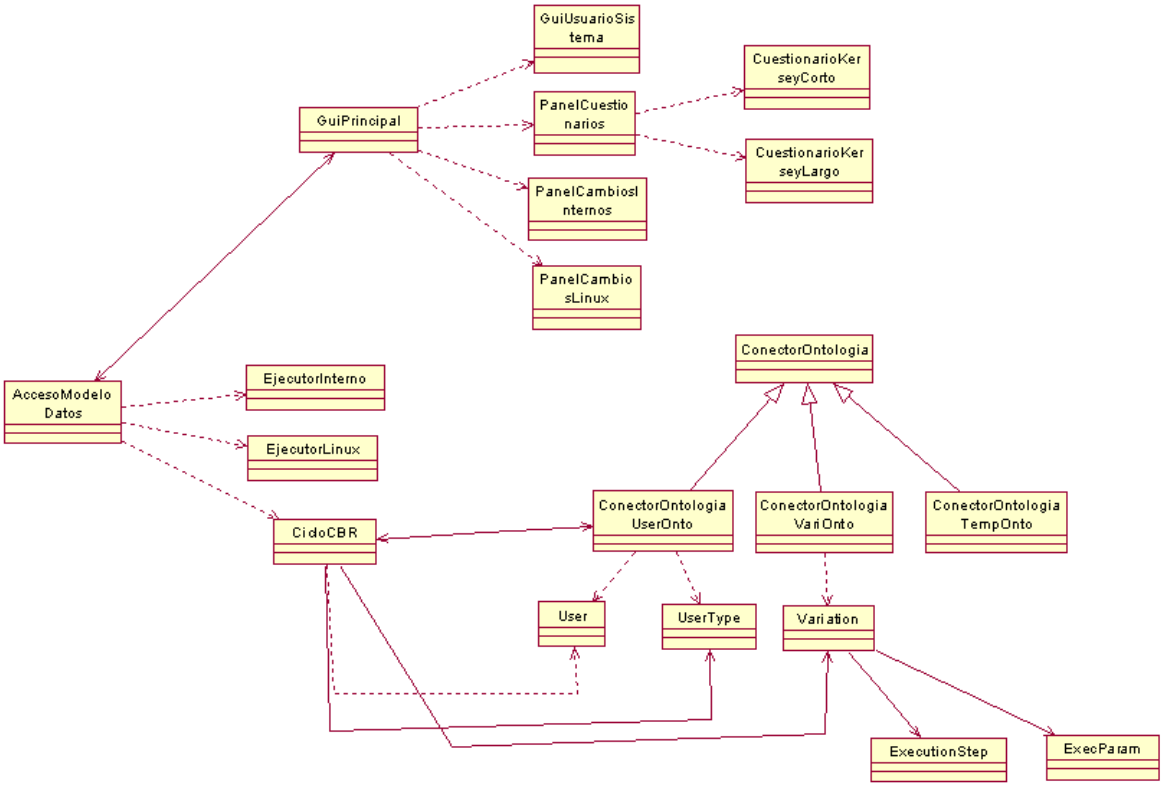
EjecutorLinux ejecuta esas variaciones.

EjecutorInterno ejecuta las variaciones que afectan al aspecto del programa.

Todos los conectores extienden a la clase ConectorOntología ampliando sus funcionalidades para que se adapten a las clases java que van a modelar. Por ejemplo conectorOntologiaUserOnto se encarga de recuperar, guardar y modificar la información que contenga la ontología UserOnto. Lo mismo sucede con ConectorOntologiaVariOnto y ConectorOntologiaTempOnto.

Cuando se leen las ontología para trabajar con ellas es necesario volcar la información que contienen a clases Java para recoger esa información se han creado las clases User, UserType, Variation, ExecutionStep, ExecParam.

Por lo tanto teniendo en cuenta todo esto el diagrama de clases es el siguiente.



4.2.2 Componentes y clases principales

El código de la aplicación está incluido en varios paquetes Java para una mayor legibilidad. Todas las clases están, por lo tanto, agrupadas según su funcionalidad. De esta manera se consigue en aplicaciones como ésta, con un gran número de clases, sea más sencillo la localización de las mismas y su comprensión global dentro de un todo.

A continuación se enumeran todos los paquetes de los que consta la aplicación y sus clases:

com.clasesOntologias.conectores

AuxiliarSinOrden.java
AuxiliarThemes.java
AuxiliarThemesMusica.java
Auxliar.java
AuxliarConMusica.java
ConectorOntologia.java
ConectorOntologiaTempOnto.java
ConectorOntologiaUserOnto.java
ConectorOntologiaVariOnto.java
IConectorOntologia.java
IConectorOntologiaTempOnto.java
IConectorOntologiaUserOnto.java
IConectorOntologiaVariOnto.java

com.clasesOntologias.tempOnto
- ITempCharacter.java
- ITempCommunication.java
- ITempImplementation.java
- ITempLetter.java
- ITempTrait.java
- ITempType.java
- TempCharacter.java
- TempCommunication.java
- TempImplementation.java
- TempLetter.java
- TempTrait.java
- TempType.java

com.clasesOntologias.userOnto
- IUser.java
- IUserType.java
- User.java
- UserType.java

com.clasesOntologias.variOnto
- ExecParam.java
- ExecutionStep.java
- IExecParam.java
- IExecutionStep.java
- IVariation.java
- Variation.java

com.utilidades.constantes
- INombreFicherosConfiguracion.java
- com.utilidades.datos
- AlmacenPropiedades.java
- IMapaPropiedades.java
- IPropiedadesFichero.java
- MapaPropiedades.java
- PropiedadesFichero.java

com.utilidades.funcionalidades
- AccesoModeloDatos.java
- CicloCBR.java
- EjecutorInterno.java
- EjecutorLinux.java
- EstilosLetra.java
- IAccesoModeloDatos.java
- ICicloCBR.java
- IEjecutorInterno.java
- IEjecutorLinux.java
- IEstilosLetra.java
- IPropiedadesFichero.java

PropiedadesFichero.java

com.utilidades.grafico

CuestionarioKerseyCorto.java
CuestionarioKerseyLargo.java
EstilosLetra.java
GuiAlegriaUsuario.java
GuiPrincipal.java
GuiUsuarioSistema.java
IGuiPrincipal.java
InterfazGrafico.java
InterfazGrafico2.java
IPanelCambiosInternos.java
IPanelCambiosLinux.java
IPanelCuestionarios.java
PanelCambiosInternos.java
PanelCambiosLinux.java
PanelCuestionarios.java

com.utilidades.musica

ReproductorMP3.java

A continuaciones explican los distintos paquetes dentro de la aplicación para su mejor comprensión y para facilitara la legibilidad de la aplicación.

com.clasesOntologias.conectores

Las clases referidas a todos los conectores de las distintas ontologías. Aquí se incluye el conector más genérico y los particulares, así como las interfaces. El uso de interfaces como buena práctica de programación se ha seguido durante la realización de todo el código del proyecto.

Además también se incluyen las clases auxiliares para precargar las ontologías para la realización de las pruebas.

com.clasesOntologias.tempOnto

Clases referidas a la ontología temp, donde se guardan las distintas partes de ésa ontología para poder trabajar con ellas. Todos los elementos de la ontología tienen su correspondiente clase en éste paquete.

com.clasesOntologias.userOnto

Clases referidas a la ontología user, donde se guardan las distintas partes de ésa ontología para poder trabajar con ellas. Todos los elementos de la ontología tienen su correspondiente clase en éste paquete.

com.clasesOntologias.variOnto

Clases referidas a la ontología vari, donde se guardan las distintas partes de ésa ontología para poder trabajar con ellas. Todos los elementos de la ontología tienen su correspondiente clase en éste paquete.

com.utilidades.datos

Aquí se incluyen las clases relacionadas con los mapas y almacenes de propiedades.

com.utilidades.funcionalidades

En este paquete se incluyen las clases que agrupan la funcionalidad de la aplicación. El ciclo cbr y los cambios disponibles en el programa.

com.utilidades.grafico

Como su propio nombre indica aquí se incluyen las clases de la interfaz gráfica.

com.utilidades.musica

Este paquete incluye la clase reproductor para la reproducción de la canción.

Para terminar se explican superficialmente algunas de las clases más representativas de la aplicación.

Auxliar.java

Hay más clases como ésta que contienen la palabra auxiliar. Todas éstas sirven para realizar cargas de datos en las ontologías de forma automática al ejecutarse. De esta manera la realización de las pruebas de los conectores eran más sencillas al realizarse guardados y cargados masivos de datos. Hay que tener en cuenta que cada clase de éstas representa un momento en el estado de la realización del proyecto, existiendo cargas de datos sin orden, con orden, con los temas de escritorio ya añadidos y con las músicas.

Pueden ser usadas en futuras modificaciones del proyecto para realizar comprobaciones en los conectores.

ConectorOntologia.java

Esta clase es el conector más genérico y del que cuelgan los conectores más específicos de cada ontología. Reúne las funcionalidades más básicas y generales que no dependen de la estructura y datos que contienen las distintas ontologías. Como todas las clases tiene definida su interfaz.

ConectorOntologiaUserOnto.java

Esta clase es el conector específico para la ontología de user. Realiza todas las funciones básicas como guardar, cargar, borrar, modificar...

ConectorOntologiaVariOnto.java

Esta clase es el conector específico para la ontología de variaciones. Realiza todas las funciones básicas como guardar, cargar, borrar, modificar...

User.java

Se explica ésta clase como ejemplo de otras muchas que representan una parte una ontología. Estas clases engloban todas las funcionalidades que se necesitan para el manejo de estos elementos de las distintas ontologías.

Disponen de distintos constructores, los accedentes y mutadores de todos sus atributos, los clone, los toString, los compare... para poder servir de forma eficiente a las demás clases que las utilizan. Como ya se ha indicado anteriormente disponen todas de sus interfaces como práctica aconsejable de programación que hemos seguido.

Sus atributos representan todas las partes de la ontología:

Nombre: Nombre del usuario en el sistema. Este nombre es único y sirve para cargar tus variaciones cuando se arranca el programa.

hasQuestionResult: Respuestas que ha proporcionado el usuario al contestar el cuestionario corto Keirsey.

hasArtisanProp: Porcentaje de artesano que tiene la personalidad del usuario.

hasGuardianProp: Porcentaje de guardián que tiene la personalidad del usuario.

hasIdealistProp: Porcentaje de idealista que tiene la personalidad del usuario.

hasRationalProp: Porcentaje de racional que tiene la personalidad del usuario.

hasPersonalData: Puede guarda cualquier dato de interés sobre el usuario.

hasUserAdaptationsFromReasoner: Adaptaciones (variaciones de GNOME) que ha deducido el sistema basándose en el test de personalidad de Keirsey rellenado por el usuario.

hasUserAdaptationsFromFeedback: Adaptaciones que el usuario ha modificado sobre las que le había propuesto el sistema. Cuando se rellena el test de Keirsey PERSIA deduce una serie de modificaciones del escritorio GNOME y las ejecuta. Una vez hecho esto el usuario puede cambiar estas variaciones, pues estas modificaciones son las que se guardan en este campo.

hasUserAdaptations: Contiene las variaciones de los campos

hasUserAdaptationsFromReasoner

hasUserAdaptationsFromFeddBack.

hasUserType: A cada usuario le corresponde un userType, el nombre de ese userType se guarda en este campo.

hasTemperamentProportions: Guarda las proporciones de la personalidad del usuario.

PropiedadesFichero.java

Se encarga de guardar un fichero de configuración, modificarlo, salvarlo de nuevo al fichero... Ésta es la clase genérica que se utilizará para el manejo de los distintos ficheros de configuración de la aplicación.

CicloCBR.java

Esta es la clase principal de la realización del ciclo CBR como su propio nombre indica. Utiliza la mayoría de las clases nombradas anteriormente y manda lanzar las distintas variaciones.

Cuando se realizó esta clase y se probó se tuvo que volver a realizar un gran esfuerzo de codificación en los conectores.

EjecutorInterno.java

Esta es la clase que ejecuta las distintas variaciones internas del programa, los cambios de color de fondo, letra, etcétera...

EjecutorLinux.java

De la misma manera que la anterior esta clase se encarga de la realización de los cambios en linux.

CuestionarioKerseyCorto.java

Clase que se encarga de comprobar si está bien relleno el cuestionario corto y de calcular el temperamento del usuario.

CuestionarioKerseyLargo.java

Clase que se encarga de comprobar si está bien relleno el cuestionario largo y de calcular el temperamento del usuario.

PanelCambiosInternos.java

Clase que se encarga del aspecto de la interfaz gráfica de los cambios internos. Es una clase que extiende a un Jpanel, por lo tanto es una clase de interfaz gráfica.

PanelCambiosLinux.java

Clase que se encarga del aspecto de la interfaz gráfica de los cambios en linux. Es una clase que extiende a un Jpanel, por lo tanto es una clase de interfaz gráfica.

