Diseño e implementación de una interfaz orientada a la docencia para el paquete estadístico R

Néstor Arocha Rodríguez
Tutora: Inmaculada Luengo Merino

Índice general

Agradecimientos

Agradezco a todos los que me apoyaron, especialmente a Luis, a Carlos, a Daniel y sobre todo a mi padre.

Capítulo 1

Introducción

1.1. Descripción del proyecto

El objetivo de *Diseño e implementación de una interfaz orientada a la docencia para el paquete estadístico R* es la creación de un programa de estadística que, en los términos del software libre, permita asistir las asignaturas de estadística. Como interfaz para R, ha de permitir el acceso sencillo a las funcionalidades de éste. Como programa de estadística, ha de complementar los estudios de estadística a los alumnos universitarios.

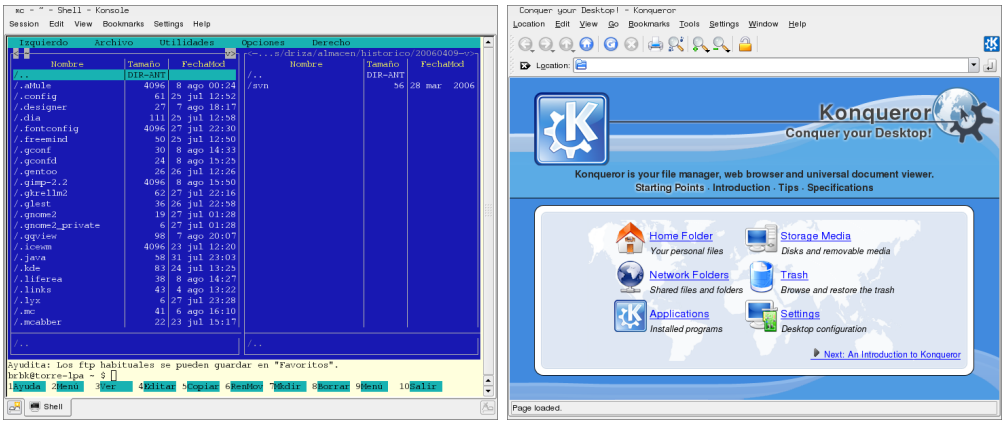
Para una definición más completa de este proyecto, procederemos con la definición de los conceptos asociados.

1.1.1. Definición de una interfaz gráfica de usuario

Una interfaz de usuario es un mecanismo de comunicación entre éste y la máquina. En contraposición con las basadas en texto, las interfaces gráficas permiten al usuario comunicarse con los elementos del programa de forma visual. Algunos elementos característicos de las interfaces de usuario son los siguientes:

- Iconos
- Ventanas
- Imágenes
- Botones

La Wikipedia [5] ofrece una definición alternativa:



(a) Midnight Commander, interfaz de texto

(b) Konqueror, interfaz gráfica

Figura 1.1: Aspecto de diferentes interfaces gráficas

> "Es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático"

Un ejemplo bastante ilustrativo de cambio de interfaz de texto a interfaz gráfica es la transición de los procesadores de texto de sus versiones en MS-DOS (Word Perfect 5.1) a las versiones modernas de Word Perfect, Word u OpenOffice. Otro ejemplo es el ilustrado en la figura 1.1 sobre gestores de ficheros.

1.1.2. Definición de programa de estadística

Los programas de estadística sirven para que los usuarios puedan hacer ciertas operaciones estadísticas sobre un conjunto de datos.

El manejo de datos

Los programas de estadística han de estar preparados para permitir manipular las variables. Especialmente efectiva debe ser la forma de obtener los datos de otros programas, ya que es un escenario bastante habitual recoger los datos de la salida de otros programas. El método más habitual de manipulación directa es el uso de "hojas de cálculo", que consiste en una rejilla que tiene una casilla por elemento, mostrando los valores distribuidos según

la variable a la que pertenecen. También han de permitir formas de manipulación más complejas, como las operaciones aritméticas con las variables o la conversión de tipos de variables.

Operaciones con los datos

Los programas de estadística pueden realizar diferentes tipos de operaciones. El abanico de operaciones que pueden ejecutar es realmente grande, y depende de cada programa. Alguna de las operaciones más importantes y frecuentes son:

Cálculos descriptivos: Son datos acerca de las variables estadísticas. No se pretende establecer alguna propiedad de las mismas, simplemente se pretende describirlas.

Contraste de hipótesis: Se intenta demostrar alguna propiedad acerca de una o más variables. Por ejemplo, si se busca demostrar si unos datos se ajustan a un tipo de distribución o no, o si se desea comparar la media de dos poblaciones, se emplean los contrastes de hipótesis.

Gráficos: Realizan una representación gráfica de un determinado aspecto de una o más variables. Hay muchos tipos de representaciones gráficas que se pueden presentar según el tipo de datos que se desee analizar.

Distribuciones: El usuario puede consultar la forma o la probabilidad asociada a un percentil de cualquier distribución. También es posible generar los datos de trabajo a partir de una distribución.

Visualización de resultados

Una vez realizados los cálculos sobre los datos, los programas de estadística muestran el resultado al usuario. En este caso, la diferencia entre los programas evaluados es poca, puesto que casi todos recurren a las tablas y a las imágenes según es necesario. La diferencia radica en la posibilidad de manejo de esta información y el formato de fichero de salida.

1.1.3. ¿Qué es el Software Libre?

Según la Wikipedia [8]:

> "Es el software que, una vez obtenido, puede ser usado, copiado, estudiado, modificado y redistribuido libremente"

Básicamente, es un software que posee una licencia que otorga las libertades anteriormente citadas. Este modelo de software está asociado a un movimiento de personas e instituciones con gran repercusión en Internet.

Un ejemplo claro de este movimiento es el desarrollo del sistema operativo Linux. Comenzó en 1991, y desde entonces no ha parado de crecer.

1.1.4. ¿Qué es R?

R es un lenguaje de programación y un entorno orientado a la estadística. Su estructura es altamente modular y tiene elementos de lenguaje de programación que permiten añadir nuevas funcionalidades fácilmente. Su licencia es GPL[1], por lo tanto permite que cualquier programa con la misma licencia utilice su código con total libertad (Pertenece al movimiento del Software Libre)

La mayoría de los programas de estadística utilizan un motor como R, es decir, existe una separación entre la interfaz y el motor de datos. R proviene de otro lenguaje con el mismo objetivo, S.

Según la web oficial[2], R incluye los siguientes elementos:

- Manejo efectivo de datos y almacenamiento
- Un conjunto de operadores para cálculos en arrays y matrices
- Una colección integrada de herramientas intermedias grande y coherente para el análisis de datos
- Ayuda gráfica para el análisis de datos, mostrable tanto en pantalla como en fichero
- Un lenguaje de programación (bien desarrollado) que incluye condicionales, bucles, funciones definidas por usuario, recursividad, y facilidades para la entrada y la salida

1.2. Entorno del proyecto

Una vez definido el proyecto, resulta conveniente explicar el origen y el entorno que ha acompañado al desarrollo.

[1] El acrónimo GPL viene de *General Public License*. Puede consultarse más información en la sección 3.7

[2] http://www.r-project.org/about.html

1.2.1. La Oficina del Software Libre

La OSL es una institución que nació en el año 2003, en el ámbito universitario. Una de las ideas que propulsó fue la creación de una bolsa de proyectos de carrera. La motivación de esta iniciativa es la implantación del software libre como modelo, y la reducción de la dependencia de la ULPGC de proveedores de software.

Otra iniciativa interesante es la elaboración de la distribución de Linux SILU, que contiene material especifico de la universidad, y que se distribuye de forma gratuita.

1.2.2. Motivación

La motivación de este proyecto es la de incentivar el desarrollo de software dentro de la universidad. Está relativamente extendida la idea de que el desarrollo del software es algo ajeno al estudiante universitario. Cuando un alumno requiere utilizar un software de alto costo, lo copia ilegalmente, o bien lo solicita a través de la universidad (si existe convenio) o, simplemente, se lo descarga.

El hecho de que exista la opción del desarrollo interno, da la posibilidad a los alumnos de ser emprendedores dentro de la universidad, de acceder a equipos de trabajo más cercanos al mundo laboral y, sobre todo, permite limitar el alcance de la idea de que el software es algo inaccesible, o algo que se "descarga".

1.2.3. Estado del software en general

La industria del software comenzó casi en paralelo con la de las computadoras. Sistemas operativos, aplicaciones de ofimática, utilidades multimedia o aplicaciones de cálculo son necesarias para la realización de nuestras tareas cotidianas.

Altamente relacionado con el modelo de propiedad intelectual, la forma de obtener compensación por la labor de la elaboración del software ha sido a través de licencia. Un usuario que desease usar un sistema operativo ha de pagar una licencia por éste.

Sin embargo, desde hace al menos dos décadas[3], existe el movimiento del Software Libre, respaldado por un conjunto de licencias que permiten la libre distribución del código fuente de los programas.

[3]El proyecto GNU, que es uno de los pilares del movimiento del software libre, comenzó en 1984

Ambos modelos coexisten desde entonces y, por lo general, aquellos programas que son de uso general y relativamente sencillos de implementar acaban teniendo una alternativa clara con software libre, como por ejemplo Firefox y OpenOffice.

El software en la universidad

Actualmente hay una mezcla de uso entre las aplicaciones de software libre y aplicaciones de software cerrado. Durante el tiempo que he cursado las diferentes asignaturas de la carrera, he conseguido identificar tres patrones diferentes:

Asignaturas que exigen modificación de código: En estas asignaturas se suele escoger software libre, ya que el software cerrado requiere algún acuerdo de cesión de código. Por ejemplo, en la asignatura Sistemas Operativos utilizamos el programa Nachos, que es un sistema operativo de código fuente disponible pensado para ser investigado y modificado.

Asignaturas que requieren software complejo: En este caso, las alternativas de software libre no suelen tener calidad suficiente. Por ejemplo, en Diseño de Sistemas Digitales, la herramienta Xilinx (Diseño de circuitos lógicos). También es el caso de las asignaturas de estadística y el SPSS.

Resto de asignaturas: En estos casos, existe alternativa en ambos modelos de software. El factor que hace inclinar la balanza en uno de los dos sentidos es el criterio del profesor, o la popularidad del programa en cuestión.

Estado del Software de estadística

Existen alternativas tanto en el modelo de software propietario como en el modelo libre.

Suites de código propietario El mercado de software de estadística incluye una serie de alternativas que han ido mejorando y estabilizando con el tiempo. Algunas de las suites de pago que se puede encontrar son las siguientes:

- Lesphinx
- Minitab
- SPSS

- Statgraphics
- S-PLUS

Una de las características destacables de este tipo de programas es el alto precio de las licencias. El uso de un programa u otro varía según decisiones institucionales o empresariales. Es notorio el hecho de que en ciertas universidades se empleé la suite SPSS como apoyo docente, mientras que en otras se utilice el Minitab. O también según países, ya que por ejemplo la suite Lesphinx es muy usado en Francia y apenas tiene relevancia en otros países.

Suites libres Alguna de las suites libres, cuyo código fuente se puede descargar en la red, son las siguientes:

- R
- Rcommander
- PSPP
- Gstat
- Rkward

El mercado del software libre es mucho más variable. Al no haber una interfaz gráfica claramente superior a las demás, las comunidades de usuarios están más dispersas que en el caso de las suites de pago. En el caso de los programas con interfaz de texto, sin duda alguna es R el que tiene mayor repercusión.