PanelCuestionarios.java

Clase que se encarga del aspecto de la interfaz gráfica de los distintos cuestionarios. Es una clase que extiende a un Jpanel, por lo tanto es una clase de interfaz gráfica.

GuiPrincipal.java

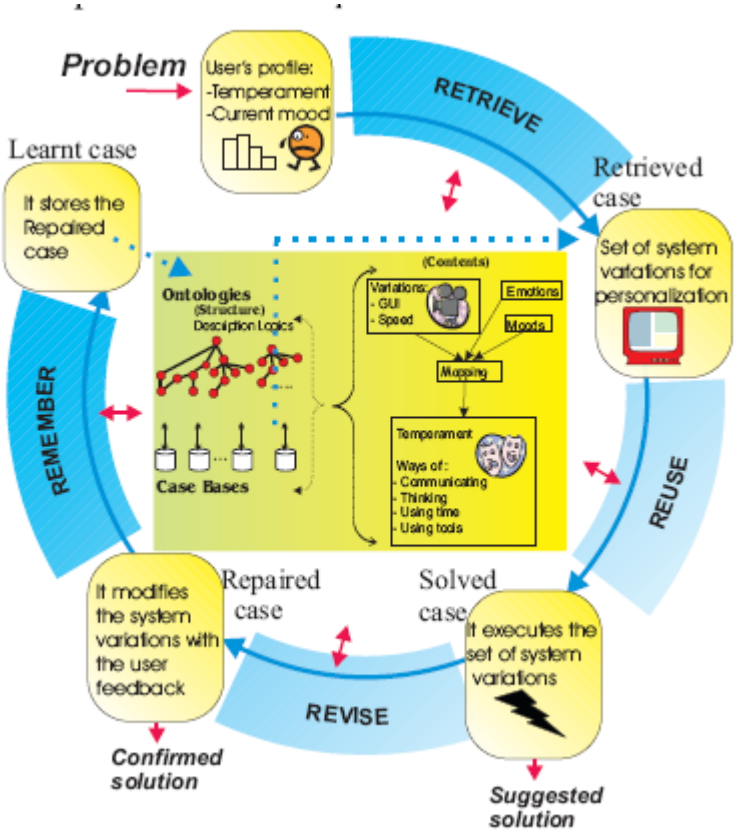
Clase que agrupa las tres anteriores y que presenta el aspecto final de la aplicación.

4.2.3 Ciclo global

PERSIA funciona internamente con un CBR. De esta manera esta aplicación se irá comportando mejor a medida que se vaya usando y su base de conocimiento se haga mayor. Es normal por ello que en unas primeras ejecuciones los resultados obtenidos no sean los esperados. Una de las razones fundamentales es que el programa origen parte con cuatro casos, cado uno representado por un UserType.

A medida que se vaya usando la aplicación estos casos irán creciendo y se irán creando casos más específicos para diferentes porcentajes de los temperamentos.

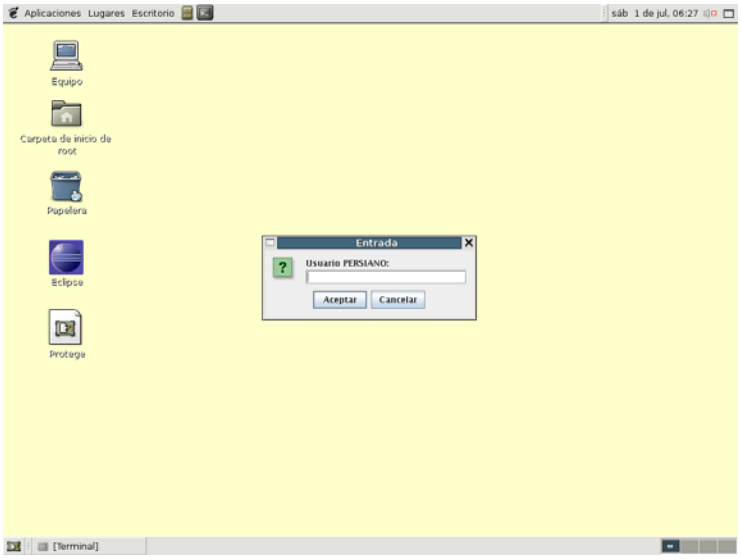
En este apartado, se explicará el proceso detallado del CBR y por que etapas va pasando. De esta manera se espera poder dar un mayor conocimiento de la aplicación sin adentrarnos demasiado en detalles técnicos.



Paso 1, Problema, recoger datos del usuario.

El primer paso que debe realizar el usuario es rellenar un test de personalidad.

Esto nos dará el temperamento del usuario dividido en cuatro porcentajes. Estos porcentajes serán usados más tarde por el CBR para encontrar los casos con mayor similitud.



Contamos con dos test, uno corto y uno largo. Cada uno nos da unas medidas.

Actualmente sólo está implementado el corto. Actualmente el test largo solo está implementada la parte gráfica pudiendo rellenarlo pero no entrando en el CBR una vez está completado. Debido al tamaño de este test una opción para que sea más accesibles a los usuarios es permitir que lo realicen por etapas y que se almacenen los resultados temporales en las ontologías.

Paso 2, Consultas la ontología de usuarios para recuperar el caso mas similar.

Una vez tenemos los porcentajes del temperamento del usuario debemos de compararlo con todos los casos disponibles actualmente, de esta manera nos quedaremos con el que mayor similitud tenga.

Para calcular esto se hace la resta de los porcentajes del usuario actual con los que nos encontramos en la ontología siendo el más apto el que tenga menor diferencia.

Paso 3, Adaptación de las variaciones.

Una vez hayamos recuperado el UserType (o caso) más “similar” debemos de adaptarlo ya que el mejor caso encontrado no tiene porque ser realmente bueno debido a: que el número de casos que hay en ese momentos almacenados sea bajo por el poco uso de la aplicación, se ha dado con un individuo muy diferente a los que había hasta ahora en la ontología, etc. Estos u otros casos pueden hacer que la similitud encontrada sea demasiado baja y estemos trabajando con dos casos completamente distintos.

De esta manera todas las variaciones tienen un rango de posibles valores, pudiendo modificarse proporcionalmente dependiendo de la diferencia entre el porcentaje de nuestro nuevo usuario y el del caso recuperado. Con esta técnico adaptaremos todas las adaptaciones al nuevo usuario.

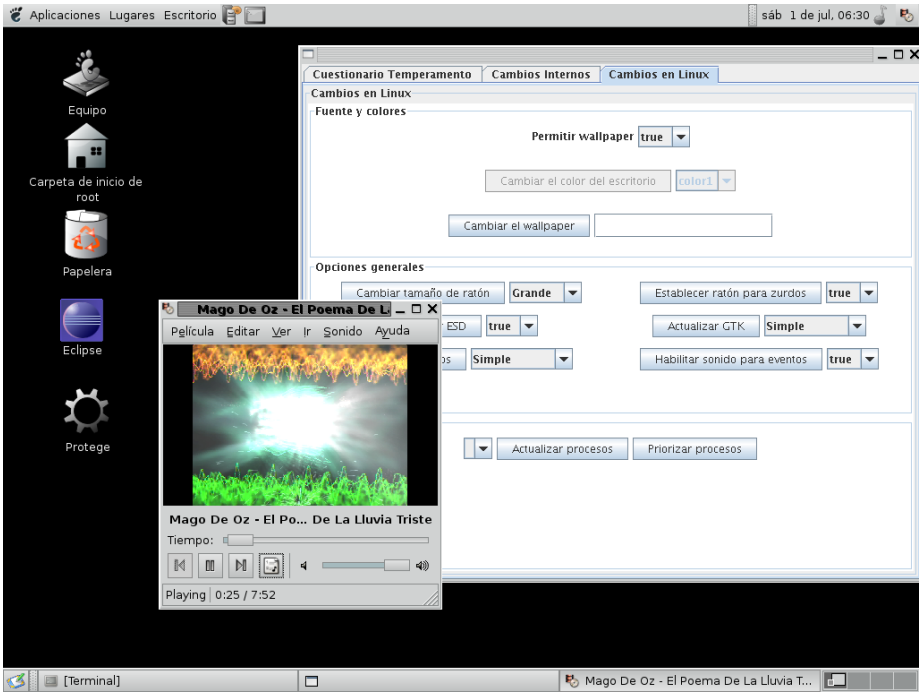
Paso 4, Ejecución de las variaciones.

Una vez se hayan adaptado las diferentes variaciones al nuevo usuario se ejecutarán en el sistema.

Paso 5, Modificaciones por el usuario.

Cuando las variaciones hayan sido ejecutadas, el usuario puede cambiar las variaciones que ha hecho automáticamente el sistema. Para hacer esto PERSIA dispone de una funcionalidad.

Todos los cambios que haga manualmente el usuario se plasmarán en las ontologías para las futuras ejecuciones. Esto permite corregir el comportamiento del sistema para futuros usuarios, haciendo que converjan las características que le gustan a los usuarios de determinados temperamentos.



Paso 6, Almacenamiento del nuevo caso.

Este paso no se da siempre. La aplicación tiene un parámetro de similitud. Este parámetro es el umbral que damos para considerar que el caso actual es lo suficientemente diferente como para considerarlo como un caso nuevo. De esta manera si fijamos este parámetro muy bajo, usuarios muy parecidos puedes dar origen a un caso nuevo, esta es una buena técnica al principio de la aplicación para crear casos. Si damos a este parámetro un valor lo suficientemente alto será difícil que se creen nuevos casos.

De esta manera al crear un nuevo usuario si este es lo suficientemente diferente al que mas se asemeja de la ontología se creará uno nuevo en la aplicación.

4.3 Estado de la implementación

La implementación de nuestra aplicación está centrada en lo que podríamos denominar el núcleo, de hecho, dos de los tres componentes del grupo de trabajo se han dedicado de forma especial a eso.

La idea era conseguir unas clases lo más robustas y fiables posibles para que los continuadores de la aplicación sólo se tengan que preocupar de añadir mejoras referidas a los cambios que se producirán en el sistema cuando se ejecute el programa.

4.3.1 Funciones implementadas en el del Prototipo Persia 0.1

El primer prototipo realizado, que a partir de ahora denominaremos Persia 0.1, fue la primera versión más o menos estable que realizaba ya por completo el acceso a las ontologías de forma fiable, aunque a la larga se encontraron algunos fallos que fueron siendo subsanados, el ciclo cbr, y la variación del cambio de color del fondo de la pantalla. De esta manera la idea era que a partir de este prototipo ya sólo nos tendríamos que centrar en añadir cambios para hacer más vistosa la aplicación y que variara lo más posible el escritorio y las demás propiedades del sistema.

Así pues se realizaron las clases que forman el esqueleto de la aplicación:

Conectores de las distintas ontologías. Los conectores que son la base del proyecto, ya que era la parte de código más novedosa para nosotros y que se acabó llevando gran parte del esfuerzo global de dos de los miembros del equipo. Los conectores fueron realizados de forma que respetaran distintos patrones de diseño. Todos utilizaban una clase más genérica ConectorOntologias que era más genérica y que servirá a futuros desarrolladores si se incluyen nuevas ontologías.

Las clases que se utilizan para almacenar la información de cada uno de los elementos de las ontologías también fueron realizados para este prototipo. Estas clases sirven de apoyo a los conectores anteriormente citados.

También se realizaron ya los distintos cuestionarios y los cambios que aparecen en la interfaz gráfica. Todos implementados, así como la comprobación de si está bien o mal rellenado el cuestionario largo y el corto.

También se realizaron las clases correspondientes al ciclo cbr, amén de clases auxiliares que nos serían muy útiles para cargar y borrar las ontologías para las distintas pruebas que necesitaríamos realizar.

De esta forma el prototipo PERSIA 0.1 contaba ya con una interfaz gráfica que mostraba los dos cuestionarios, los posibles cambios internos del programa y cambios en linux. Comprobaba ya si los cuestionarios estaban bien rellenos y al rellenarlos ejecutaba el ciclo cbr lanzando el cambio de color del fondo del escritorio.

4.3.2 Funciones implementadas en el del Prototipo Persia 0.2

Para el prototipo Persia 2.0 se tuvieron que realizar cambios a nivel de los conectores. Tras realizar múltiples pruebas con la versión anterior se tuvieron que añadir nuevas funciones como la de sobrescribir un individuo que nos dio múltiples problemas o las opciones para borrar.

En los conectores a nivel de pruebas y de errores que se iban encontrando según se añadían nuevas funcionalidades se continuó trabajando, aunque cambiando líneas de código en lugar de añadir.

Otra mejora que se añadió a la aplicación a nivel de código fueron comentar mejor las clases que más se estaban tocando.

A nivel de lo que ve el usuario se añadieron varias funcionalidades que a continuación se enumeran:

Se añadieron cambios de escritorio más vistosos, como el cambio de temas. Los temas se escogieron de los disponibles en la distribución linux que teníamos instalada en el laboratorio para evitar incompatibilidades. Para añadir los nuevos temas se tuvieron que modificar las clases de los conectores y las ontologías para recuperar los parámetros. A medida que se añadían prestaciones y se probaban se seguían depurando los conectores.