Capítulo 2

Objetivos

Los cálculos de estadística son importantes en multitud de trabajos, de hecho, en la mayoría de las carreras técnicas existe alguna asignatura dedicada a introducirlos. Por tanto, *Diseño e implementación de una interfaz orientada a la docencia para el paquete estadístico R* tiene como objetivo ser una interfaz gráfica para el lenguaje estadístico R. Los apartados fijados para cumplir dicha meta son los siguientes:

- Poder Crear ficheros de datos
- Asegurar la lectura de ficheros de datos creados en los formatos de texto, formato propio y html para salida
- Permitir las funciones más usuales con los datos (cortar, pegar, seleccionar, insertar, segmentar...)
- Implementar las transformaciones matemáticas con los datos
- Realizar un estudio descriptivo completo de la variable o variables de interés.
- Realizar todos los procedimientos básicos de estadística inferencial paramétrica y no paramétrica
- Realizar un estudio gráfico de los datos con posibilidades adecuadas según sean discretos o continuos.

2.1. Crear ficheros de datos

La creación de ficheros de datos consiste en permitir almacenar aquellos datos que ha generado el usuario en un fichero, para que éste pueda ser

recuperado en otra sesión. En el caso de este programa, consiste en almacenar las variables y los registros introducidos por el usuario.

2.2. Lectura de ficheros de datos creados en los formatos de texto, propio y html para salida

El manejo de ficheros es una característica común a la mayoría del software moderno. Se guarda una copia del trabajo en la memoria no volátil del ordenador[1], permitiendo recuperarla más adelante o el intercambiar información con otras personas.

Durante la elaboración del proyecto apareció la restricción de formatos de trabajo. El motivo fue el exceso de horas que suponía realizar dichas implementaciones, junto a la falta de especificaciones (Formatos de SPSS o Excel).

2.3. Permitir las funciones más usuales con los datos

Estas operaciones permiten manipulaciones más complejas de los datos que la simple inserción y extracción. Básicamente son un extracto de las operaciones de manejo de las hojas de cálculo, que son una interfaz muy frecuente en las aplicaciones de ámbito numérico.

Inserción de datos: El usuario ha de poder introducir los datos, extraídos de cualquier suceso, a través del teclado y del ratón. Se tiene que garantizar un método intuitivo para el usuario.

Selección de datos: También conocida como filtrado, esta característica consiste en permitir un criterio para descartar ciertos registros de cara a las operaciones estadísticas. Es decir, cuando se filtran unos datos se está escogiendo un subconjunto de estos.

[1] El término memoria no volátil engloba discos duros, disquetes, memorias flash y otros muchos dispositivos

Cortado de selección: Dada una selección en la interfaz de usuario, cortarla significa extraerla de los datos y mantenerla en una zona de memoria conocida como portapapeles.

Copiado de selección: Es equivalente al cortado, sólo que la información que es representada por la selección se mantiene en los datos sin ser borrada.

Pegado desde el portapapeles: Consiste en trasladar aquello que se encuentra en el portapapeles a los datos que está manejando el programa en ese momento.

2.4. Implementar las transformaciones matemáticas de los datos

El objetivo de las transformaciones matemáticas es obtener nuevos datos a partir de los ya existentes con la ayuda de algunas operaciones matemáticas.

El programa debe incluir los operadores aritméticos y lógicos y, además, debe permitir al usuario definir nuevas funciones de transformación.

2.5. Realizar un estudio descriptivo completo de la variable o variables de interés

La estadística descriptiva es la parte de la estadística que se encarga de analizar y representar los datos. Obtiene parámetros que, aunque no permiten establecer conclusiones acerca de la naturaleza u origen de dicha variable aleatoria, permiten obtener una visión más o menos amplia de su morfología.

No presuponen que los datos se ajusten a alguna distribución, por tanto, son no paramétricos. Las pruebas que vamos a implementar de la estadística descriptiva son las siguientes:

2.5.1. Medidas de centralización

Indican los valores que actúan como "centro" sobre el que se agrupa la población. Estas son las medidas a implementar:

- Media
- Mediana
- Moda

2.5.2. Medidas de posición

Estas medidas tratan sobre la división en grupos de la población. Únicamente se implementará la siguiente medida:

- Cálculo de percentil

2.5.3. Medidas de dispersión

Mensuran la dispersión de los datos repecto a los resultados de las medidas de centralización. De las medidas de dispersión serán implementadas las siguientes:

- Varianza
- Desviación estándar
- Máximo
- Mínimo
- Rango

2.5.4. Medidas de forma

El objetivo de las medidas de forma es cuantificar ciertas propiedades asociadas a la disposición de los datos. Las operaciones a implementar son:

- Curtosis
- Coeficiente de asimetría

2.5.5. Tabla de frecuencias

Es una tabla en la que se muestra la frecuencia de los valores de la variable.

2.6. Realización de los procedimientos básicos de la estadística inferencial paramétrica y no paramétrica

La estadística paramétrica y la estadística inferencial son dos ramas de la estadística con diferentes objetivos:

- La estadística paramétrica es aquella que presupone una determinada distribución para los datos.
- La estadística inferencial es aquella que permite inferir característica de la población a partir de una muestra dada.

2.6.1. Procedimientos de la estadística inferencial paramétrica

El objetivo de este tipo de procedimientos es deducir si un conjunto de datos tiene una determinada característica. En estos procedimientos se supone que los datos siguen una determinada distribución (habitualmente la distribución normal).

- Comparación de medias

2.6.2. Procedimientos de la estadística inferencial no paramétrica

A diferencia de los estudios paramétricas, en estos casos se pretende conocer una cualidad de la población sin suponerle ninguna distribución.

Tests de bondad de ajuste Las pruebas de bondad de ajuste permiten averiguar si una población se ajusta a una distribución especifica. Hay dos tipos de test de bondad de ajuste que interesa implementar:

- Test de Kolmogorov-Smirnov
- Test χ^2

2.7. Realización de estudios gráficos

Los estudios gráficos son simplificaciones que muestran una o más cualidades de los datos. Son especialmente efectivos para comprender el aspecto de los datos. Deben tener en cuentas si se trabaja con variables discretas o continuas.

2.7.1. Estudios gráficos para variables discretas

Son aquellos estudios gráficos pensados para aquellas variables de naturaleza discreta, como los enteros. Los estudios a implementar son:

Diagramas de barras Se representa en el eje de ordenadas los diferentes casos de la muestra, mientras que en el eje de abscisas se representa su valor. El valor máximo de la gráfica en el eje y coincide con el valor máximo de la muestra.

2.7.2. Estudios gráficos para variables continuas

Son los estudios gráficos destinados a variables que no son discretas, como por ejemplo las variables reales. Habitualmente, y a diferencia de los estudios para variables discretas, trabajan con intervalos. Los estudios que vamos a implementar son los siguientes:

Histogramas de frecuencia Estas gráficas consiste en representar en un eje de coordenadas la frecuencia de los diferentes valores o intervalos de valores. En el eje x aparece los valores o intervalos de valores, mientras que en el eje y aparece la frecuencia.

Capítulo 3

Análisis

Los proyectos extensos tienden a seguir modelos ya existentes en su desarrollo. Por ejemplo, en el modelo de desarrollo en cascada, existe un análisis de riesgo previo al diseño.

En el caso de *Diseño e implementación de una interfaz orientada a la docencia para el paquete estadístico R*, en el análisis se llevarán a cabo las siguientes acciones:

- Sintetizar las necesidades del proyecto
- Delimitar el alcance del proyecto
- Escoger las herramientas y los enfoques que permitan cumplir los objetivos

En las próximas secciones se analizarán los diferentes aspectos del proyecto.

3.1. Elementos internos

Definimos elementos internos como aquellas estructuras que, sin implicar relación con ninguna otra parte (ya sea usuario o programa), son necesarias para el funcionamiento del programa. En esta categoría se encuentra, principalmente, el manejo de datos, que iremos abordando en los próximos apartados.

3.1.1. Organización interna de los datos

Tener definida correctamente la forma de manejar los datos a nivel de programa es imprescindible para poder manipularlos y exportarlos a cual-

quier otro formato. Por tanto, para cubrir el objetivo de crear ficheros de datos, la correcta definición de clases y de elementos es imprescindible.

Tipos de variables

Para facilitar la labor del usuario, el programa debe permitir transformar unos datos en un formato de manipulación humana a un formato con el que pueda trabajar el motor R.

Por ejemplo, los estudios estadísticos sobre el calendario de eventos requieren por parte del usuario la transformación de cualquier fecha en un número. Permitir la inserción directa de la fecha es el objetivo del manejo de tipos de variables.

Algunos tipos útiles podrían ser, por ejemplo, además de los numéricos:

- Fecha
- Hora

Creación de variables

El usuario puede desear crear una variable cuyos valores sean función de los ya existentes. Por ejemplo, puede interesar calcular el precio final de un artículo incluyendo los impuestos. El valor de la nueva variable sería obtenido a través de la fórmula

$$preciofinal = precio + precio * impuestos \tag{3.1}$$

Esta funcionalidad es muy frecuente es los programas de hoja de cálculo, como el *Excel* o el *Oocalc*[1]

Conversión entre variables

Una vez creada una variable, puede darse el caso de que el usuario quiera cambiar el tipo de la variable. Por ejemplo, puede interesar que una variable real se convierta en una entera, para simplificar o redondear los cálculos.

Filtrado

El filtrado es una función que, para un conjunto de valores de entrada, devuelve un subconjunto de éstos. Tomando notación de conjuntos, tendríamos:

$$(a, b, c, d) \rightarrow (a, d) \tag{3.2}$$

[1] Oocalc es la hoja de cálculo de la suite OpenOffice

Es posible que para ciertos estudios estadísticos el usuario quiera excluir algunos registros. Por ejemplo, si tenemos un estudio de las notas de clase, y queremos excluir a la gente que ha repetido, podríamos emplear un filtro

La implementación consiste en usar una variable especial como filtro. Dado un registro, si la variable tiene un valor nulo, dicho registro será descartado. La no existencia de la variable especial implica la ausencia de filtro.

Etiquetado de casos

Para facilitar la comprensión de los resultados, el etiquetado de casos permite que para cada caso exista una etiqueta asociada. Este elemento facilita la interpretación de los datos, especialmente cuando existen muchos índices numéricos que agrupan la muestra.

En el ejemplo del apartado anterior, sin el etiquetado de casos, el usuario no podría saber que casos son los anteriores al 95 y cuales son posteriores.

3.1.2. Deshacer y rehacer

Toda interfaz humana, en su diseño y concepción, debería tener la posibilidad de deshacer las acciones realizadas. Es una medida que garantiza la seguridad y disminuye el alcance de los errores.

Lo idóneo sería que se pudiera deshacer cualquier acción del usuario, tanto en la introducción de datos como en el manejo de ficheros o resultados. No obstante, mientras más elementos se incluyen para deshacer y rehacer, el diseño requiere mayor complejidad.

A nivel de patrones de diseño, hay dos particularmente apropiados para esta cuestión:

Command Pattern: Las operaciones de inserción o extracción de variables o registros serán objetos de una clase que herede de otra clase base llamada "transacción". Cada operación (transacción) tendrá un método para volver al estado anterior. Por este motivo, en algunos casos almacenará ciertos datos como, por ejemplo, qué había en una posición antes de un cambio.

Una pila almacenará las transacciones, permitiendo al usuario navegar a través de la lista de acciones y recuperar un estado anterior.

Memento Pattern: Una clase "portero" almacena los estados, que serán copias independientes "memento". La clase que actúa de interfaz de datos pregunta al portero en cada momento, devolviendo éste el estado que se le haya solicitado.

Idoneidad de los patrones de diseño

Ambos modelos son aplicables para todas las acciones del programa, sin embargo, ambos no se adaptan igual de bien:

Manejo de datos: Ambos patrones funcionan correctamente; la implementación del memento es casi trivial.

Ventana de salida: Ambos patrones funcionan correctamente.

Apertura de diálogos: Command Pattern lo permite hacer de forma más natural.