Los cambios de temas se dividen en dos, los de iconos y los de la vista de las ventanas del sistema. También se añadió que el usuario pudiera cambiar directamente los valores. El sistema guarda estos cambios y los memoriza para la siguiente ejecución del mismo usuario.

Otra funcionalidad que se añadió fue que el sistema también arrancara una canción al ejecutarse. Estas canciones están en una carpeta y se pueden cambiar por otras.

Este prototipo fue probado durante varios días y depurado y empaquetado para su distribución. Una vez empaquetado se probó su ejecución.

4.3.3 Posibles mejoras futuras y limitaciones actuales

Las mejoras futuras básicamente se pueden englobar en la introducción de nuevos cambios que hagan más vistosa la ejecución de la aplicación. Las variaciones nuevas que se pueden introducir dependerán siempre del entorno en el que se trabaje. La idea es que al final pueda modificar todo los parámetros que nos permite ajustar el sistema mediante los ficheros de configuración del sistema. Hay que tener en cuenta que se podrán programar las variaciones del sistema gráfico (X-Window) y las del sistema operativo y las de la distribución, con lo que las posibilidades son muy grandes una vez resuelto el tema de los conectores.

Una limitación que tiene nuestro sistema es que, a pesar de tener la posibilidad de rellenar el cuestionario largo, que te indica si alguna pregunta está mal respondida o sin responder, y que te muestra por pantalla el resultado del test, no trabaja con esa información en el sentido de que no sabemos qué tipo de variación corresponde a cada temperamento. Esto lo debería decidir algún psicólogo. Para ampliar la aplicación se debería hacer que trabajara también con ese test y dar la posibilidad de dejarlo a medio rellenar. Hay que tener en cuenta que los tipos de temperamento con los que trabajamos en el cuestionario corto, y por ende en la aplicación, no son los mismos que los que resultan del cuestionario largo. Éste último tiene subtipos de temperamento que harán que se necesite variar las ontologías añadiendo campos.

Una de las cosas que estaban y que al final se quitó por no saber cómo debía afectar al cambio escogido era el estado actual del usuario. Ésta es una posible futura mejora. Dar la posibilidad al usuario de indicar cómo se encuentra de estado de ánimo (triste, alegre, enfadado) y que eso afecte también a la hora de escoger los valores que deben tener los parámetros de la instrucción a ejecutar para cambiar el aspecto del equipo.

Otro punto que queda sin realizar es la internacionalización. Meter más idiomas. Esto ya está preparado de antemano y sólo hay que añadir ficheros de idiomas a la aplicación. No hay que toca el código sino sólo añadir ficheros de configuración en texto plano.

5. Experimentos y Resultados:

Los experimentos realizados con distintas personas para obtener sus puntos de vista, críticas, sugerencias y demás opiniones respecto al proyecto que aquí se presenta fueron realizados delante de alguno de los programadores a la conclusión del prototipo Persia 0.2.

Los resultados obtenidos y que se muestran a continuación pueden servir a los futuros continuadores del proyecto como guía para mejorar, añadir o modificar cosas de la aplicación. También para que nosotros mismos nos diéramos cuenta de cómo lo juzgan unos ojos ajenos a la informática y que no han visto crecer día a día el proyecto, con sus dificultades y sus problemas.

Hay que tener en cuenta que nuestro prototipo, aunque calcula los porcentajes y tipos de temperamento correctamente de los usuarios, no tiene enlazado ese tipo de temperamento al color exacto. Igual que se dice del color se dice de las otras variaciones. Ése es un ámbito de estudio que desde el principio se nos indicó que quedaba fuera de nuestro trabajo. Para los futuros desarrolladores será interesante trabajar con algún psiquiatra que les ayude a saber qué variación está más indicada a cada temperamento.

5.1 Participantes

Los participantes de las encuestas se han escogido de forma que representen al mayor rango posible del espectro social, atendiendo a la edad, nivel cultural y nivel de manejo y comprensión de los ordenadores y otros aspectos de la informática.

Naturalmente son geste de nuestro entorno más próximo

5.2 Pruebas y encuestas

Las pruebas consistían sentar al participante delante del ordenador y arrancar la aplicación. No se les hacía rellenar ningún papel sino que simplemente se le iba preguntando su opinión en distintos momentos de uso de la aplicación. Se tomaban notas tanto de preguntas con cuatro o cinco posibles respuestas para configurar las gráficas como de sugerencias. A continuación se describe someramente el procedimiento utilizado.

Primero se le pedía que mirara el diseño y las distintas opciones que se presentaban y formulara valoraciones previas al respecto. Si echaba alguna en falta o sobraba alguna. Si le gustaba la interfaz gráfica, si era sencilla e intuitiva, etcétera. También probaba las pestañas de cambios internos del programa y de cambios en linux.

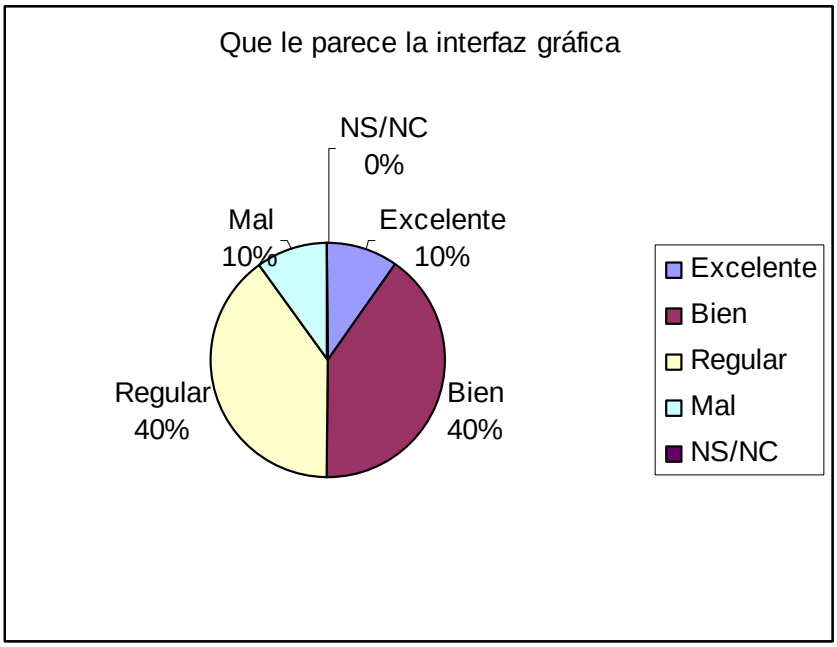
De forma seguida se le hacía rellenar el cuestionario corto, aunque también se le dejaba ver el largo, y se procedía a aceptar las respuestas y ver cómo el programa se ejecuta y los distintos cambios que produce en el sistema.

Para finalizar se le indicaba si quería añadir algo, algún comentario o alguna indicación y se le agradecía su paciencia y dedicación.

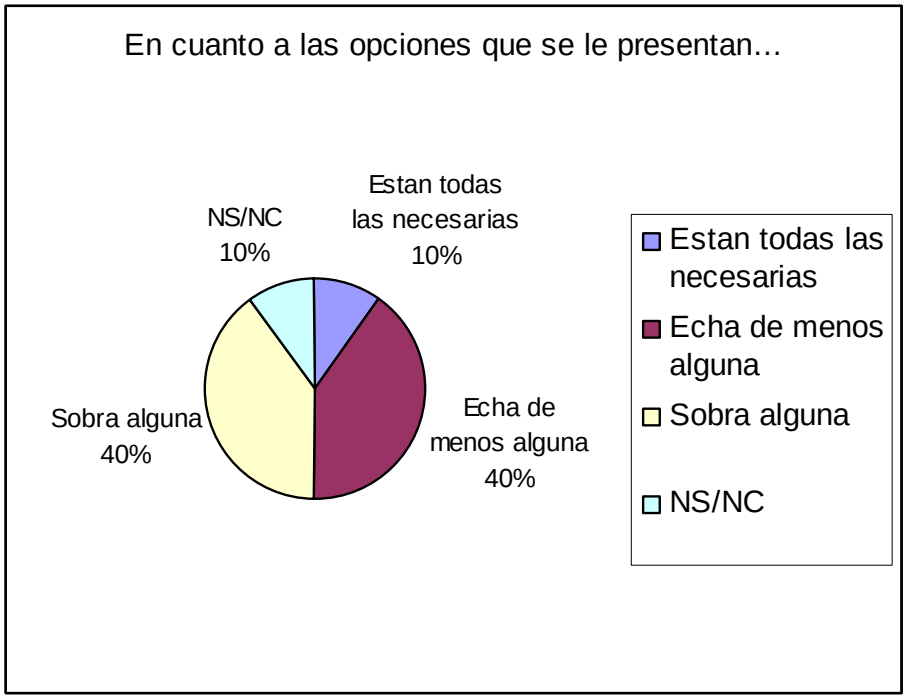
5.3 Resultados

A continuación se muestran los resultados a las distintas preguntas formuladas en porcentajes. Para mejor visualización se utilizan gráficas.

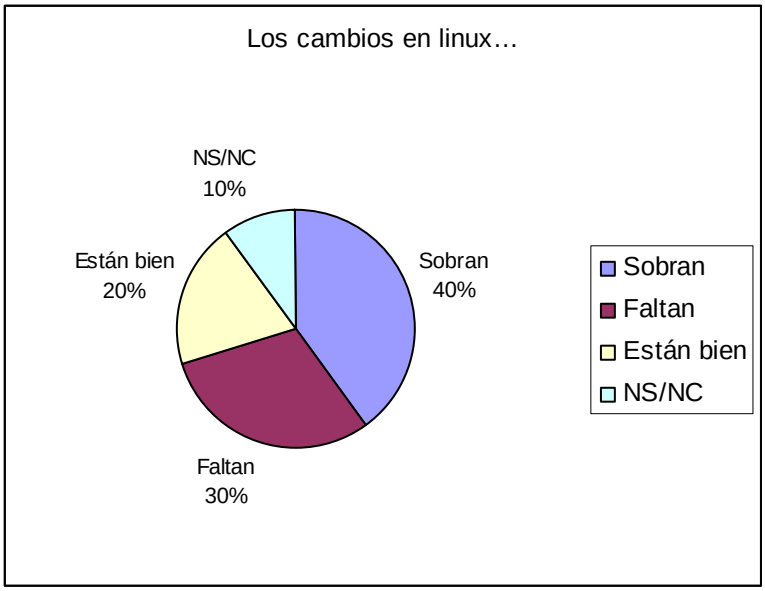
También se manifiestan algunas de las opiniones o sugerencias recogidas en el proceso de las entrevistas.



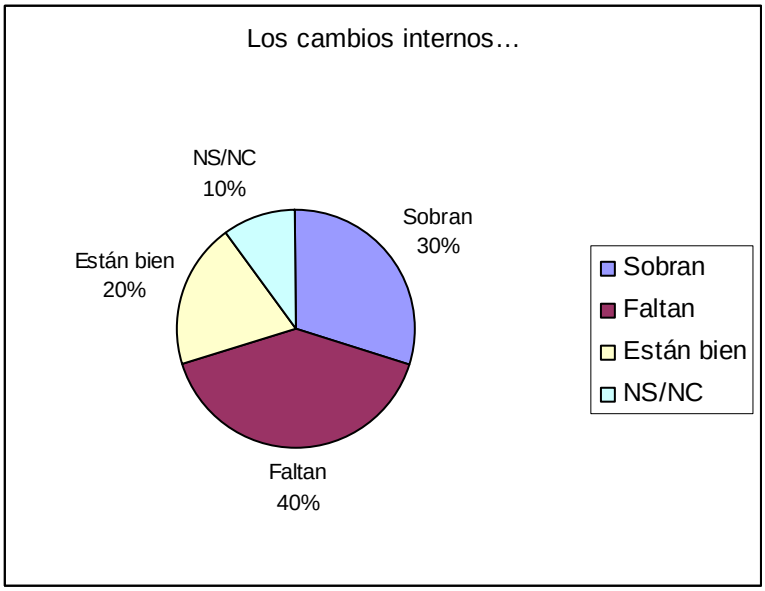
Las opciones, en general, dieron bastante juego, ya que había gente que le sobraban y gente que le faltaban. Así decidimos a continuación preguntar más específicamente sobre las opciones de las distintas pestañas.