3.1.3. El modelo de consultas a los datos

Suponiendo una estructura interna de los datos, para poder ser utilizados por el resto de componentes es necesario tener un mecanismo de comunicación o interfaz. Un ejemplo de lenguaje de consulta es el *SQL* (Structured Query Language). Es utilizado por la mayoría de las bases de datos relacionales. La siguiente sentencia es un ejemplo de uso del lenguaje SQL:

```
SELECT * FROM TABLE WHERE FIELD1 = 1;
```

En el caso de un desarrollo en el que el intercambio de datos se realiza entre elementos funcionales del lenguaje de programación, realizar o implementar un lenguaje de consultas de datos resulta demasiado complejo. En estos casos, es preferible utilizar funciones en las que se pasen ciertos parámetros que emulen dicha sintaxis.

Por ejemplo, si el lenguaje de implementación tiene orientación a objetos, es posible implementar clases que sean condiciones, y pasar a la función de obtención de datos instancias de esa clase.

3.1.4. Intérprete de expresiones

Para permitir la manipulación con expresiones, el programa debe permitir interpretar las expresiones introducidas por el usuario. Para ello, hay que diseñar un lenguaje que permita interpretarlo y convertirlo en la expresión en el lenguaje de implementación. Esto implica un gran esfuerzo de implementación aunque, por suerte, en los lenguajes dinámicos es posible interpretar en ejecución una sentencia cualquiera. Este método tiene el inconveniente de que la sintaxis de la expresión tiene que ser la misma que la del lenguaje, no obstante, simplifica enormemente el desarrollo.

Funciones propias

Además de las funciones aritméticas y lógicas que son parte del lenguaje de programación, resulta conveniente facilitar al programador la creación de nuevas funciones de trabajo. Estas funciones utilizarían los mecanismos de herencia y carga dinámica de código del lenguaje de implementación.

Problemas asociados a los tipos

¿Cómo sería el resultado de una expresión tipo "19/6/2005 + 5"? Existen ciertas combinaciones de tipos y operaciones que carecen de sentido. Por ello, limitando el alcance del proyecto, la interfaz impondrá las siguientes restricciones:

1. Si el resultado es de tipo x, sólo podrán participar variables de tipo x en la expresión, salvo en el caso de los reales en los que podrán actuar también el tipo entero.

2. Los operadores lógicos solo podrán ser usados con tipos lógicos.

3.2. Interacción con el usuario

Existen muchos paradigmas sobre la interacción con el usuario. En este proyecto se ha adoptado un enfoque conservador en este aspecto ya que sigue, dentro de lo que cabe, los esquemas habituales de las interfaces gráficas de usuario modernas.

3.2.1. Estructura de la vista de datos

Teniendo un conjunto de casos y un conjunto de variables como eje principal de la interfaz, parece conveniente dividir la mayor parte de la interfaz principal en dos pestañas, una de casos, y otra de variables. La alternativa de mostrarlo simultáneamente es más compleja y, probablemente, menos intuitiva.

Para no complicar la interfaz, el resto de elementos serían accedidos a través del menú principal.

3.2.2. Navegación sobre los datos

Analizaremos los aspectos de manejo de datos que puede realizar el usuario:

Navegación directa

Se debe permitir acceder a cualquier registro o variable de forma sencilla. Normalmente los entornos de widgets proporcionan esas facilidades.

Búsqueda

El usuario puede pretender buscar un valor concreto, o una fecha. Si el número de registros es muy elevado, es preferible que la interfaz lo haga por nosotros.

3.2.3. Inserción y manipulación de datos

Permitir la inserción de datos es un requisito indispensable de la interfaz. El usuario ha de poder introducir los datos con el teclado de forma coherente. También es importante que el usuario pueda manipular los datos, combinándolos o borrándolos.

Manipulación directa

En este caso, al igual que en la navegación sobre los datos, los entornos de widgets proporcionan herramientas que facilitan el manejo por parte del usuario. Por ejemplo, al seleccionar una celda permite modificarla.

Manipulación con expresiones

Esta forma de manipulación implica poder crear nuevas variables a través de una expresión introducida por el usuario de una forma más o menos guiada. En principio, suponiendo un soporte sólido de la interpretación de expresiones, el usuario debería poder escribir la sentencia sin tener demasiados conocimientos del lenguaje objetivo. Por lo tanto, es idóneo tener toda clase de elementos visuales, tales como botones y pestañas, que introduzcan el texto por el usuario.

3.2.4. Manejo de ficheros

Las operaciones que debería poder realizar cualquier usuario con los ficheros deben ser tanto de lectura como de escritura. La mayoría de las librerías gráficas ofrecen diálogos que otorgan esta utilidad.

La única característica reseñable de estos diálogos es la posibilidad de filtrar el contenido de directorios.

3.2.5. Los diálogos con expresiones

En estos diálogos se debe permitir escribir y asistir la elaboración de expresiones por parte del usuario. Esto es peligroso, puesto que existen bastantes probabilidades de que la expresión no sea correcta.

El filtrado de registros

El filtrado de registros requiere que el usuario escriba una expresión compuesta de los nombres de variables y por los diversos conectores lógicos. Al tener las expresiones interpretadas por el lenguaje de implementación, es natural tener un conjunto de operadores lógicos muy similares al de éste.

Operadores: Si lo que se pretende es primar las operaciones lógicas, se debe facilitar un listado de los comparadores y operadores lógicos más básicos:

- Comparación
  - igual
  - distinto
  - mayor que
  - menor que
  - mayor o igual que
  - menor o igual que
- Lógicos
  - AND
  - OR
  - XOR

La creación de variables

La creación de variables es una operación equivalente a una expresión en una hoja de cálculo. Partiendo de dos o más variables, y por medio de un conjunto de operadores compatibles, generamos una nueva variable. Cada registro tendrá como valor el resultado de aplicar la expresión a las variables fuente. Matemáticamente:

$$f(x, y) = z \tag{3.3}$$

Al usuario se le debe facilitar la escritura tanto de las variables como de las funciones. Estas funciones deberían ser tanto las del lenguaje de implementación, como aquellas que genere el programador a través de los mecanismos de implementación de funciones propias.

Operadores: El usuario podría usar cualquiera de los operadores lógicos y aritméticos que ofrece Python. En esta lista estarían, por ejemplo:

- Operador de suma
- Operador de resta
- Operador de multiplicación
- Operador de división
- Operador de potenciación

3.2.6. Cortado y pegado

El cortado, copiado y pegado son también características básicas de cualquier programa de manipulación de datos. Agilizan la manipulación de bloques de datos.

Estructura de copy/paste en Linux

La forma de manejar el copiado y el pegado es diferente según el sistema operativo que se utilice. Según la referencia de freedesktop[1], existe tres tipos de almacenes para copiado y pegado en las X, que es el entorno gráfico más utilizado en Linux:

CLIPBOARD: Es el que se utiliza habitualmente con Control+C y Control+V.

PRIMARY: Almacena el texto que ha sido seleccionado con el ratón.

SECONDARY: No es utilizado habitualmente.

Objetos mime e implementación

Los MIME, según la wikipedia[6]:

> MIME (Multi-Purpose Internet Mail Extensions, Extensiones de correo Internet multipropósito), son una serie de convenciones o especificaciones dirigidas a que se puedan intercambiar a través de Internet todo tipo de archivos (texto, audio, vídeo, etc.) de forma transparente para el usuario. Una parte importante del MIME está dedicada a mejorar las posibilidades de transferencia de texto en distintos idiomas y alfabetos.

Básicamente define la codificación que siguen los datos. El sistema y las aplicaciones han de lidiar con los tipos MIME, convirtiéndoles al formato que interese. Un ejemplo de tipo mime es "text/plain", que corresponde el texto plano.

MIME en la salida de datos: En el cortado y copiado, es necesario escoger un tipo MIME como formato de envió de portapapeles. La única alternativa evaluada fue el texto plano ya que otras alternativas requieren el manejo de datos a bajo nivel.

MIME en la entrada de datos: En el pegado, debe haber soporte para el tipo que producen las operaciones de cortado y copiado. Además, sería recomendable dar soporte a otros formatos, pero debido al esfuerzo que requiere, queda fuera del alcance del proyecto.

3.2.7. Visión de los resultados

Al usuario le interesa ver los resultados de las operaciones que ha pedido. El programa debe permitir una visión rápida y efectiva de las variables implicadas, el tipo de operación realizado,y los casos si proceden.

Sintetizando los requisitos, en los resultados debe aparecer la siguiente información:

- Cálculos realizados
- Variables de actuación
- Resultados numéricos
- Resultados gráficos

La salida debería tener una ventana propia. En esta ventana, irían apareciendo los diferentes resultados. A modo de ejemplo, si nos interesase mostrar el resultado como una tabla, tendríamos presentación que figura en el cuadro 3.1

Titulo de la operación			
	prueba1	prueba2	prueba3
Variable1	x	x	x
Variable2	x	x	x
Variable3	x	x	x

Cuadro 3.1: Ejemplo de tabla de salida

También puede interesar otros formatos de salida, como por ejemplo los siguientes:

- Texto
- Imágenes

Cada operación genera en su salida uno o más de los elementos citados anteriormente. Es en la definición de la operación donde debe indicarse tanto los elementos como el método para rellenarlos.

3.2.8. Deshacer y Rehacer

Un usuario puede equivocarse realizando una acción que no desea. En tal caso, el programa debe permitir deshacer sus últimos cambios, evitando la necesidad de que sea el propio usuario el que sepa como deshacer su acción. De cara al usuario, hay dos factores de importancia:

- Ser fácil de localizar (habitualmente esta en el menú de edición)
- El funcionamiento debe ser sólido ya que en caso contrario el usuario desconfiará y le obligará a guardar su trabajo con frecuencia.

3.2.9. Ayuda

Prácticamente todos los programas recientes incluyen alguna forma de ayuda en ejecución. La ayuda es, en lineas generales, un conjunto de explicaciones ubicados en lugares de acceso sencillo. Las alternativas más usuales son las siguientes:

Tooltips

Son las cajas que aparecen cuando dejamos el cursor encima de un elemento gráfico. Es particularmente efectivo en los toolbars, aunque también puede aparecer en botones convencionales. El objetivo es dar una pequeña descripción al instante, sin que moleste o influya en la acción del usuario.

Dialogo de ayuda

Es un diálogo que, de forma redactada y con ayuda de elementos visuales, explica un determinado aspecto del programa.

3.3. Interacción con el sistema

Los programas residen en la memoria de los ordenadores. Cuando están siendo ejecutados, han de interactuar con el sistema operativo y con el resto de programas. Esta integración es clave para no restringir las posibilidades del usuario.

3.3.1. Los entornos UNIX

Unix comenzó en la década de los setenta. Tiene una fuerte orientación a ficheros y una estructura modular. La interfaz gráfica está separada del sistema operativo y también tiene una estructura modular.

Interacción en el entorno gráfico

Una de las cuestiones que diferencia al mundo UNIX de otros sistemas operativos es que hay gran variedad de entornos de escritorio. Esto implica que la interfaz gráfica debería ser coherente con muchos entornos diferentes, lo que normalmente es posible. También implica que al escoger una librería gráfica se favorece la integración en un determinado escritorio.

Interacción con la linea de comandos

En Unix la comunicación de procesos se puede realizar de varias formas (sockets, pipes, ficheros). Aunque en este proyecto prime la interfaz gráfica, es posible y conveniente admitir opciones en la linea de comandos ya que se pretende facilitar la integración con el sistema operativo. Las opciones que ofrece son las siguientes:

- Mostrar la versión del programa
- Permitir activar y desactivar el diálogo de bienvenida
- Cargar directamente un fichero

3.3.2. El almacenamiento de datos

Prácticamente todos los programas de manipulación de datos tienen la posibilidad de usar ficheros como forma de almacenamiento, ya que es una facilidad imprescindible. Sin esta posibilidad, el usuario tendría que reescribir los datos o cargarlos desde algún medio cada vez que inicia el programa.

Formato de ficheros

Para poder crear ficheros de datos es necesario tener definidos los siguientes elementos:

- Un formato interno, un mecanismo para leer y escribir de forma coherente y rápida los datos en memoria.
- Un formato de fichero, un método para volcar la información del formato interno a un fichero cualquiera y que este pueda ser recuperado nuevamente por el programa.

Una vez cubierto el formato interno, corresponde detallar las descripciones de los formatos de fichero:

Texto: Es una representación de los datos escritos directamente en un fichero. No tiene reglas concretas, por tanto, los campos de los registros pueden estar separados por cualquier elemento, puede tener o no cabeceras, etc. Al no existir una estandarización, cada programa ha de lidiar con el resto de formatos.

Excel: Es el programa de hoja de cálculo de la suite Microsoft Office. Su especificación está cerrada, es decir, no hay certeza de que haya características sin documentar, o que su formato cambie y sea ilegible.

SPSS: En el SPSS hay cuatro tipos de ficheros:

- de datos (extensión .SAV)
- de salida (extensión .SPO)
- de sintaxis (extensión .SPS)
- de scripts (extensión .SBS)

En este formato de fichero se detallan tanto los registros(casos) como las variables, soportando tantos tipos como el SPSS. En R existen rutinas para la importación de ficheros .SAV, aunque no pueden garantizar la lectura de todas las versiones.

Volcado de datos Una posibilidad sencilla para los lenguajes de programación dinámicos consiste en hacer un volcado de datos desde la memoria del ordenador a un fichero. El inconveniente que tiene es que es un formato poco tolerante a los cambios, ya que cualquier cambio en la implementación convierte en incompatible los ficheros guardados anteriormente.

3.4. Interacción con R

R actua como motor de cálculo de la interfaz, por lo que es necesario definir como comunicar los dos programas y de que forma hacerlo.

3.4.1. Métodos de comunicación

Al utilizar R como motor de cálculo, resulta imprescindible tener definida la vía de comunicación por la que solicitar y recibir los cálculos. De la búsqueda de mecanismos han surgido dos alternativas:

Entrada estándar (UNIX)

Este método consiste en utilizar los mecanismos de comunicación de UNIX. R permite la entrada de fichero, con la opción en la linea de comandos

```
R --vanilla < infile
```

Esta operación también se puede realizar con *pipes*

```
echo "mean(c(1,2,3,1,2,3))" | R --vanilla
```

El inconveniente de este método es que implica aumentar la complejidad del proyecto, junto al control de errores que puede surgir durante esta comunicación

Rpy

Rpy es una librería desarrollada en Python que sirve para encapsular las llamadas a R a través de un objeto. Su instalación es sencilla, ya que la mayoría de las distribuciones de Linux lo permiten instalar fácilmente. Alguna de sus características más relevantes son las siguientes:

- Acceso a todas las funciones con leves cambios en la sintaxis.
- Integración con los tipos básicos de Python.
- Soporte de excepciones.
- Manual disponible en Internet.

3.4.2. Modelos de interacción

R es un entorno por si mismo. Tiene sus variables, sus tipos, y sus funciones para manejarlos. La interfaz puede tener mayor o menor grado de utilización de estas facilidades:

Sólo funciones: Los datos son trabajados de forma separada y, en última instancia, sólo se le pasa vectores de numéricos a las funciones de R.

Modelo Intermedio: Se puede manipular los datos tanto con las funciones de R como con las que suministra la interfaz.

Utilización total: La interfaz únicamente envía órdenes a R y muestra los datos en ese momento. Este es el modelo de funcionamiento de RCommander.

El modelo que se va a utilizar es el de sólo funciones ya que es el que da mayor flexibilidad, aunque tiene mayor coste en la implementación. Sin embargo, es de esperar que en el desarrollo se vayan incluyendo nuevos elementos de R que provoquen un reajuste del proyecto hacia el modelo intermedio.

Integración de tipos de datos

R dispone de clases para el manejo de fechas, enteros y tipos numéricos. Algunas de estas clases se encuentran en la distribución principal de R, otras en paquetes independientes. El inconveniente de la utilización de estos tipos es que impiden la adición de características de Python. El modelo a seguir es, aunque complique el desarrollo, encapsular los tipos de datos de R en clases de Python. Este elemento diferencia este desarrollo respecto a otras interfaces de R.

3.5. Realización de estudios estadísticos

La característica común a todos los estudios estadísticos es que hacen un trabajo sobre los datos, con ciertas opciones suministradas por el usuario. Sus resultados normalmente se representan en formato tabular, o bien en formato gráfico.

3.5.1. Estudios descriptivos

Los estudios descriptivos a realizar, prefijados en los objetivos, son los siguientes:

Medidas de centralización

Media: Existe varios tipos de media:

- Aritmética
- Armónica
- Geométrica
- Ponderada

La media que se solicita habitualmente como medida de centralización es la media aritmética. Su expresión matemática es la siguiente:

$$\frac{\sum_{i=0}^{n} a_i}{n} \tag{3.4}$$

En R se realiza con la función

```
mean(vector)
```

Mediana: Es otra medida de centralización que devuelve aquel valor para el que existe el mismo número de valores mayores que de valores menores. En R se realiza con la función

```
median(vector)
```

Moda: Es el valor más frecuente de una colección de valores. En R se calcula con el primer valor de una tabla de frecuencias.

Medidas de dispersión

Varianza: Es un estimador de dispersión de una población cuya fórmula matemática es

$$V(X) = E[(X - E[X])^2] \tag{3.5}$$

En R se calcula con la función

```
var(vector)
```

Desviación típica: Es otro estimador de dispersión de población. Está vinculado con la varianza, ya que la desviación típica es su raíz cuadrada. En R se calcula con la función:

```
sd(vector)
```

Máximo, Mínimo y Rango: Los elementos máximo y mínimo son el mayor y el menor elemento de un conjunto de datos respectivamente, mientras que el rango es el intervalo que delimitan estos dos valores. En R tenemos las funciones:

```
max(vector)
min(vector)
range(vector)
```

Para obtener el valor de los tres elementos

Cálculo de percentiles(cuantiles): Es aquel valor que tiene tantos elementos inferiores con respecto al total como indica la cantidad. Los más utilizados son la mediana (50) y los cuartiles (25 y 75).

Medidas de forma

Miden numéricamente alguna cualidad de forma de la muestra.

Curtosis: La curtosis es una medida de la cantidad y forma de alejamiento de la media de los valores de un conjunto. Su formula en R es:

```
require("e1071")
kurtosis(vector)
```

Coeficiente de asimetría: Medida de la simetría de la distribución de la muestra. Su formula en R es:

```
require("e1071")
skewness(vector)
```

Tabla de frecuencias

Es una tabla que muestra el número de ocurrencias de cada valor de la variable. En R se calcula con el comando:

```
table(vector)
```

3.5.2. Estudios paramétricos

Son aquellos estudios que presuponen que los datos siguen una distribución (habitualmente normal).

Comparación de medias

Una comparación de medias es una prueba de hipótesis. En este caso, se desea comprobar si dos muestras independientes tienen la misma media o no. Las hipótesis que vamos a contrastar son las siguientes:

$$H_0 : E_1 - E_2 = 0$$
$$H_1 : E_1 - E_2 \neq 0$$

Una peculiaridad de este estudio es que se puede realizar asumiendo varianzas distintas o iguales. De cara al usuario, es preferible darle ambos resultados y un estimador de igualdad de varianza. En R, para realizar una comparación de medias, tenemos que introducir las siguientes lineas:

```
t.test(vector1,vector2,var.equal=True,alt="two.sided")
```

Para realizar este cálculo se utiliza la distribución t de student, y su resultado se devuelve como significación bilateral.

3.5.3. Estudios no paramétricos

En estos estudio no se le presupone ninguna distribución particular a los datos.

Test de bondad de ajuste χ^2

El estimador de este test se puede hallar con la siguiente formula:

$$\chi^2 = \sum_{i=1}^{k} \frac{(o_i - ei)^2}{e_i} \tag{3.6}$$

o es la frecuencia obtenido, y e la esperada.

El iterador i corresponde a cada celda (cada segmento). Con el valor obtenido, se consulta con la distribución χ^2, con k-1 grados de libertad. Si el valor es cercano a 0 significa que existe un buen ajuste.

Esta distribución requiere al entorno de trabajo dividir en intervalos la muestra, y obtener la frecuencia de cada intervalo. Tiene la restricción de que no se debe utilizar si la frecuencia esperada de algún valor es menor de 5.

Una vez tenidos en cuenta todos estos inconvenientes, la forma de solicitar a R este cálculo es:

```
chisq.test(vectorfrecuenciasobtenidas,p=vectorfrecuenciasesperadas)
```

El vector de frecuencias esperadas se puede obtener, en el caso de la distribución uniforme, con el comando de r “runif”.

Test de bondad de ajuste Kolmogorov-Smirnov

Es también una prueba de bondad de ajuste. En vez de trabajar con las frecuencias de cada valor o intervalo, este test trabaja con las frecuencias acumuladas, buscando la mayor diferencia posible. Al igual que χ^2 es unilateral por la derecha.

Su utilización en R es más sencilla que la del test anterior, puesto que no es necesario dividir la muestra ni obtener el vector de frecuencias esperadas. La forma de llamar a la función en R es:

```
ks.test(vector,función)
```

donde "función" es la función de R que obtiene dicha distribución.

3.5.4. Estudios gráficos

Los estudios gráficos se caracterizan por mostrar gráficamente una cualidad de los datos para que sea evaluada visualmente por las personas.

Histograma

Consiste en agrupar los datos en intervalos del mismo tamaño y representar en el eje vertical cuantos valores hay en ese segmento. Es altamente efectivo para ver la distribución de los datos. La sintaxis en R es:

```
hist(vector,main=titulo,nclass=numerointervalos)
```

Diagrama de barras

Tiene una utilidad similar al histograma, salvo que en vez de agruparse los datos, al tener naturaleza discreta, se muestra la ocurrencia de los valores en el eje vertical. La sintaxis en R es:

```
barplot(vector,main=titulo)
```

3.6. Las herramientas de trabajo

La elección de las herramientas de trabajo no es algo que a priori tenga que ser analizado como parte del proyecto. No obstante, la correcta elección de las herramientas aumenta la productividad, lo que conlleva un mejor aprovechamiento de las horas estipuladas.

3.6.1. El lenguaje de programación

Los lenguajes de programación tienen múltiples características que lo pueden hacer más o menos convenientes para uso. Estas son algunas de las características que han sido valoradas a la hora de escoger el lenguaje de programación:

Orientación a objetos: Por ejemplo, la orientación a objetos es una característica muy saludable para poder desarrollar la metodología OMT (Object Modeling Technique). Relacionado con esto, es conveniente que el lenguaje tenga una implementación a objetos completa.

Disponibilidad: Otra característica importante del lenguaje es la disponibilidad. Es necesario que el lenguaje esté disponible en el mayor número de sistemas operativos posibles.

Documentación: Otra característica importante es la documentación. Un lenguaje que no esté bien documentado es más o menos equivalente a un lenguaje inútil.

Librerías: Otra característica importante es que el lenguaje tenga bindings de librerías gráficas. Sin estas, es prácticamente imposible realizar una interfaz gráfica.

Velocidad: Otra característica no tan importante es la velocidad. Es conveniente que el lenguaje sea aceptablemente rápido

Popularidad: Otra característica no tan importante es la popularidad del lenguaje. La popularidad crea comunidades, y las comunidades crean documentación, feedback y calidad en general. Un lenguaje sin comunidad tiene menos probabilidades de sobrevivir.