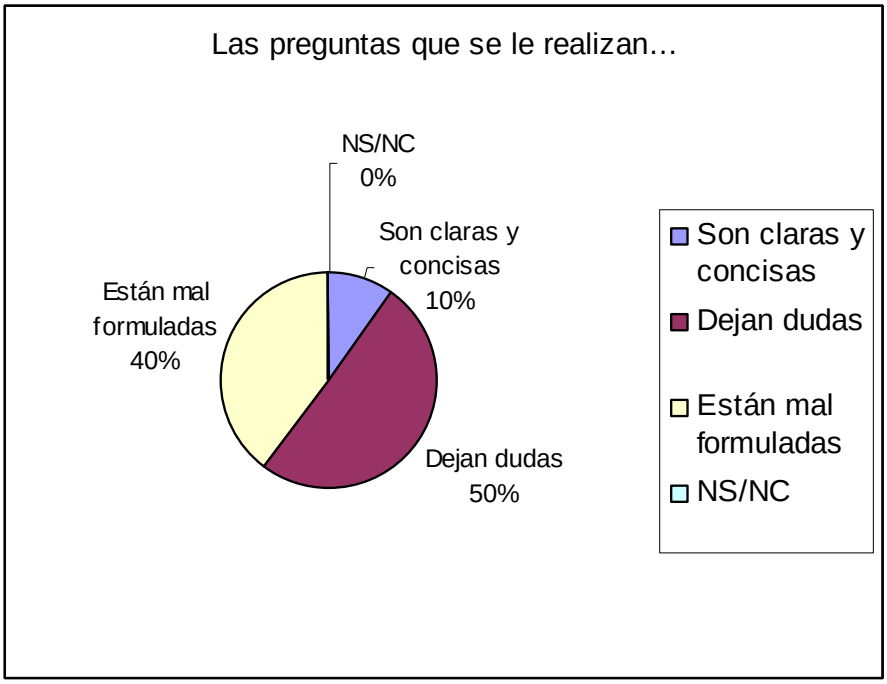
En los cambios de linux una de las cosas que nos decían que sobraban eran los cambios referentes al ratón para zurdos y el tamaño del ratón y que echaban de menos una lista de temas para cambiarlos desde ahí directamente. Si hubiéramos metido el cambio de temas directamente no tendría mucho sentido los cambios de fondo e iconos. De todas maneras la impresión fue favorable.



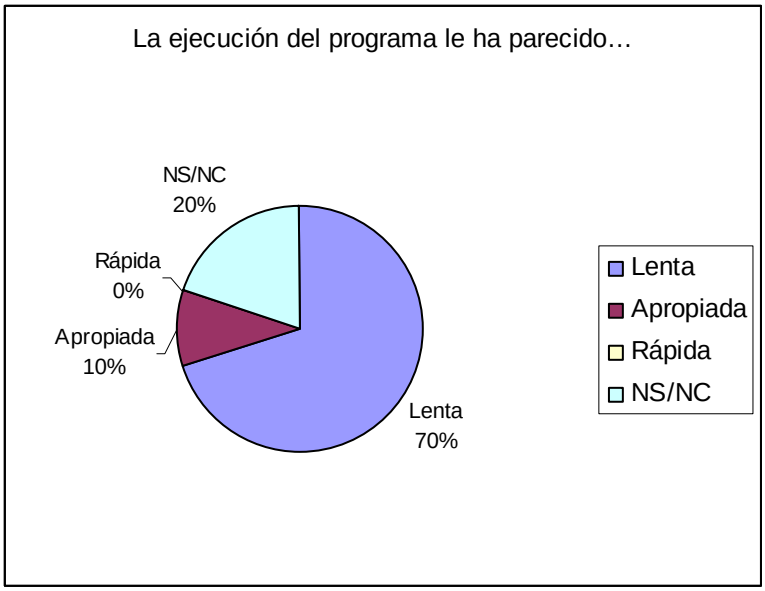
Los cambios internos la mayoría de la gente decía que le resultaban poco útiles, pero que ya puestos deberían reflejar una mayor cantidad de tipos de letras y colores.



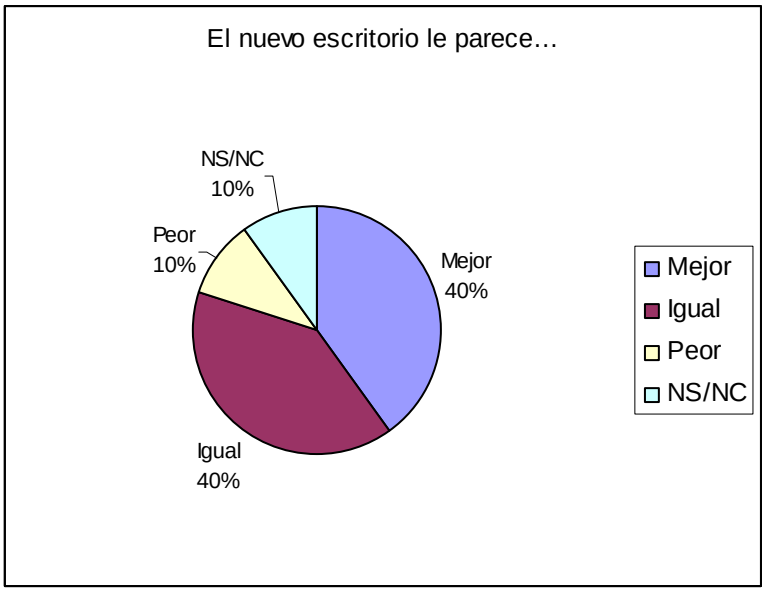
Una de las cosas que más me llamó la atención como entrevistador fue el tiempo que se timaban para responder algunas de las preguntas y cómo resoplaban sin saber muy bien qué responder. Esto nos llevó a incluir ésta pregunta que, aunque no tenga nada que ver con nosotros puesto que las preguntas vienen dadas, pueden servir si alguien se anima a hacerlas más fácilmente respondidas.

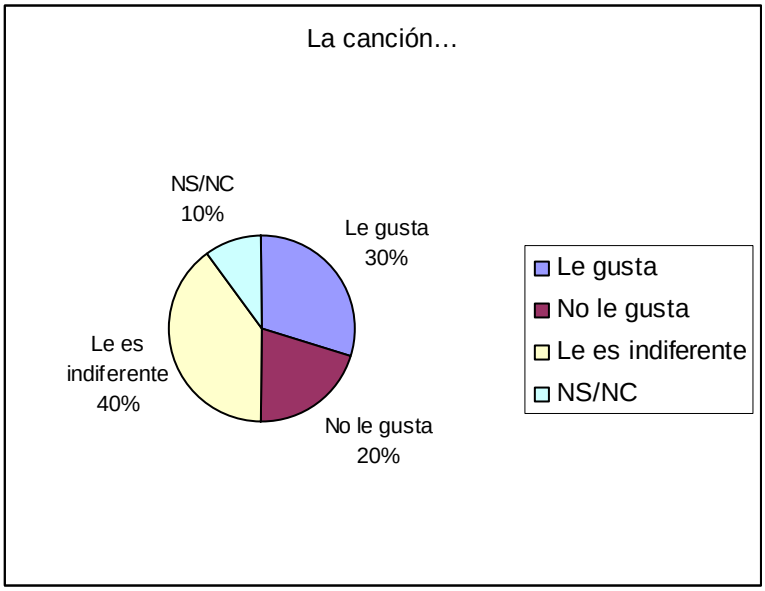


Una vez respondidas las preguntas, aceptadas y viendo cómo se ejecuta el programa se le pregunta por la velocidad, un tema que nos preocupa puesto que nos parece muy lento.

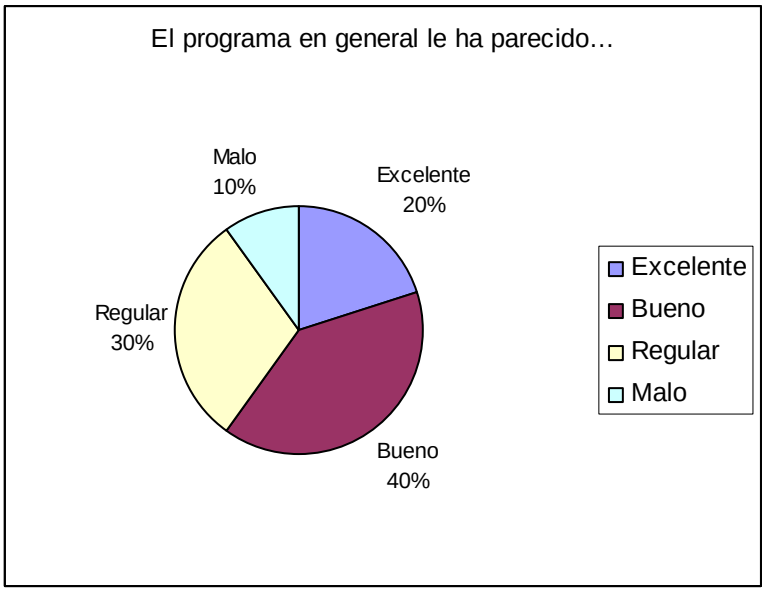


Una vez que el sistema ha cambiado el escritorio y los temas y los iconos y ha ejecutado una canción que suena de fondo se le pregunta por cómo le han parecido esos cambios. Hemos de tener en cuenta que no hemos seguido el orden de ningún psicólogo que nos dijera qué colores o cambios o tipos de música son los más apropiados a cada temperamento. Aún así los resultados obtenidos no son del todo malos.





Ahora llega la última pregunta y la más importante. La valoración final del programa.



Y hasta aquí la encuesta. Hemos de reconocer que la muestra, al estar compuesta de amigos, compañeros y familiares, puede estar algo sesgada, pero siempre se pidió sincerid

6. Documentación

6.1 Manual de usuario

6.1.1 Instalación PERSIA:

Se entrega un archivo comprimido contiene:

- JAR con la aplicación
- Ficheros de configuración
- Ontologías Base de la aplicación
- Ficheros MP3 para posibles variaciones basadas en sonido

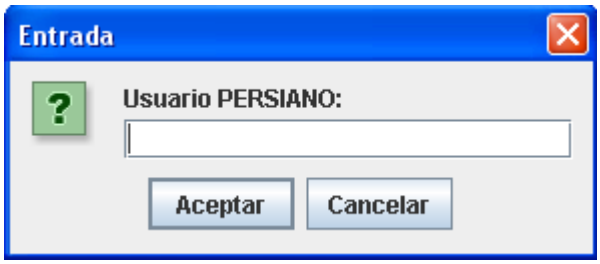
Para su instalación simplemente deberá de descomprimir el fichero en un directorio y ejecutar el JAR.

Nota: para ejecutar un JAR se debe de lanzar desde la consola el comando "java –jar nombreFichero.jar"

6.1.2 Uso de la aplicación

Paso 1:

Al lanzar la aplicación le saldrá una ventana de inicio en el que se pedirá que introduzca el usuario de acceso a la aplicación

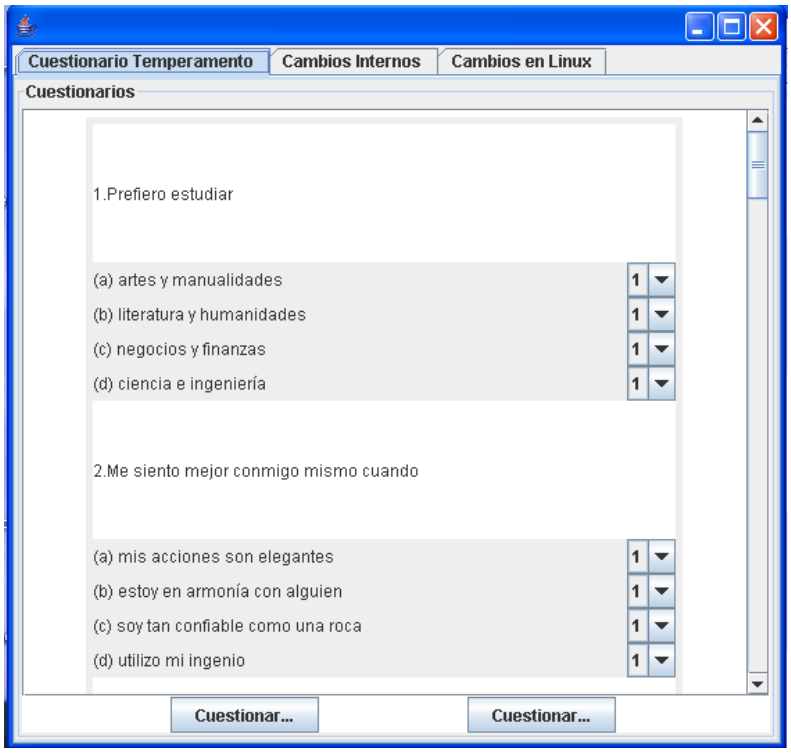


El usuario con el que ha accedido a la sesión de Linux es independiente del usuario de la aplicación. De esta manera a lo largo de una sola sesión de Linux puede haber usado PERSIA con distintos usuarios.