Soporte de excepciones: Las excepciones son un mecanismo que permite alterar el flujo de ejecución del programa. Al generar una excepción, se vuelve hacia atrás (backtrace) hasta encontrar una entidad que maneje dicha excepción. Es posible alterar el diseño de forma que no sean necesarias. No obstante, sin las excepciones el diseño puede requerir más clases y ser más complejo

De las características citadas, estas son las más importantes para la consecución del proyecto:

- Para poder cumplir con el objetivo de disponibilidad en varias plataformas, es necesaria la disponibilidad.

- Para permitir el desarrollo de la parte gráfica, es necesario que tenga soporte para librerías gráficas.

- Para hacer el desarrollo cercano al modelo OMT, es necesaria una buena orientación a objetos, y muy conveniente el uso de excepciones.

- Para hacer el desarrollo más simple, es conveniente que tenga buena documentación y popularidad

Alternativas disponibles

Existe gran cantidad de lenguajes que cumplen los requisitos mínimos del proyecto, de entre ellos quería destacar los siguientes:

C#: Es un lenguaje relativamente reciente. Deriva de C++ y C, y es parte de la plataforma .NET . Posee orientación a objetos.

C++: C++ es un lenguaje de programación más antiguo que C#. Combina las operaciones de alto y bajo nivel, y tiene soporte completo de orientación a objetos. Uno de los inconvenientes más destacables es que el manejo de la memoria dinámica es llevado por el programador.

Java: De sintaxis similar a C++, es parte de una plataforma con máquina virtual, al igual que C#. Es uno de los lenguajes más extendidos, llegando incluso al terreno de los móviles. Tiene un espacio muy consolidado en el mundo empresarial.

Python: Considerado un lenguaje de scripting. Fue creado con la filosofía de "todo es un objeto", y algunos de sus objetivos son facilitar la lectura y el diseño. Es un proyecto de código abierto.

Ruby: También considerado un lenguaje de scripting, es otro lenguaje de código abierto.

La decisión tomada

Teniendo en cuenta el soporte de objetos, el soporte de librerías, la documentación y el soporte de excepciones, la elección tomada fue Python. El hecho de que sea un lenguaje dinámico ahorra mucho tiempo a la hora de hacer el desarrollo.

3.6.2. Las librerías gráficas

Las librerías gráficas son un conjunto de funciones o clases que ayudan a programar interfaces gráficas. Las características que se ha tenido en cuenta en el ámbito de este proyecto son las siguientes:

Portabilidad: Poder utilizar la librería en varios sistemas.

Características: Tener todos los widgets necesarios ya implementados evita muchas horas de programación.

Documentación: Una buena documentación ahorra muchos quebraderos de cabeza. Debido en parte a mi experiencia como programador, el hecho de usar un lenguaje conocido o desconocido no es un factor de peso, puesto que la relativa cercanía entre estos, provoca que haya una cierta base común. Sin embargo, al no tener experiencia en la programación te interfaces gráficas, una librería poco documentada es un lastre para el desarrollo. Otra vez, debido a las características del software libre, evaluar la calidad de la documentación puede ser complicado.

Rendimiento: El rendimiento es clave para una buena experiencia del usuario.

Licencia: Debe ser libre en todas las plataformas, ya que en caso contrario plantearía problemas legales con el uso de resto de componentes.

Facilidad de mantenimiento: Por desgracia, en el software libre es relativamente fácil encontrar proyectos que quedan abandonados a su suerte. En el caso de las librerías, esto puede llegar a comprometer la continuidad del proyecto en el futuro. Factor directamente relacionado con la popularidad.

Programa generador de interfaces: Estos programas son los que, a partir de una interfaz gráfica, permiten diseñar otra interfaz gráfica. Cada librería gráfica no suele disponer de más de una herramienta de este tipo.

- **Glade:** Es la herramienta oficial de Gtk. Su interfaz es bastante sencilla, sin embargo tiene todas las características necesarias.
- **Gazpacho:** Es otra herramienta para Gtk. Es más completa que Glade, pero implica el uso de librerías del propio programa.
- **QtDesigner:** Es la herramienta oficial de Qt. También desarrollada por Trolltech, es la más completa de las evaluadas.

Requisitos del proyecto

Principalmente documentación, programa de diseño de interfaz y características para facilitar el desarrollo. No se puso ningún énfasis en el rendimiento de la librería. La portabilidad era uno de los objetivos iniciales, pero al ser restringido el proyecto, ésta paso a un segundo plano.

Alternativas disponibles

Qt: Son unas librerías desarrolladas por una compañía llamada Trolltech. Tiene una licencia dual, siendo GPL para proyectos libres. Están escritas en el lenguaje C++.

Gtk: La librería Gtk comenzó como toolkit para el programa Gimp. Desde ese momento, se convirtió en una de las librería gráficas más importantes, empleándose en el entorno de escritorio GNOME.

Wxwidgets: Son unas librerías escritas en C++ cuyo objetivo es la portabilidad.

Java y .Net: Son lenguajes de programación con máquina virtual que incluyen las librerías gráficas dentro de la definición del lenguaje.

La decisión tomada

En el caso de las librerías gráficas, la prioridad ha sido la buena documentación por encima del resto de características. Esto se debe a la dificultad que entraña el uso de librerías gráficas. También fue descartado Java y .Net, ya que la elección del lenguaje impedía su uso.

3.6.3. El entorno de desarrollo

El entorno de desarrollo es clave en la productividad. Escoger un entorno de desarrollo que no es familiar, o en el que faltan funcionalidades reduce drásticamente la eficiencia en el trabajo. Algunas de las alternativas evaluadas fueron las siguientes:

Netbeans y Eclipse: Son entornos de desarrollo muy pesados, con una curva de aprendizaje elevada y que están orientados principalmente a Java

Kdevelop: Es un IDE del escritorio Kde que tiene muchas funcionalidades y con menor orientación a java que las alternativas anteriores.

Vim y Emacs: Son dos editores con muchas características que ayudan a la productividad en la programación.

Leo: Es un programa que permite dividir los ficheros de edición según su significado.

En este caso, dada la dificultad de aprendizaje, la solución fue emplear aquellas soluciones que requerían menor aprendizaje, es decir, el entorno de desarrollo Vim. El resto de alternativas obligaba a una inversión en tiempo de aprendizaje mayor que el beneficio esperado en productividad.

3.6.4. El sistema de control de versiones

Los sistemas de control de versiones son programas que permiten mantener un seguimiento estricto de los cambios de los ficheros de un proyecto cualquiera. Por tanto, permiten recuperar el estado del proyecto en cualquier momento dado, eliminando el riesgo de pérdida por error de manejo o por deterioro físico del medio de trabajo.

Los modelos centralizados

Existe una sola copia de la estructura con todas las versiones en un servidor. Los clientes solamente tienen una copia de la última versión.

- CVS
- SVN

Los modelos descentralizados

En este modelo, todos los integrantes tienen una copia de todas las versiones. Existe un servidor central, que es el público, que es al que todos los clientes se conectan inicialmente. Tiene la ventaja de que se puede trabajar sin conexión al servidor central y, por lo general, tiene buenos algoritmos para combinar las diferentes versiones. Los inconvenientes son los siguientes:

- Cada cliente tiene una copia completa, puede suponer mucho espacio.
- No están tan extendidos en la red como los centralizados, por lo que es más complicado encontrar un proveedor gratuito.

Algunos sistemas de control de versiones distribuidos:

- Darcs
- Bzr

3.6.5. Herramientas de visualización

Muchas veces es necesario tener una buena visualización de la estructura del proyecto, ya que con el tiempo se hace demasiado grande. Algunos de los entornos de desarrollo (eclipse y netbeans) ya lo incluyen; no es el caso de vim, por lo que hay que examinar nuevas alternativas

Herramientas UML

UML es el lenguaje de modelado unificado. Permite realizar esquemas que facilitan la visualización del estado del proyecto, aunque en este proyecto solamente se han utilizado los diagramas de clases. Todos los clientes que han sido evaluados trabajan con la versión 1.0 de UML, ya que no hay alternativas sin coste para la versión 2.0. Los clientes evaluados son:

Argouml: Es un programa para la plataforma Java. Es bastante pesado, no tiene soporte de deshacer/rehacer, pero el consumo de memoria se mantiene más o menos constante. También cabe resaltar la buena interfaz de usuario que tiene, que es bastante intuitiva

Posseidon: Es un Argouml mejorado (y cerrado) hecho por la empresa Gentleware. Tiene todo lo que tiene Argouml, junto a deshacer/rehacer y mejor rendimiento. Tiene varias licencias, entre ellas una llamada "Community Edition" que es gratuita. El inconveniente fue que, durante el desarrollo, cambiaron el régimen de licencia, impidiendo el uso de la versión gratuita.

Umbrello: Es la única alternativa evaluada que no está escrita en Java. Las ventajas principales de Umbrello son tanto su velocidad con pocos elementos en pantalla como su eficacia importando de código. Sin embargo, teniene problemas de rendimiento ya que aumenta el consumo de procesador con muchos elementos en pantalla.

Durante el desarrollo se emplearon las tres alternativas, pero fueron abandonadas por diferentes motivos. Primero se utilizó Umbrello, pero fue descartado porque no soportaba correctamente los esquemas de gran tamaño. Después fue utilizado el Posseidon, pero debido al cambio de condiciones de licencia, tuvo que ser reemplazado. Finalmente se empleó Argouml, que no ha sido reemplazado hasta el final del desarrollo

Otras herramientas

En la búsqueda de la correcta visualización de los componentes del proyecto, durante el desarrollo han aparecido otras herramientas de visualización:

Generadores de diagramas: Son programas que evalúan el código generando diagramas. Normalmente se apoyan en graphviz, que es un lenguaje para creación de esquemas. El único generador de diagramas que se utilizó con éxito fue una implementación de Python para generar diagramas de jerarquía de clases[2].

También se empleo un generador de diagramas de ejecución. Con estas herramientas es posible apreciar la generación y la relación entre instancias de clases. El inconveniente de estos diagramas es que son demasiado extensos y detallistas como para aportar información útil al desarrollo.

3.7. Licencia

La elección de la licencia es clave para delimitar las libertades que se otorgan al usuario del código. Las alternativas evaluadas fueron las siguientes:

La familia GPL: Permite copia y distribución, obligando a que los trabajos derivados mantengan las condiciones. No es posible integrar código GPL en proyectos que no vayan a distribuir el código.

La familia BSD: No es tan restrictiva como la GPL, ya que únicamente obliga a la correcta atribución de la autoría del trabajo en cualquier proyecto derivado. Es posible incluir el código en cualquier proyecto, incluyendo uno de naturaleza cerrada.

Otras licencias (X11, mozilla, apache, etc.): Normalmente son compatibles con las licencias anteriormente citadas. La diferencia radica en ciertas clausulas, que están enumeradas y analizadas en la web `gnu.org`[3].

La licencia que se ha tomado es la GPL versión 2, principalmente por la obligación del mantenimiento del código con la misma licencia.

3.8. Paradigmas de desarrollo

Un proyecto de software relativamente grande esta dividido por etapas. Hay varias formas de dividir las etapas:

Desarrollo en cascada: En cada iteración se realizarán todas las etapas de software, es decir, diseño, análisis, pruebas, documentación y otros.

[2]Se puede encontrar en `http://www.tarind.com/depgraph.html`

Desarrollo en espiral: Es una variación del desarrollo iterativo en la que el análisis de riesgos cobra más importancia.

Desarrollo unificado: El sistema esta divido en varias etapas, en cada una de ellas se utilizan varios artefactos como, por ejemplo, casos de uso, que normalmente se apoyan en el estándar UML.

La desventaja de utilizar de forma estricta un paradigma de desarrollo es que genera mucho trabajo organizativo que, por lo general, es innecesario para un trabajo unipersonal. Por lo tanto, solo se utilizarán alguno de los artefactos del Desarrollo unificado.

3.9. Patrones de diseño

Según wikipedia [7],

> "Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias)"

Fue utilizado para resolver el problema de deshacer/rehacer (Sección)

Durante el resto del proyecto no fue utilizado ningún otro patrón específicamente, aunque en cierta forma se siguió el patrón *Flyweight*, que consiste en eliminar redundancia cuando hay gran cantidad de objetos. También, y debido a las peculiaridades de las librerías Qt, fue conveniente mirar el patrón de diseño *Decorator*. No obstante, Python no proporciona la sintaxis adecuada como para implementarlo directamente.

Capítulo 4

Diseño

Una vez concluido el análisis, abordaremos el diseño. Éste consta de muchas decisiones generales que están explicadas a lo largo de varios capítulos. La elaboración del diseño tiene como objetivo mantenerlo lo más sencillo posible ya que, por lo general, mejora la comprensión del resultado, facilita su mantenimiento e, incluso, mejora el rendimiento y la productividad.

4.1. Interfaz de usuario

El diseño de interfaz de usuario trata de, a partir de la utilidad de cada diálogo y ventana, exponer de que elementos debe estar compuesto. La interacción entre los diálogos y las ventanas será abordada en la siguiente sección, diseño de componentes.

4.1.1. Ventana principal

La ventana principal es el lugar donde el usuario va a realizar la mayoría de las acciones. Muestra dos pestañas, una para los casos y otra para las variables. En cada pestaña aparecerá una rejilla con los datos. A su vez, permite el acceso a todas las funcionalidades del programa, incluyendo diálogos, gestión de ficheros, configuración, etc.. El usuario podrá realizar estas acciones:

- Introducir, modificar y borrar registros y variables.
- Cortar, copiar y pegar registros.
- Abrir y guardar un proyecto.
- Importar datos de un fichero de texto.

- Acceder al diálogo de operaciones y a cualquiera de sus pestañas.
- Acceder al diálogo de configuración.
- Acceder al diálogo de creación de variables.
- Acceder al diálogo de filtrado.
- Visualizar el diálogo "acerca de".
- Acceder a la ayuda.

Los elementos que el usuario debe ver en la ventana principal son los siguientes:

- Tabla de datos.
- Tabla de variables.
- Los botones de acceso rápido.
- Menú con acceso a todas las acciones.
- Los registros que han sido filtrados.

Visualmente, la mayor parte del área corresponde a las tablas. El resto de los elementos son desplegables, de forma que ocupen el menor espacio posible.

4.1.2. Ventana de salida

Esta ventana muestra el resultado de las operaciones que han sido solicitadas. Es independiente a la ventana principal.

El objetivo principal de este diálogo es permitir ver al usuario el resultado de las operaciones y manejar estos resultados. Estas son todas las acciones que el usuario debe realizar:

- Navegación entre resultados.
- Abrir o guardar un fichero de salida.
- Exportar a html.
- Borrar un resultado.

El área principal debe corresponder al contenido. El resto de los elementos ocupará menos espacio:

- Menú con acceso a todas las acciones.
- Lista con los resultados solicitados.
- Contenido de los resultados solicitados.

La utilidad de la lista con los resultados solicitados es permitir seleccionarlos para poder borrarlos.

4.1.3. Diálogo de búsqueda

Este diálogo permite al usuario buscar un texto en la tabla de la ventana principal que esté activa en ese instante. Solamente permite introducir el texto a buscar y realizar la búsqueda. Si la búsqueda falla, el diálogo se comunicará con el usuario con ventanas de aviso.

Al ser un diálogo hijo de la ventana principal y realizar la búsqueda sobre el contenido de esta, uno de las cualidades que debe tener es entorpecer lo menos posible la visualización de los datos.

4.1.4. Diálogo de operaciones

En este diálogo es posible solicitar cualquiera de las operaciones disponibles. Dentro del propio diálogo, se puede escoger la operación a realizar. Por tanto, la elección del cálculo a realizar se puede hacer tanto en el menú de la ventana principal como en el propio diálogo. Para cada operación existe un área reservada para mostrar sus opciones, que está generada a partir de la definición de la operación. Resumiendo, las acciones a desempeñar por parte del usuario son las siguientes:

- Elegir la operación a realizar.
- Activar o establecer las opciones de la operación.
- Pedir los resultados de la función.

Las operaciones podrían ser elegidas de una lista. No obstante, al tener categorías, es preferible organizarlas en un árbol y permitir al usuario navegar por ese árbol. Estos serían los elementos a visualizar:

- Árbol con las opciones (organizados por categorías).
- Dentro del área de cada operación, las opciones y variables.
- Botones de aceptación y cancelación del diálogo.

4.1.5. Diálogo de filtrado

Con este diálogo el usuario puede escribir la expresión que determina que registros están filtrados. La interacción del usuario se centra en establecer esa expresión, o desactivar el filtro. A continuación enumeramos los elementos de interacción con el usuario:

- Establecer la expresión del filtro.
- Activar el filtro.
- Desactivar el filtro.

La efectividad de este diálogo depende de la facilidad que otorgue al usuario para introducir la expresión. Hay varios elementos que pueden conseguir este efecto:

- Un listado de variables para construir la expresión.
- Un listado de funciones de comparación.
- Un listado de funciones lógicas.

Todos estos elementos introducen texto en el área de escritura de expresión que, en última instancia, contendrá la expresión completa. En caso de error de la expresión, el diálogo se comunicará con el usuario a través de ventanas de aviso emergente.

4.1.6. Diálogo de bienvenida

Este diálogo se muestra al principio de la ejecución del programa. Permite al usuario acceder a las funciones principales del programa, que son las siguientes:

- Trabajar en un archivo nuevo.
- Abrir un archivo reciente.
- Abrir un archivo cualquiera.
- Ir al diálogo de salida.

El diálogo de bienvenida sólo aparecerá al inicio del programa, y puede ser desactivado.

4.1.7. Diálogo de configuración

Este diálogo permite al usuario configurar cualquier aspecto del programa. Los aspectos a configurar están divididos en categorías. La interacción con el usuario consta de los siguientes elementos:

- Elegir el apartado de configuración.
- Modificar cualquier opción.
- Guardar los cambios de configuración.

4.1.8. Diálogo de creación de variables

Permite crear una variable a partir de una expresión. En el diálogo acepta como entrada una expresión compatible con el lenguaje *Python*. Este diálogo es similar al diálogo de filtrado, únicamente se diferencia en los siguientes aspectos:

- En la lista de funciones aparecen operadores aritméticos y funciones de usuario.
- Tiene entrada para el nombre y descripción de la variable.
- Permite controlar los casos de sobreescritura y filtrado.

Al igual que en el diálogo de filtrado, la comunicación de errores se hará a través de diálogos de aviso y error.

4.1.9. Diálogo de casos

El diálogo de casos, lanzado desde la ventana principal, permite asociar de una variable una etiqueta, de forma que al visualizar los resultados sustituya el valor por la etiqueta. Su interacción con el usuario se reduce a la gestión de asociaciones, es decir, añadirlas y borrarlas.

4.2. Componentes

Los componentes son unidades funcionales de diseño. En el lenguaje de implementación, Python, se expresan como módulos. Para poder plasmar el diseño de componentes, emplearemos los diagramas de clases de UML. Los esquemas realizados no contienen todas las características que permite el estándar, por los siguientes motivos:

- No se expresan los tipos de los argumentos de funciones y de retorno, debido al tipado débil de Python.
- No se incluyen todos los elementos ni funciones privadas (no aportan claridad a los esquemas).
- No se incluyen las funciones en los módulos, sólo las clases, ya que son equivalentes a los métodos privados de las clases.

Las lineas que se han seguido para la división de los módulos han sido las siguientes:

- El ejecutable está separado del resto del código para permitir la instalación en sistemas Unix.
- Los ficheros correspondientes a los diálogos que son generados tienen directorio propio.

4.2.1. Categorías

Dada una operación cualquiera, como por ejemplo, la comparación de medias, tenemos varios atributos para definirla.

- Medias.
- Paramétrico.
- Contraste de Hipótesis.

Al existir muchas operaciones, con diferentes categorías, es necesario asistir a los diálogos que las muestran, generando un árbol y que este sea único. La labor del módulo de categorías es generar este árbol. Las clases que lo componen son las siguientes:

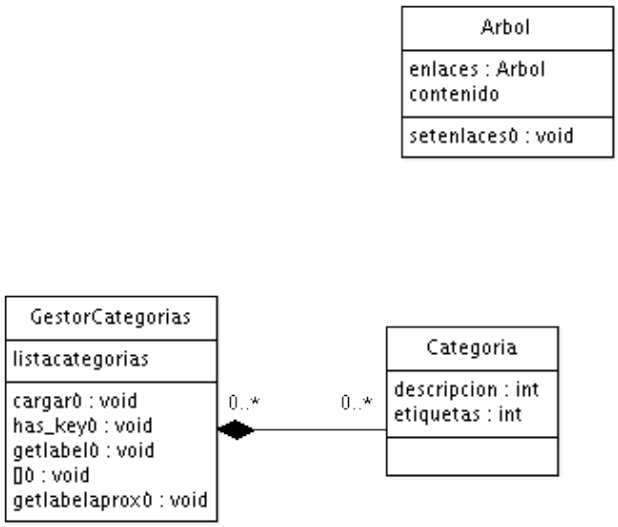


Figura 4.1: Módulo Categorías

Arbol: Implementación de un árbol genérico, en el que cada nodo contiene texto y enlaces a N subárboles. Los diferentes diálogos extraerán de una instancia de árbol la información de como rellenar sus elementos.

Categoria: Esta clase almacena la información de una categoría, que consiste en:

- Una descripción.
- Un conjunto de etiquetas que la identifican.

GestorCategorias: Clase encargada de almacenar las categorías dando funciones de acceso y actuando como interfaz para el resto de los módulos. Como índice se utilizan conjuntos, que son listas en las que no importa el orden.

Interacción con el resto de los módulos

El diálogo de operaciones accede a la única instancia de *GestorCategorias*, que contiene todas las categorías extraídas de las operaciones. La clase *Arbol* aparece tanto en el diálogo de operaciones como en la ventana principal, para generar los árboles de acceso a las operaciones.

4.2.2. Configuración

Existen determinados aspectos del programa que deben ser configurables por parte del usuario. Esta configuración debe ser única en ejecución, y debe dar la opción de ser guardada y cargada desde un fichero.

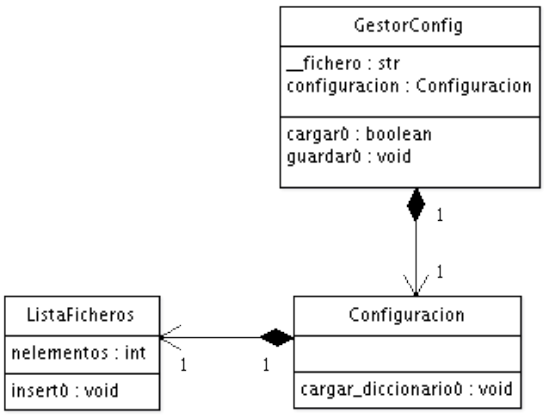


Figura 4.2: Módulo Driza.Configuracion

Las clases que componen este módulo son las siguientes:

ListaFicheros: Es una lista FIFO, que guarda la lista de ficheros abiertos recientemente. Es hija del tipo *list* del lenguaje Python.

Configuracion: La configuración en sí no es más que un diccionario de Python, es decir, un conjunto de elementos con un índice de cualquier tipo (en este caso texto). Por lo tanto, esta clase hereda sus características de *dict*. Ofrece un método que carga la información desde un diccionario genérico.

GestorConfig: Actúa como interfaz para el resto de módulos, manteniendo un enlace a la instancia de la clase *Configuracion* vigente en el programa, y ofrece las funciones de cargado y guardado en el sistema de ficheros.