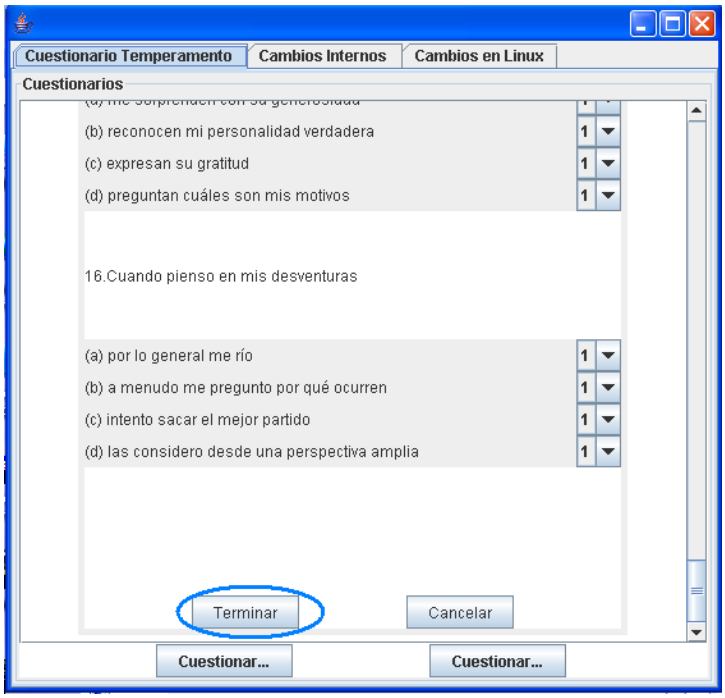
Después de introducir el usuario, debe de dar a "Aceptar".

Paso 2:

Una vez ha introducido el nombre del usuario con el que entrar a la aplicación y haya dado a “Aceptar”, la aplicación le llevará a una ventana con una encuesta.

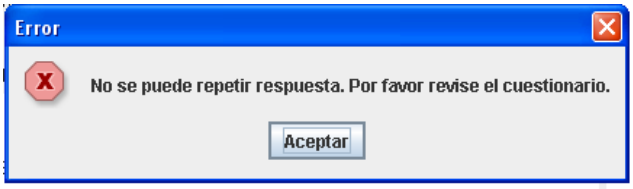


La encuesta contiene dieciséis preguntas con cuatro opciones cada una. Cada opción puede tener el valor de 1-4. Cada uno de los apartados de las preguntas debe de dársele un valor diferente a las otras opciones.

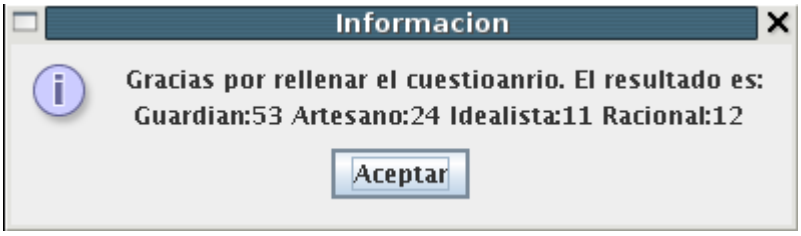


Una vez que se haya rellenado correctamente toda la encuesta, deberá de darle al botón “Terminar”. Esto calculará las características del usuario según el test de personalidad.

Si el test no está correctamente completado cuando el usuario da al botón finalizar se dará un aviso al usuario:



Una vez bien relleno el test y completado, se indicará con una pantalla al usuario y se darán los diferentes porcentajes de su personalidad



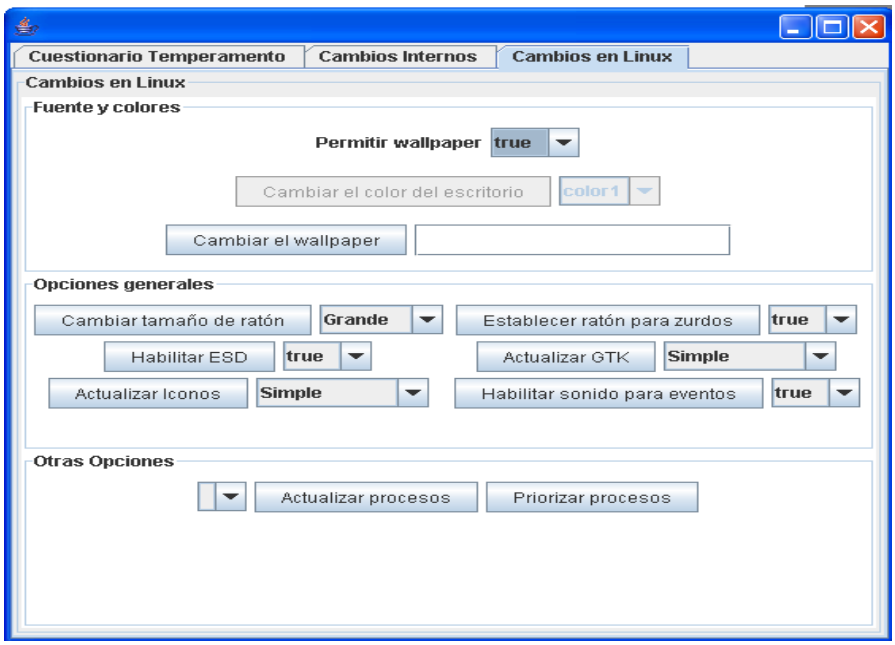
Al dar al botón “Aceptar”, PERSIA realizará los cambios que se asemejen más a los porcentajes calculados en el Test. Estos cambios se verán en el entorno pudiendo cambiar el Theme, color de fondo, lanzando programas musicales, etc.

La aplicación también tiene la posibilidad de realizar un test mayor. Aunque se ofrece la posibilidad este test actualmente no está integrado con el resto de la aplicación.

6.1.3 Cambios externos realizados por el usuario

Si alguno de estos cambios no le gusta al usuario tiene la posibilidad de cambiarlo en el momento y para las futuras ejecuciones de PERSIA con ese usuario.

El primer paso para realizar estos cambios es irse a la sección de la aplicación dedicada a esta tarea. Esta es la pestaña “Cambios en Linux”.



Dentro de esta pestaña podremos modificar:

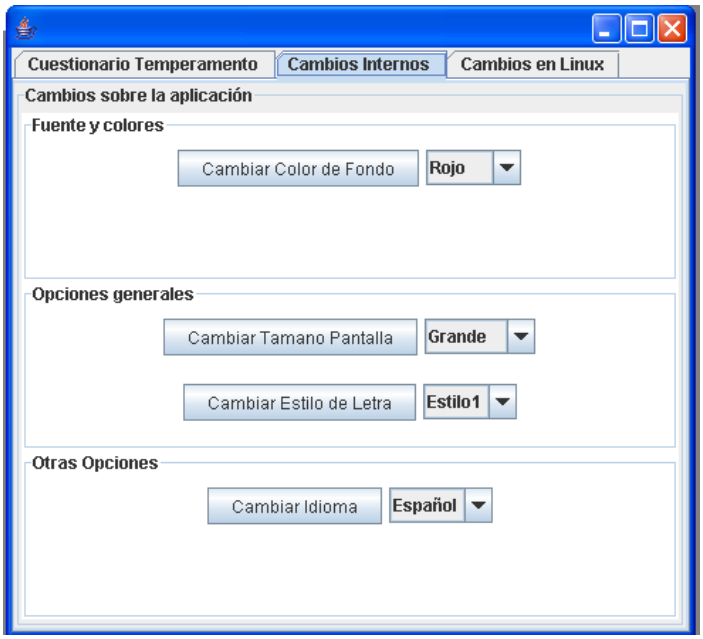
- **Permitir WallPaper:** permite poner imágenes en el fondo de escritorio. Si está desactivado, permite cambiar el color de fondo.
- **Cambiar color del escritorio:** cambia el color de fondo del escritorio al color seleccionado, guardándolo para futuras ejecuciones.
- **Cambiar el WallPaper:** cambia la imagen de fondo del escritorio. Debe de estar activada la opción de “Permitir WallPaper”.
- **Cambiar tamaño de ratón:** cambia el tamaño del puntero del ratón a “Grande”, “Mediano” o “Pequeño”.
- **Establecer ratón para zurdos:** pone el ratón para zurdos.
- **Actualizar GTK:** cambia el Theme de las ventanas de GNOME a la seleccionada. Este cambio afectará a los futuros ejecuciones.
- **Actualizar Iconos:** cambia el Theme de los iconos del escritorio de GNOME al seleccionado. Este cambio afectará a los futuras ejecuciones.
- **Habilitar sonido de eventos:** permite habilitar el sonido de los eventos que se produzcas en Linux.
- **Actualizar procesos:** saca una lista con los procesos del usuario actualizada.
- **Priorizar procesos:** da la prioridad máxima al proceso seleccionado de la lista.

Si realizamos algún cambio sobre estos elementos, estos serán guardados por el programa para futuras ejecuciones, modificando también su propio prototipo para que futuros usuarios similares.

Para activar el cambio solo es necesario seleccionar el nuevo valor y pulsar al botón correspondiente, realizándose la actualización inmediatamente.

6.1.4 Cambios internos de la aplicación

El usuario también puede modificar el aspecto del propio programa con la pestaña correspondiente.



- **Cambiar color de fondo:** permite cambiar el color de fondo de la aplicación, no de Linux, al color seleccionado. Este cambio es temporal y no permanecerá en las sucesivas ejecuciones del programa.

- **Cambiar el tamaño de la pantalla:** cambiar el tamaño de la ventana de la aplicación a tamaño “Grande”, “Mediano” o “Pequeño”. Este cambio es temporal y no permanecerá en las sucesivas ejecuciones del programa.

- **Cambiar estilo de letra:** cambia el estilo de letra usado en la aplicación. Hay tres estilos de letras diferentes. Este cambio no permanece en las sucesivas ejecuciones del programa.

- **Cambiar idioma:** cambia el idioma de la aplicación. Están disponibles el inglés y el español.

6.2 Manual del sistema

6.2.1 Requisitos

PERSIA requiere de la instalación previa del JDK 1.4 o superior con su respectiva máquina virtual. Así como un escritorio GNOME en Linux. Debido a la gran variedad de distribuciones no vamos a entrar a explicar como se instala una versión Linux con GNOME ya que en la actualidad se ha simplificado mucho el proceso, estando al alcance de todos los usuarios.

6.2.2 Instalación de Java en Linux

1. En la URL Java.sun.com descargar el paquete que contiene el RPM de *Java™ 2 Runtime Environment* 1.4.2 desde http://www.java.com/en/download/linux_manual.jsp.

2. Se hace que el fichero jre-1_5_0_06-linux-i586-rpm.linux sea ejecutable:
   - `chmod +x jre-1_5_0_06-linux-i586-rpm.bin`

3. Ejecute `jre-1_5_0_06-linux-i586-rpm.bin`:
   - `./jre-1_5_0_06-linux-i586-rpm.bin`

4. Lea la licencia y confirme que acepta los términos de la misma. Una vez hecho lo anterior, se extraerá el paquete RPM `jre-1_5_0_06-linux-i586.rpm`.

5. Como root instale jre-1_5_0_06-linux-i586.rpm:
   - Su
   - `rpm -Uvh jre-1_5_0_06-linux-i586.rpm`

6. Crear un fichero `/etc/profile.d/java.sh` para poder incluir la ruta de binarios de Java (`/usr/java/jre1.5.0_06/bin`) **siempre antes** de las rutas predeterminadas de ejecutables del sistema.
   - ```
     export PATH=/usr/java/jre1.5.0_06/bin:$PATH
     JAVA_HOME="/usr/java/jre1.5.0_06/"
     export JAVA_HOME
     ```

7. Haga ejecutable `/etc/profile.d/java.sh`:
   - `chmod 755 /etc/profile.d/java.sh`

8. Para comprobar si Java™ quedó instalado correctamente, ejecute lo siguiente desde una terminal.
   - `which java`
     ```

Con estos pasos podrá instalar y comprobar que la versión JDK 1.5 se ha instalado correctamente.

Prácticamente cualquiera de las actuales distribuciones de Linux trae la posibilidad de instalar GNOME. Si su distribución de Linux no contiene GNOME instalado, puede dirigirse a http://www.gentoo.org/doc/es/gnome-config.xml?style=printable como guía de referencia donde se le indicarán los pasos para su instalación.

Instalación PERSIA:

Se entrega un archivo comprimido contiene:

- JAR con la aplicación
- Ficheros de configuración
- Ontologías Base de la aplicación
    - La carpeta “ontología 10 usuarios” contiene ontologías de diez usuarios que han usado el sistema, si se quiere utilizar está al arrancar la aplicación solo hay que sustituir los archivos que contienen la carpeta por los originales.
- Ficheros MP3 para posibles variaciones basadas en sonido

Para su instalación simplemente deberá de descomprimir el fichero en un directorio y ejecutar el JAR.

Para su ejecución no hace falta que el usuario tenga privilegios especiales simplemente con tener permiso del root para ejecutar el archivo JAR serás suficiente para un correcto funcionamiento del sistema.

Nota: para ejecutar un JAR se debe de lanzar desde la consola el comando “java –jar nombreFichero.jar”

Una de las posibles variaciones sobre la que trabaja PERSIA al personalizar el escritorio es lanzar un reproductor de audio con una canción que se adapte a los gustos del usuario. El sistema utiliza un reproductor multimedia llamado Totem Movie Player 1.0.5 que hace uso de la librería de GNOME xine-lib 1.0.1. Las ultimas distribuciones tienen este reproductor instalado por defecto, en caso de que el sistema sobre el que va a trabajar PERSIA no tenga instalado Totem vamos a dar unos sencillos pasos para su instalación.

6.2.3 Instalación de Totem en GNOME

1. En la dirección http://ftp.gnome.org/pub/GNOME/sources/totem/1.0/totem-1.0.5.tar.gz nos descargamos el paquete del programa.
2. Descomprimimos el fichero con la orden gzip -d totem-1.0.5.tar.gz
3. Se hace que el fichero INSTALL sea ejecutable:

```
chmod +x INSTAL
```

4. Se ejecuta el archivo de la instalación tecleando ./INSTALL
5. Se aceptan las condiciones y se configuran la ruta de los archivos.

6.2.4 Descripción detallada de las ontologías

PERSIA utiliza ontologías para la representación del conocimiento dentro del sistema, vamos a explicar como se estructuran.

Hay cuatro ontologías con las que trabaja el sistema:

- PERSonto: Es una ontología vacía que sirve para que en ella se importen todas las demás y así trabajar con las ontologías como una sola unidad.

- TEMPonto: Sirve para almacenar los distintos temperamentos de los usuarios.

- USERonto: Guarda la información de los usuarios que trabajan con el sistema. Esta compuesta de los siguientes campos.
  - UserType:
    - Nombre: Nombre del userType. Al principio solo va a contener las cuatro personalidades “extremas”, es decir los usuarios que son 100% racional y 0% las otras, etc.
    - hasArtisanProp: Porcentaje de artesano que tiene la personalidad del userType.
    - hasGuardianProp: Porcentaje de guardián que tiene la personalidad del userType.
    - hasIdealistProp: Porcentaje de idealista que tiene la personalidad del userType.
    - hasRationalProp: Porcentaje de racional que tiene la personalidad del userType.
    - hasArtisanVariations: Variaciones del sistema asociadas a la personalidad artesano.
    - hasGuardianVariations: Variaciones del sistema asociadas a la personalidad guardián.
    - hasIdealistVariations: Variaciones del sistema asociadas a la personalidad idealista.
    - hasRationalVariations: Variaciones del sistema asociadas a la personalidad racional.

        - hasTemperamentVariations: En este campo se guardan todas las variaciones de todas las personalidades, es decir aquí estarán almacenadas las variaciones de hasArtisanVariations, hasGuardianVariations, hasIdealistVariations y hasRationalVariations.
        - hasTemperamentProportions: Guarda las proporciones de la personalidad del userType.

    - User:
        - Nombre: Nombre del usuario en el sistema. Este nombre es único y sirve para cargar tus variaciones cuando se arranca el programa.
        - hasQuestionResult: Respuestas que ha proporcionado el usuario al contestar el cuestionario corto Keirsey.
        - hasArtisanProp: Porcentaje de artesano que tiene la personalidad del usuario.
        - hasGuardianProp: Porcentaje de guardián que tiene la personalidad del usuario.
        - hasIdealistProp: Porcentaje de idealista que tiene la personalidad del usuario.
        - hasRationalProp: Porcentaje de racional que tiene la personalidad del usuario.
        - hasPersonalData: Puede guarda cualquier dato de interés sobre el usuario.
        - hasUserAdaptationsFromReasoner: Adaptaciones (variaciones de GNOME) que ha deducido el sistema basándose en el test de personalidad de Keirsey rellenado por el usuario.
        - hasUserAdaptationsFromFeedback: Adaptaciones que el usuario ha modificado sobre las que le había propuesto el sistema. Cuando se rellena el test de Keirsey PERSIA deduce una serie de modificaciones del escritorio GNOME y las ejecuta. Una vez hecho esto el usuario puede cambiar estas variaciones, pues estas modificaciones son las que se guardan en este campo.
        - hasUserAdaptations: Contiene las variaciones de los campos hasUserAdaptationsFromReasoner y hasUserAdaptationsFromFeddBack.
        - hasUserType: A cada usuario le corresponde un userType, el nombre de ese userType se guarda en este campo.
        - hasTemperamentProportions: Guarda las proporciones de la personalidad del usuario.

- VARIonto: Guarda la información de todas las adaptaciones que se pueden hacer en GNOME y la forma de ejecutarlas.
    - Variation: Este atributo esta jerarquizado, es decir tiene subclases que conforman una estructura donde se clasifican las distintas variaciones. Por ejemplo: Variation -> VarOverComputer -> ActionCustomized -> EventRemainer en esta ruta nosotros guardamos las variaciones que hace ejecutarse un reproductor de música con una canción que se ajuste al temperamento del usuario que esta usando el sistema, otras rutas posibles pueden ser Variation -> VarOverComputer -> ActionGUI -> GuiLook ->

GuiLookBackgroundColor aquí irían las variaciones relacionadas con cambiar el color de fondo del escritorio. Siguiendo esta forma de proceder hemos clasificado todas las variaciones que el sistema es capaz de hacer.

- Nombre: Nombre asociado a la variación.
- affectsToTraits: Rasgo al que afecta la variación.
- hasExecutionSteps: Este campo guarda las ordenes a ejecutar para que se produzca un cambio en el sistema.

- ExecutionStep:
  - Nombre: Nombre asociado al paso de ejecución.
  - hasActivationCommand: Comando a ejecutar para que se lance el cambio en GNOME.
  - hasActivationParameters: Campo de tipo ExecParam que junto con hasActivationCommando forman la instrucción a ejecutar.
- ExecParam:
  - Nombre: Nombre del parámetro.
  - hasExecParamUseInStep: Guarda una parte del parámetro a ejecutar.
  - hasExecParamName: Guarda la ruta hasta la clave dentro del configurador de GNOME que vamos a cambiar para que se produzca un cambio apreciable en el sistema. Por ejemplo para cambiar el color de fondo de pantalla su clave se encuentra en /desktop/gnome/background/primary_color.
  - artisanRange: Posibles valores que puede tomar una variación concreta para personas en donde predomine el temperamento artesano.
  - guardianRange: Posibles valores que puede tomar una variación concreta para personas en donde predomine el temperamento guardian.
  - idealistRange: Posibles valores que puede tomar una variación concreta para personas en donde predomine el temperamento idealista.
  - rationalRange: Posibles valores que puede tomar una variación concreta para personas en donde predomine el temperamento racional.
  - possibleAdaptationRange: Conjunto de todos los valores posibles a ejecutar incluidos.
  - hasExecParamValue: Valor concreto que va a tomar el cambio sobre GNOME.

6.2.5 Como incluir nuevas variaciones en PERSIA

En este apartado se darán los pasos básicos para añadir una nueva variación a PERSIA

Las variaciones deben de estar incluidas en los prototipos para que se expandan por toda la aplicación. En PERSIA hemos creado una clase que se encarga de cargar estos prototipos de manera automática con unas variaciones predefinidas. Por lo que una primera manera de añadir una nueva variación sería modificar la clase

"AuxiliarThemesMusicaBack". Para modificar esta clase se necesitan conocimientos previos del paquete "com.clasesOntologias".

La otra opción en la que nos centraremos en este apartado es usando Protegé. El uso de Protegé no es obligatorio, pero si se recomienda usar algún editor de ontologías, aunque bien es cierto que se podría hacer con un editor de texto.

Para añadir una nueva variación necesitamos modificar los ficheros USERonto.owl y VARIonto.owl.

Primero se harán las modificaciones pertinentes en el fichero VARIonto.owl. Este fichero contiene las variaciones que tendrán después los usuarios. Cada variación está formado por varios componentes: variation, execParam, executionStep.

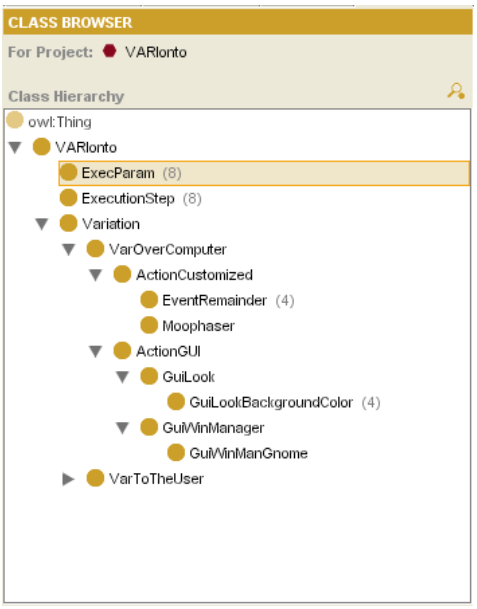
En la creación de una nueva variación habrá que ir creando cada uno de estos componentes independientemente y en un orden concreto. Por último habrá que añadir la nueva variación a los UserType que creamos convenientes, con el uso de la aplicación y la creación de nuevos usuarios esta nueva variación se irá expandiendo por las ontologías.

Ejemplo de añadir una nueva variación

Vamos a presentar un ejemplo de cómo se crearía una nueva variación para lanzar canciones MP3 desde una aplicación Java que hayamos construido nosotros. Esta aplicación se llamara "ReproductorMP3".

Creación del ExecParam

Lo primero que debemos de crear en uno o varios ExecParam dependiendo del número de pasos que son necesarios para ejecutar la variación. En nuestro caso, sólo será necesario crear un ExecParam ya que con una sóla orden se realiza todo lo que queremos. Si por ejemplo queremos introducir una variación que consta de varios pasos que deben de realizarse en un orden concreto deberíamos de crear un ExecParam para cada uno de estos pasos.



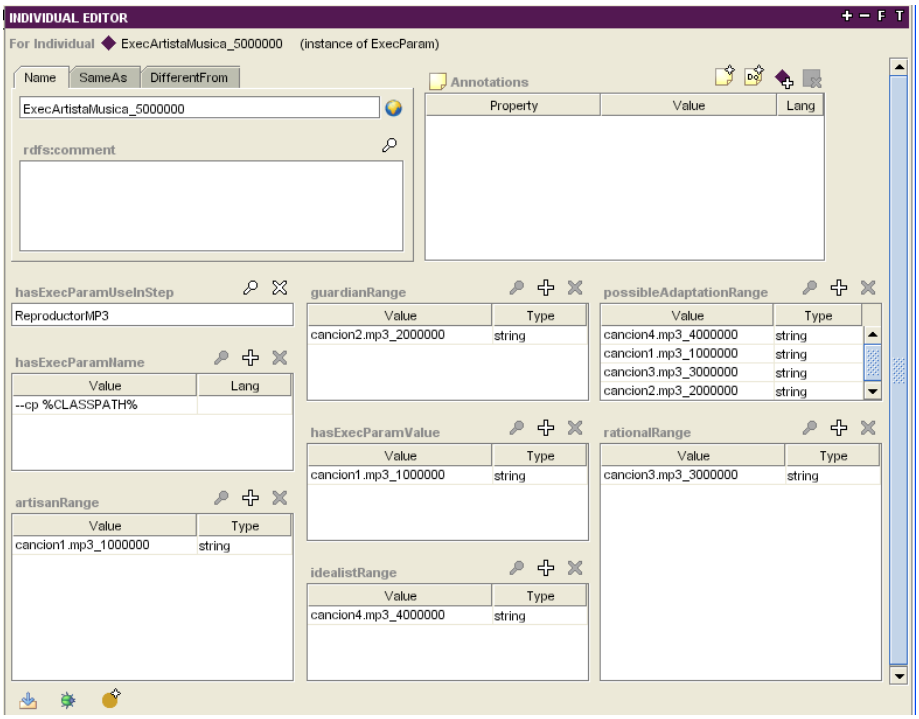