Interacción con el resto de los módulos

El resto de módulos solamente tiene acceso a *GestorConfig*, preguntándole por su miembro *configuracion* cuando desean obtener alguna información. La única instancia de *GestorConfig* se crea con la clase principal *Driza*.

4.2.3. Condiciones [datos]

Las condiciones son clases de apoyo a la interfaz de datos. Tal y como fue sugerido en el análisis, era necesario imitar un lenguaje de petición de datos que permitiese el uso de condiciones. Las clases que componen este módulo

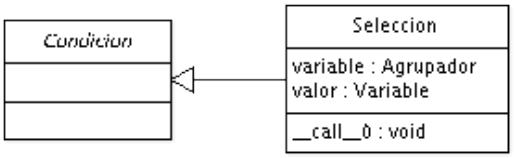


Figura 4.3: Módulo Driza.datos.condiciones

son las siguientes:

Condicion: Clase abstracta, que es padre de todas las condiciones. Su única utilidad es el control de tipos (Verificar si un objeto pasado es una condición).

Seleccion: Heredera de *Condicion*. Como miembros tiene una copia de la variable y del valor seleccionado. El método *__call__* imita el comportamiento de una función en el lenguaje Python, de forma que se puede llamar directamente como si no fuera una instancia de una clase.

Interacción con el resto de los módulos

La única interacción viene de las interfaces de datos (Driza.datos.interfaces). Este módulo crea, para la resolución de los *querys*, instancias de las clases *Condicion*.

4.2.4. Agrupadores [datos]

Los agrupadores actúan como tipo de variables en la interfaz principal, y en todo el programa. Realmente todas las instancias de la clase "variable" enlazan a un agrupador. Hay varios tipos de agrupadores que en la interfaz coinciden con los tipos de variable. Todos los agrupadores son instancias de agrupadores, no son clases hijas de éste. El motivo de esta simplificación es que lo único que cambiaba entre los tipos eran atributos estáticos de la clase padre. Los tipos de agrupadores son los siguientes:

Reales: Los números de precisión flotante. No pueden ser utilizados como discriminadores al no ser discretos.

Enteros: Números enteros, sin límite. Tiene tanto propiedades numéricas como de agrupación.

Ordinales: Equivalente a los enteros, salvo que no pueden ser utilizados como número (solo como agrupadores).

Factores: Cadenas de texto.

Lógicos Almacenan valores booleanos (Verdadero y Falso).

Cada tipo tiene definidas sus propiedades como, por ejemplo, si pueden ser utilizadas en funciones numéricas, que funciones de conversión tienen respecto al resto de tipos, o la clase que se utiliza para la representación interna.

Interacción con el resto de los módulos

Los agrupadores están íntimamente ligados a las variables y a todas las clases del módulo Driza.datos.datos, ya que junto a las variables son parte de la representación interna de datos.

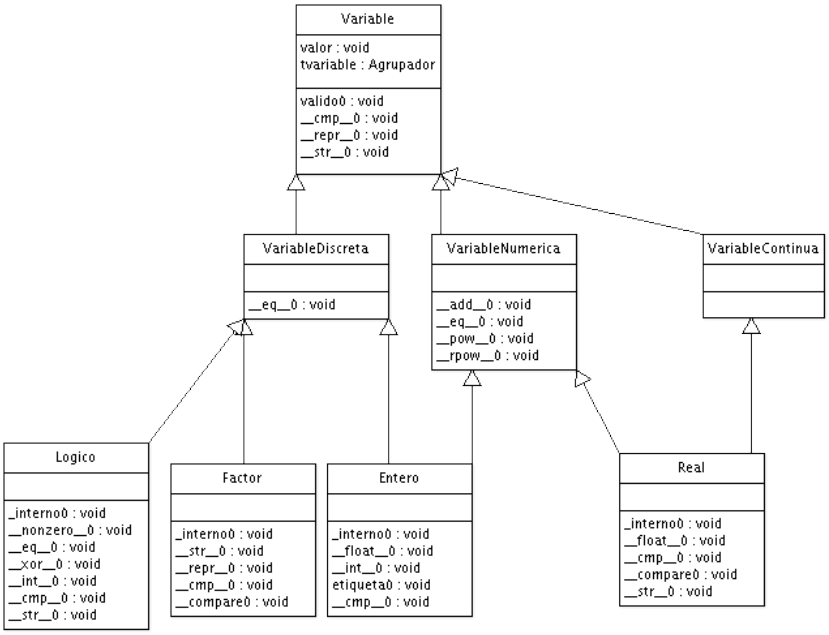


Figura 4.4: Módulo Driza.datos.variables

4.2.5. Variables [datos]

Las variables son las clases que encapsulan a los datos. Estas clases se encargan de definir la creación a partir de otros tipos y la interacción con el resto de variables. Las clases no finales que forman este módulo son las siguientes:

Clases no finales

Variable: Clase abstracta, es antecesora de todas las variables. Contiene un enlace a la instancia de *Agrupador* que contiene a la variable. También tiene funciones miembro comunes a todas las variables.

VariableDiscreta: Es hija de *Variable*, y antecesora de todas las clases que son discretas. Aporta funciones para determinar la igualdad entre elementos.

VariableContinua: Es hija de *Variable*, es antecesora de todas las clases que son continuas, sirve para control de tipos.

VariableNumerica: Clase hija de Variable, es antecesora de todas las clases numéricas. Tiene funciones aritméticas.

Clases finales Los tipos de variable implementados son los siguientes:

Entero: Variable de tipo entero, heredera de *VariableDiscreta* y *VariableNumerica*. Internamente utiliza un tipo *int* de Python. Es utilizado por los Agrupadores enteros y ordinales.

Logico: Variable de tipo lógico. Heredera de *VariableDiscreta*, solamente puede tener valor verdadero o falso. Internamente utiliza los elementos True y False del lenguaje Python. Incluye procedimientos lógicos para emular las operaciones de comparación (utilizadas en el filtrado de variables).

Real: Variable de tipo real, heredera de *VariableContinua* y de *VariableNumerica*. Internamente utiliza la clase *float*.

Factor: Variable de tipo cadena, Hereda sus atributos de *VariableDiscreta*. Se utiliza básicamente como discriminador.

Interacción con el resto de los módulos

Esta vinculado con el módulo de agrupadores, ya que cada variable enlaza con un *Agrupador*. También son los componentes de la clase *Registro*, con lo que junto a los agrupadores forman los elementos de trabajo de los datos internos.

4.2.6. Conversion [datos]

Las clases de conversión son utilizadas para la transformación de tipo de agrupadores junto a sus variables. En este diseño hay unas pocas funciones de transformación, pero la infraestructura está pensada para soportar la existencia de muchas más. Las clases que forman el módulo de conversión son

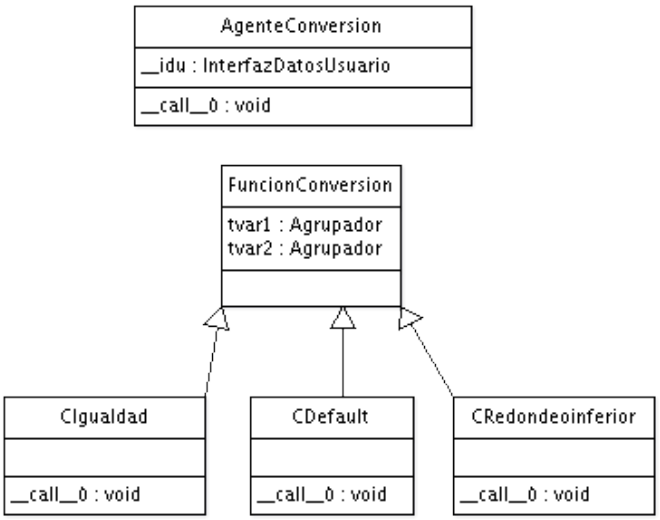


Figura 4.5: Módulo Driza.datos.conversion

las siguientes:

AgenteConversion Es la clase encargada de hacer las conversiones entre variables. Cuando es llamada a través del método *__call__* realiza la transformación, tanto del *Agrupador* como de todas sus variables, con el método indicado. Actúa como interfaz del módulo.

FuncionConversion Clase padre de todas las conversiones. Almacena los agrupadores de las variables para tener acceso a la información de estos, como por ejemplo su valor por defecto.

CIgualdad, CDefault y CRedondeoInferior Son las implementaciones de las clases de conversión. Tienen el atributo *__call__* , que permite hacer una llamada a la instancia.

Interacción con el resto de los módulos

El *AgenteConversion* es llamado desde la ventana principal para los casos en los que el usuario solicita un cambio de tipo de variable.

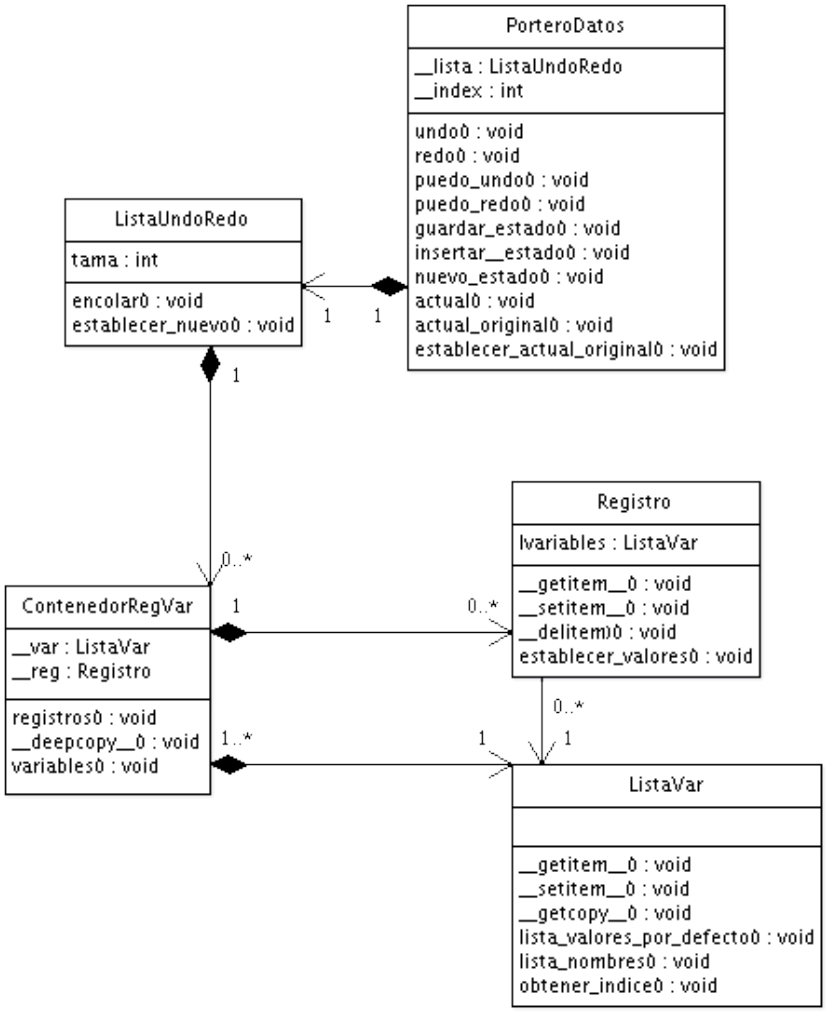


Figura 4.6: Módulo Driza.datos.datos

4.2.7. Datos [datos]

Este módulo define los elementos más generales que contienen los datos que maneja el programa. También contiene las clases que soportan la infraestructura de estados que permiten deshacer y rehacer. Las clases que forman el módulo de datos son las siguientes:

Registro: Es un diccionario modificado, ya que hereda el tipo dict de Python. Su propósito es almacenar un caso, que es un conjunto de variables. Permite el acceso por nombre de *Agrupador* además de por índice.