A nuestro ExecParam lo vamos a llamar ExecArtistaMusica_5000000. Es importante seguir la nomenclatura "_NUMERO". Ya que servirá para mantener el orden en caso de que haya que ejecutar varios ExecParam de una manera determinada. Después internamente el programa ejecutara los diferentes ExecParam de menor a mayor siguiendo estos números.

En hasExecParamUseInStep debemos de poner el nombre del programa, en este caso el nombre de la clase Java. **No el comando para lanzarlo que sería java.**

En el atributo hasExecParamName debemos de poner los posibles parámetros que tenga el comando.

El resto de los atributos tiene los posibles valores con los que se puede lanzar el programa. Estos valores también contienen la misma nomenclatura ya que será importante para realizar la adaptación.

Como última observación la lista "possibleAdaptationsRange" debe tener todos los valores que estén presentes en el resto de las listas. Es decir, esta lista es la unión de todas las demás.

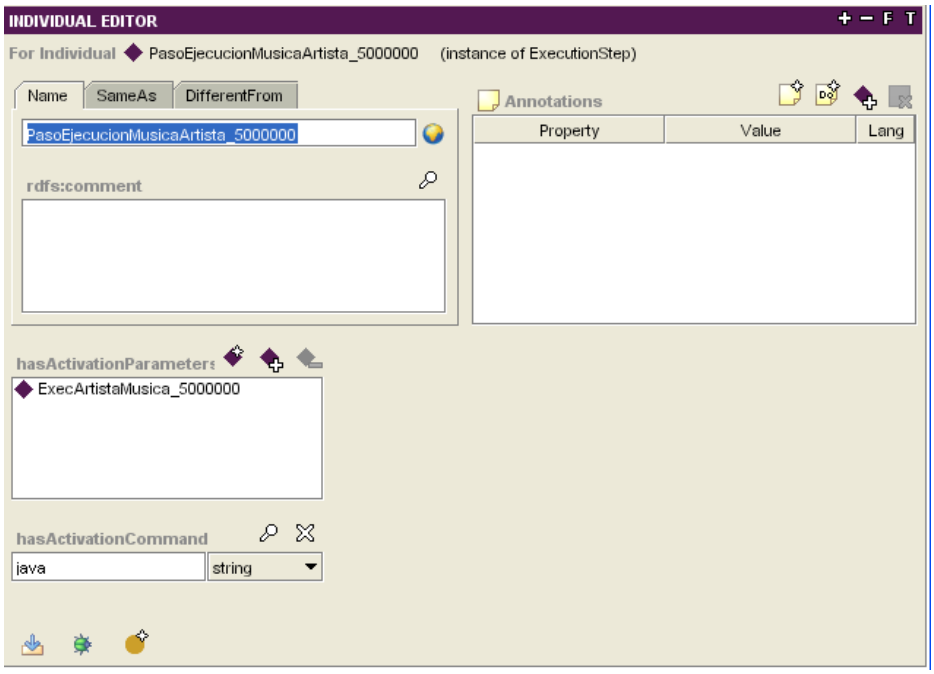


Creación del ExecutionStep

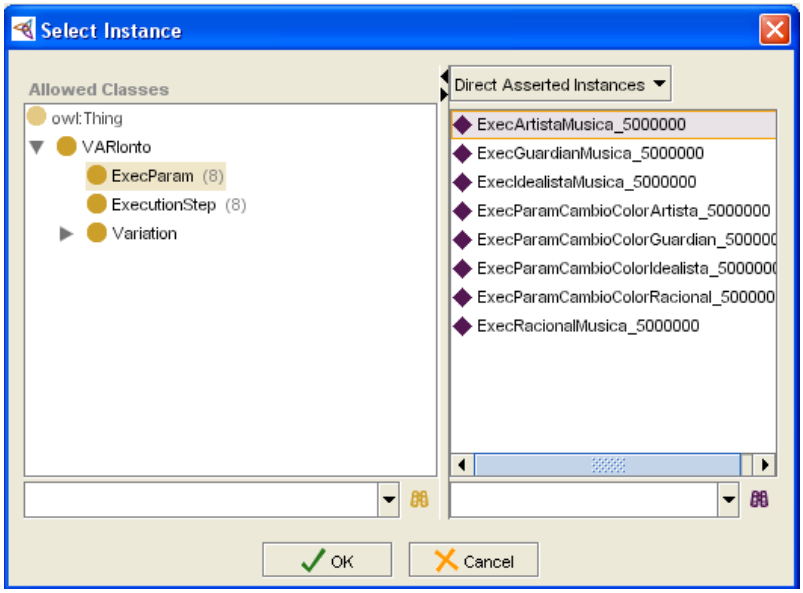
El “ExecutionStep” contiene la lista de los “ExecParam” a ejecutar y el comando para lanzarlo.

Como en el caso anterior, lo primero es definir el nombre con la nomenclatura. Nosotros lo hemos llamado “PasoEjecucionMusicaArtista_5000000”. De esta manera si en el futuro nuestra variación requiere de más de un “ExecParam” deberemos de indicarlo aquí.

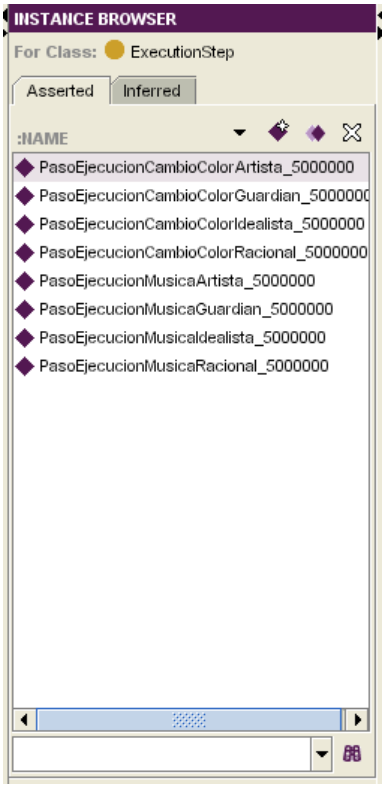
El segundo paso es decidir que programa se va a encargar de lanzar nuestra aplicación o cambio. De esta manera en el atributo “hasActivationCommand” debemos de poner **java**.



Por último disponemos de una lista donde tendremos que añadir cada uno de los "ExecParam" que queremos que se ejecuten. Sólo debemos de seleccionar el ExecParam que hemos creado anteriormente y añadirlo.



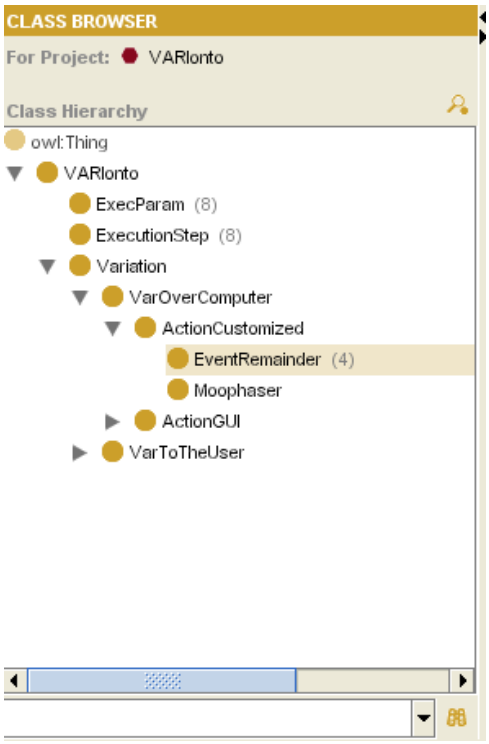
De esta manera tenemos nuestro “ExecutionStep creado completamente y lo debemos de poner ver en la pestaña izquierda.



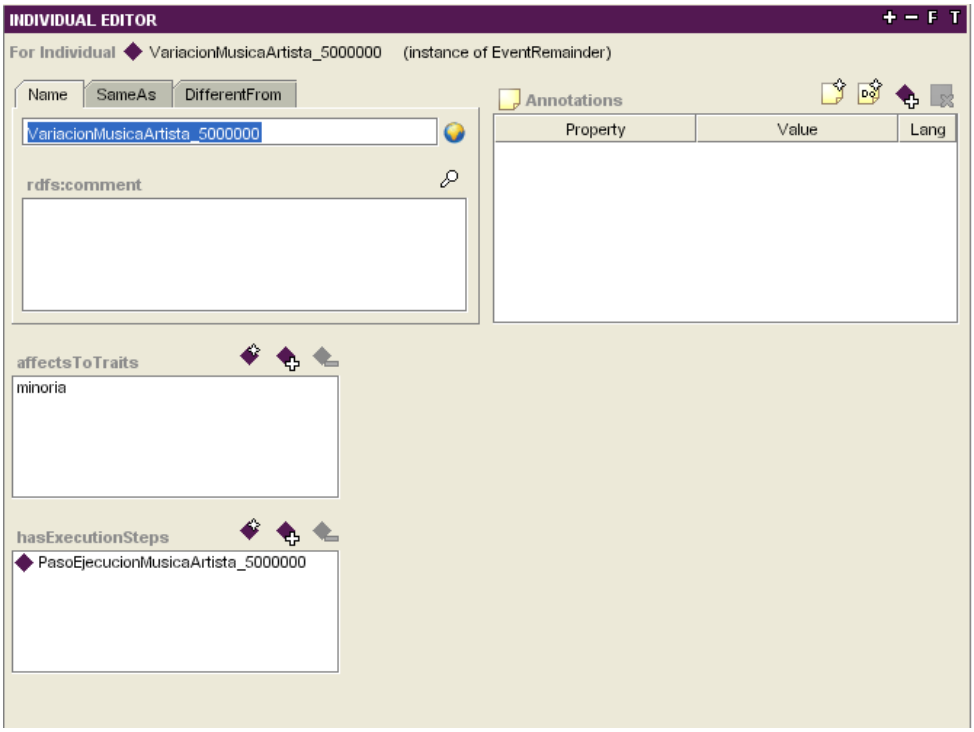
Creación del Variation

Este es el úlitmo paso de la creación, después sólo tendremos que añadirlo al UserType que queramos.

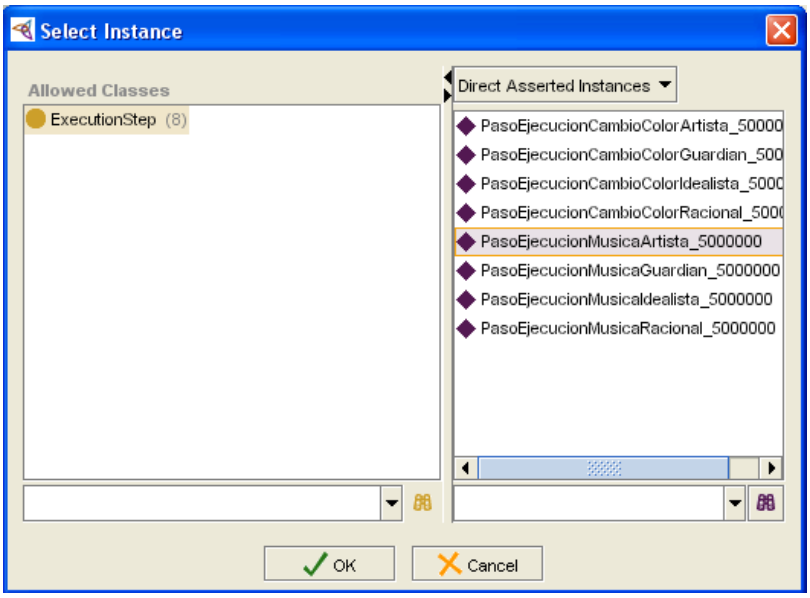
El primer paso es decidir en que parte del árbol de la ontología vamos a añadir la variación. Aunque esto no afecta a la aplicación internamente siempre es bueno que las cosas estén clasificadas lo mejor posible. Nosotros esta variación la vamos a añadir en EventRemainer.



Una vez sepamos donde la vamos a colocar debemos de decidir el nombre que queremos dar a la variación. En nuestro caso “VariacionMusicaArtista_500000”.



A parte de esto, sólo nos falta añadir los “ExecutionStep” que queramos que se ejecuten en esta variación. Para realizar esto sólo tenemos que seleccionarlo en el listado que se nos abre.



De esta manera tendremos creada la variación y se aparecerá en la lista de la izquierda.

Modificación en el UserType

Por último sólo nos queda añadir la nueva variación a los UserType que quedamos. Para realizar esto debemos de abrir con el Protegé la ontología de usuarios y añadirlo en la lista de variaciones que queramos dependiendo de si es una variación de artista, de guardian, etc.

6.3 Pruebas realizadas

Para comprobar el buen funcionamiento del sistema se realizaron una serie de pruebas.

La normal general ha sido comprobar el funcionamiento de una clase o función por separado antes de ensamblarse con el programa principal, pero hay algunas pruebas que solo se pueden hacer teniendo el sistema completamente montado e instalado.

Prueba 1: Comprobación del cuestionario corto de Keirsey.

Motivación: Como PERSIA basa su adaptación de GNOME en el temperamento que viene dado por el cuestionario corto de Keirsey es importante que el sistema "obligue" al usuario a rellenar el test correctamente, interrumpiendo si hiciera falta el ciclo del programa.

Comprobación: Rellenar mal el formulario, repitiendo valores en las repuestas.

Resultado: El programa logra detectar el error. Muestra un aviso por pantalla, se interrumpe y obliga al usuario a rellenar el cuestionario correctamente.

Prueba 2: Visualizar correctamente los cambios de GNOME

Motivación: El correcto funcionamiento de los cambios en el escritorio Linux es la parte más importante del proyecto y es la que primero probamos.

Comprobación: Ejecutar el ciclo CBR completo del programa después de haber rellenado el cuestionario de Keirsey.

Resultado: Se comprueba que después de ejecutar la primera variación, no ejecuta las demás.

Problema detectado: El programa al leer las variaciones de las ontologías las ejecutaba en orden. Invoca la orden Process p = run.exec(comando) que utiliza el Java

Native Interface para comunicarse con el kernel de Linux y ejecuta el comando pasado como parámetro.

El problema esta en la orden p.waitFor() que espera a que el proceso acabase. Si este proceso que se lanza primero coincide con por ejemplo el que reproduce el audio, hasta que no se cierra el reproductor (mata el proceso) no se continúan ejecutando los demás.

Solución: Comentar esa línea.

Prueba 3: Recuperar un usuario del sistema

Motivación: Verificar que un usuario se guarda correctamente en el sistema.

Comprobación: Arrancamos PERSIA e intentamos cargar un usuario existente.

Resultado: El usuario se recuperó con éxito, cargando todas las variaciones que se habían deducido anteriormente de su test de Keirsey.

Prueba 4: Cambios manuales en las adaptaciones

Motivación: Comprobamos que el sistema da la posibilidad de modificar las adaptaciones que PERSIA realiza sobre GNOME por si no quedamos satisfechos y que estas variaciones se guardan.

Comprobación: Una vez ejecutadas las modificaciones de nuestro usuario, modificamos alguna de ellas, volviendo a cerrar nuestra sesión.

Resultado: Al volver a cargar el usuario observamos que las variaciones se han guardado con éxito.

Prueba 5: Cambios internos del programa

Motivación: Testamos que el programa admite los cambios sobre su propia configuración.

Comprobación: Probamos a cambiar algún aspecto del programa como el color de fondo de los paneles.

Resultado: Observamos que los cambios se producen sin problemas.

7. Anexos

Cuestionario largo de personalidad de Keirsey

1. Cuando suena el teléfono
    (a) corres para contestar primero
    (b) esperas a que otra persona conteste

2. Eres más
    (a) observador que instrocpectivo
    (b) instrocpectivo que observador

3. Es peor
    (a) estar en las nubes
    (b) seguir una rutina

4. Con la gente, por le general eres mas
    (a) firme que amable
    (b) amable que firme

5. Te sientes más cómodo al hacer
    (a) juicios críticos
    (b) juicios valorativos

6. El desorden en tu lugar de trabajo es algo
    (a) para cuyo arreglo le das tiempo
    (b) que toleras bastante bien

7. Tu estilo es
    (a) tomar decisiones rápidamente
    (b) elegir con cuidado

8. Cuando haces fila, a menudo
    (a) platicas con otros
    (b) te dedicas a lo tuyo

9. Eres más
    (a) sensato que inclinado a lo ideático
    (b) inclinado a lo ideático que sensato

10. Te interesa más
    (a) lo que es real
    (b) las que es posible

11. Cuando tomas una decisión es mas probable que lo hagas basado en
    (a) información
    (b) deseos

12. Al evaluar a otros tiendes a ser
    (a) objetivo e impersonal
    (b) amistoso y personal

13. Prefieres que los contratos
    (a) se firmen se sellen y se entreguen
    (b) se cierren con un apretón de manos

14. Te sientes más satisfecho con
    (a) un producto terminado
    (b) un trabajo que está realizando

15. En una fiesta tú
    (a) interactúas con mucha gente incluso extraños
    (b) interactúas con algunos amigos

16. Tiendes a ser más
    (a) objetivo que especulativo
    (b) especulativo que objetivo

17. Te gustan los escritores quienes
    (a) dicen claramente lo que piensan
    (b) usan metáforas y simbolismos

18. Que te atrae más
    (a) un modo de pensar coherente y sólido
    (b) relaciones armoniosas

19. Si debes desilusionar a alguien por lo general eres
    (a) franco y directo
    (b) calido y considerado

20. En el trabajo quieres que tus actividades estén
    (a) programadas
    (b) sin programar

21. Prefieres con más frecuencia
    (a) declaraciones finales e invariables
    (b) declaraciones tentativas y preliminares

22. Interactuar con extraños
    (a) te llena de alegría
    (b) agota tu energía

23. Los hechos:
    (a) hablan por ellos mismos
    (b) ilustran principios

24. Los visionarios y teóricos te parecen
    (a) un poco irritantes

(b) bastante fascinantes

25. En una discusión acalorada
    (a) no cedes en tus ideas
    (b) buscas puntos en común

26. Es mejor ser
    (a) justo
    (b) misericordioso

27. En el trabajo, para ti es más natural
    (a) señalar errores
    (b) intentar complacer a otros

28. Te sientes más cómodo
    (a) después de tomar una decisión
    (b) antes de tomar una decisión

29. Por lo general
    (a) dices inmediatamente lo que piensas
    (b) mantienes los oídos atentos

30. El sentido común es
    (a) por lo general confiable
    (b) con frecuencia cuestionable

31. A menudo los niños no
    (a) ayudan lo suficiente
    (b) utilizan su fantasía lo suficiente

32. Cuando estas a cargo de otros sueles ser
    (a) firme e inflexible
    (b) indulgente y misericordioso

33. Con mayor frecuencia eres
    (a) una persona juiciosa
    (b) una persona afectuosa

34. Eres propenso a
    (a) definir las cosas
    (b) explorar las posibilidades

35. En la mayor parte de las situaciones eres más
    (a) deliberado que espontáneo
    (b) espontáneo que deliberado

36. Piensas en ti mismo como
    (a) una persona sociable
    (b) una persona reservada

37. Eres con más frecuencia
    (a) una persona practica
    (b) una persona caprichosa

38. Hablas más de
    (a) particularidades que de generalidades
    (b) generalidades que de particularidades

39. Cual te parece un cumplido
    (a) "ahí va una persona lógica"
    (b) "ahi va una persona sentimental"

40. Que rige más
    (a) tus pensamientos
    (b) tus sentimientos

41. Cuando terminas un trabajo te gusta
    (a) dejar todo bien terminado y resuelto
    (b) pasar a otra cosa

42. Prefieres trabajar
    (a) con plazos limite
    (b) simplemente cuando sea

43. Eres el tipo de persona que
    (a) es muy platicadora
    (b) no se pierde de mucho

44. Te inclinas a entender lo que se dice
    (a) de modo más literal
    (b) de modo más figurado

45. Ves con más frecuencia
    (a) lo que está enfrente de ti
    (b) lo que solo puede imaginarse

46. Es peor ser
    (a) un sentimental
    (b) un hombre de piedra

47. En circunstancias difíciles, a veces eres
    (a) poco comprensivo
    (b) muy comprensivo

48. Tiendes a elegir
    (a) con bastante cuidado
    (b) de modo bastante impulsivo

49. Te inclinas a andar más
    (a) apresurado que relajado

(b) relajado que apresurado

50. En el trabajo tiendes a
    (a) ser sociable con tus colegas
    (b) aislarte

51. Es mas probable que confíes
    (a) en tus experiencias
    (b) en tus ideas

52. Te inclinas más a sentirte
    (a) una persona realista
    (b) una persona fuera de la realidad

53. Te consideras
    (a) una persona dura
    (b) una persona compasiva

54. Valoras más en ti el hecho de que eres
    (a) sensato
    (b) dedicado

55. Por lo general quieres que las cosas estén
    (a) arregladas y decididas
    (b) solo esbozadas

56. Dirías que eres más
    (a) serio y decidido
    (b) despreocupado

57. Te consideras
    (a) un buen conversador
    (b) un buen escucha

58. En ti mismo valoras
    (a) que tienes los pies en la tierra
    (b) que tienes una imaginación muy despierta

59. Te interesan más
    (a) los puntos esenciales
    (b) las alusiones

60. Cual te parece un defecto mayor
    (a) ser muy compasivo
    (b) ser muy desapasionado

61. Influye más en ti
    (a) la evidencia convincente
    (b) una petición emotiva

62. Te sientes mejor
    (a) al llegar al final de las cosas
    (b) al dejar las opciones abiertas

63. Por lo general prefieres
    (a) asegurarte de que todo está arreglado
    (b) dejar que las cosas pasen de forma natural

64. Tiendes a ser
    (a) una persona a quien es fácil dirigirse
    (b) un tanto reservado

65. Prefieres en los relatos
    (a) acción y aventura
    (b) fantasía y heroísmo

66. Para ti es más fácil
    (a) usar a las personas
    (b) identificarte con otras personas

67. Cuales de estas opciones prefieres
    (a) fuerza de voluntad
    (b) fuerza emocional

68. Te consideras básicamente
    (a) duro
    (b) susceptible

69. Tiendes a notar
    (a) el desorden
    (b) oportunidades para hacer cambios

70. Eres más
    (a) rutinario que caprichoso
    (b) caprichoso que rutinario

Cuestionario corto de personalidad de Keirsey

1. Prefiero estudiar
    (a) artes y manualidades
    (b) literatura y humanidades
    (c) negocios y finanzas
    (d) ciencia e ingeniería

2. Me siento mejor conmigo mismo cuando
    (a) mis acciones son elegantes
    (b) estoy en armonía con alguien
    (c) soy tan confiable como una roca
    (d) utilizo mi ingenio

3. En cuanto a mi estado de ánimo, con más frecuencia estoy o soy
    (a) emocionado y estimulado
    (b) entusiasmado e inspirado
    (c) precavido y prudente
    (d) tranquilo y distante

4. Siempre insisto en
    (a) perfeccionar mis habilidades
    (b) ayudar a que otros se afirmen
    (c) ayudar a que otros hagan lo correcto
    (d) explicarme cómo funcionan las cosas

5. Cuando se trata de hacer algo soy
    (a) práctico y oportunista
    (b) compasivo y altruista
    (c) concienzudo y diligente
    (d) eficiente y pragmático

6. Me respeto más por
    (a) ser audaz y con espíritu de aventura
    (b) ser bondadoso y con buena voluntad
    (c) realizar buenas obras
    (d) ser autónomo e independiente

7. Me inclino más a confiar en
    (a) impulsos y caprichos
    (b) intuiciones e insinuaciones
    (c) costumbres y tradiciones
    (d) la razón pura y la lógica formal

8. A veces anhelo
    (a) causar una gran impresión y tener impacto
    (b) perderme en mis sueños románticos
    (c) ser un miembro legítimo y valorado
    (d) hacer un gran descubrimiento

9. En mi vida busco
    (a) aventuras y emociones
    (b) conocimiento de mi mismo
    (c) protección y seguridad
    (d) métodos de operación eficaces

10. Al mirar al futuro
    (a) aseguro que ocurrirá algo bueno
    (b) creo en la bondad innata de la gente
    (c) hay que ser sumamente cuidadoso
    (d) es mejor ser cauteloso

11. Si fuera posible, me gustaría convertirme en
    (a) un artista virtuoso
    (b) un visionario sabio
    (c) un ejecutivo de alto rango
    (d) un genio de la tecnología

12. Funcionaría mejor en un trabajo si me ocupara de
    (a) herramientas y equipo
    (b) desarrollo de recursos humanos
    (c) materiales y servicios
    (d) sistemas y estructuras

13. Como dirigente de acciones, sobre todo miro
    (a) las ventajas inmediatas
    (b) las posibilidades futuras
    (c) la experiencia pasada
    (d) las condiciones necesarias y suficientes

14. Confío más en mí mismo cuando soy
    (a) flexible y adaptable
    (b) genuino y autentico
    (c) respetable y honorable
    (d) resuelto y tesonero

15. Aprecio cuando los demás
    (a) me sorprenden con su generosidad
    (b) reconocen mi personalidad verdadera
    (c) expresan su gratitud
    (d) preguntan cuáles son mis motivos

16. Cuando pienso en mis desventuras
    (a) por lo general me río
    (b) a menudo me pregunto por qué ocurren
    (c) intento sacar el mejor partido
    (d) las considero desde una perspectiva amplia

8. Bibliografía

- Jena Semantic Web Framework, http://jena.sourceforge.net/
- Pagina web de la profesora de la Facultad de Informática de la Universidad Complutense de Madrid. María Belén Díaz Agudo, http://www.fdi.ucm.es/profesor/belend/
- Jena-dev: Jena developers, http://groups.yahoo.com/group/jena-dev/
- Documentación de la API de JAVA
- Página oficial de Protégé, http://protege.stanford.edu/
- Sitio web del temperamento de Keirsey, http://www.keirsey.com/
- Razonador OWL Pellet, http://www.mindswap.org/2003/pellet/
- GNOME hispano, http://www.es.gnome.org/
- GNOME internacional, http://www.gnome.org/

9. Palabras clave

- Ontologías
- CBR
- Keirsey
- Pellet
- Jena
- Personalizador
- GNOME
- Conector
- Temperamento

Autorizamos a la Universidad Complutense de Madrid a la difusión y utilización con fines académicos, no comerciales y, siempre y cuando se mencione a los autores, tanto esta memoria, como el código, la documentación y/o el prototipo desarrollado.

Los autores:

Julio Alberto Martínez Real

Alberto Pedrera Castela

Guillermo Ortiz Fernández