ListaVar: Heredera de *list*. El objetivo de esta clase es almacenar todas las variables, facilitando el acceso a ciertos elementos de su contenido, y para generar listas de nombres y valores por defecto.

ContenedorRegVar: Contiene una *ListaVar* y una lista de Registros. Es la unidad mínima manejada por el portero como estado, por lo que el programa genera copias de la instancia de trabajo continuamente.

ListaUndoRedo: Es una clase hija del tipo *list* de Python. Actúa como cola FIFO, guardando los distintos *ContenedorRegVar* que representan estados de los datos.

PorteroDatos: Es la clase que actúa como portero según el patrón de desarrollo Memento. Determina cual es el estado actual. Dispone de funciones para averiguar atributos de cada estado. Permite alterar la posición del estado actual, de forma que retrocede o avanza en la lista de estados disponibles almacenados en *ListaUndoRedo*.

Interacción con el resto de los módulos

Este módulo interactúa con la mayoría de los módulos de Driza.datos. Almacena instancias de *Variable* y *Agrupador*. También trabaja con el módulo interfaz en datos, ya que la mayoría de los accesos a los datos se hará a través de las clases de dicho módulo.

4.2.8. Funciones [datos]

Las funciones son mecanismos para la transformación de datos. A diferencia de las funciones de conversión, solamente trabajan con las instancias de variables. Las clases que componen el módulo de funciones dentro de datos

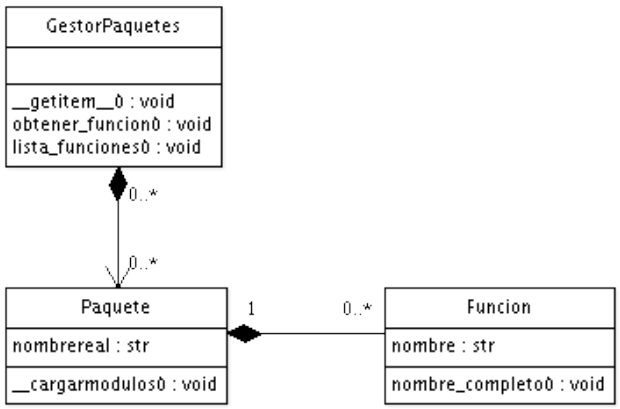


Figura 4.7: Módulo Driza.datos.funciones

son las siguientes:

Funcion: Clase padre de las funciones de transformación de la o las variables. Cuando llamamos a una función introducimos tantas Variables como se soliciten y devuelve otra *Variable*.

Paquete: Equivalente a un módulo de Python. Contiene un conjunto de funciones, y da un método para cargarlos a partir de una ruta.

GestorPaquetes: Hereda sus atributos de la clase *dict*. Gestiona dando acceso a los diferentes paquetes.

Interacción con el resto de los módulos

GestorPaquetes es consultado por el diálogo de creación de variables, ya que éste necesita saber que funciones puede escribir el usuario. Cuando el usuario escribe la expresión en el diálogo de creación de variables, la expresión es pasada a la interfaz de datos, donde las apariciones de los nombres de la función son sustituidos por instancias de las funciones.

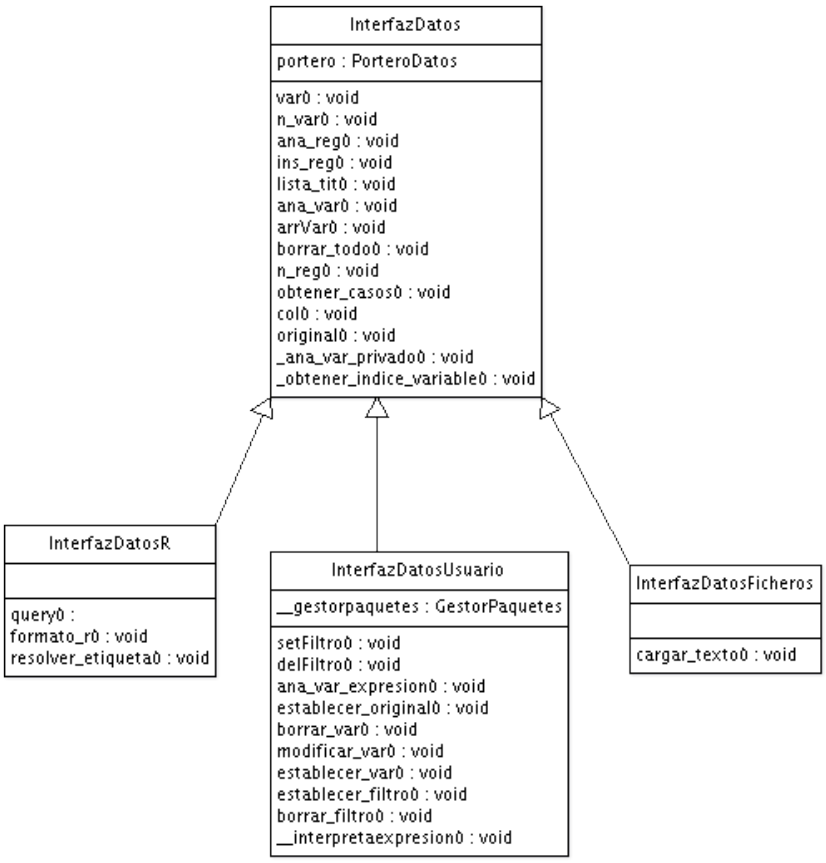


Figura 4.8: Módulo Driza.datos.interfaz

4.2.9. Interfaz [datos]

Es la interfaz del módulo de datos. Es a través de estas clases desde las cuales se accede y se modifican todos los datos. Existe una cierta especialización de clases. Esta especialización será explicada con la definición de los componentes:

InterfazDatos: Clase abstracta. Contiene todas los métodos comunes a todos los interfaces de usuario. La mayoría de los métodos son accesores, dejando a las clases hijas la responsabilidad de la modificación de los datos.

InterfazDatosUsuario: Especialización de *InterfazDatos* utilizada en la interfaz de usuario. Los métodos que posee hacen especial hincapié en la capacidad de modificación, intérprete de expresiones y el acceso al portero de datos.

InterfazDatosR: Especialización de *InterfazDatos*. La función más importante es *query*, ya que permite, a partir de los datos del *Portero*, obtener vectores numéricos sin filtros.

InterfazDatosFichero: Especialización de *InterfazDatos* orientada a la exportación e importación de ficheros. Las funciones que ofrece son de volcado a texto.

Interacción con el resto de los módulos

El miembro principal de *InterfazDatos* es *PorteroDatos*, ya que todas las funciones de las clases preguntan a éste por los datos. Fuera del módulo de datos, prácticamente todas las clases tienen una instancia de cualquiera de las tres especializaciones de *InterfazDatos*.

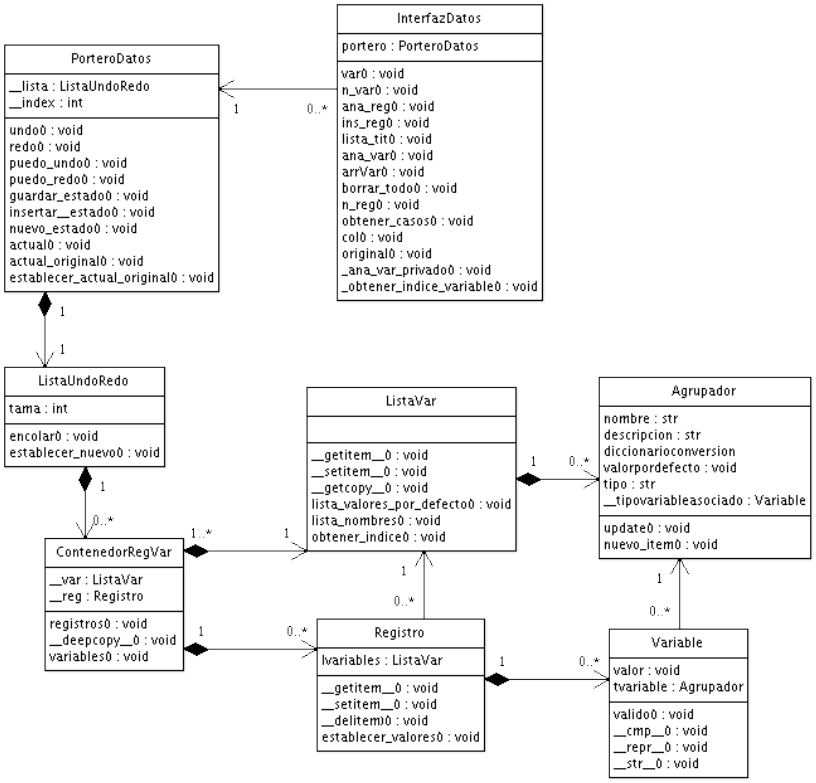


Figura 4.9: Módulo Driza.datos

4.2.10. Datos

Este módulo contiene los módulos condiciones, agrupadores, variables, conversión, datos, funciones e interfaz. No tiene clases, sin embargo, es necesario explicar ciertos aspectos de la relación entre todos estos módulos.

Interacción entre módulos

Como se puede observar en la figura 4.9, La interfaz de datos obtiene de *PorteroDatos* los elementos del *ContenedorRegVar* que necesite. Simultáneamente, *PorteroDatos* contiene una lista con los estados, que incluye las variables y los registros.

Interacción con el resto de los módulos

El resto de módulos sólo tienen acceso a las instancias de las especializaciones de *InterfazDatos*, y algunas clases también a *PorteroDatos*. El resto

de clases no son instanciadas directamente en otros módulos.

4.2.11. Gestores

En este módulo se encuentran las clases encargadas del manejo de ficheros. El módulo gestores contiene las siguientes clases:

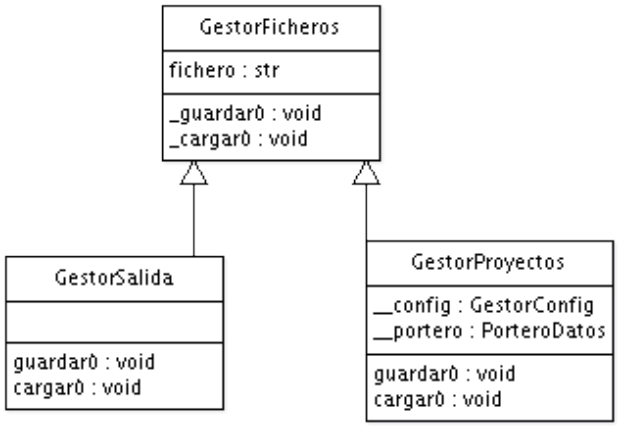


Figura 4.10: Módulo Driza.gestores

GestorFicheros: Clase abstracta. Contiene una referencia a la ruta del fichero y los procedimientos comunes al guardado y carga.

GestorSalida: Heredera de *GestorFicheros*, se encarga de gestionar los ficheros de la ventana de salida.

GestorProyectos: Se encarga de los ficheros de datos. Tiene que trabajar con *PorteroDatos* ya que cada carga significa insertar un nuevo estado.

Interacción con el resto de los módulos

El módulo GestorSalida es utilizado por la ventana de salida para gestionar el fichero de trabajo. A su vez, el módulo GestorProyectos está asociado a la ventana principal, determinando el proyecto que se encuentra abierto en cada instante.

4.2.12. Operaciones

Las operaciones son un conjunto de acciones que se realizan con los datos, acompañadas de una interfaz con el usuario en ambos sentidos, es decir, tanto de entrada como de salida. Una operación contiene la definición de los siguientes elementos:

- Definición de la interfaz de entrada de usuario, que está compuesta de:
  - Selector
  - Opciones
- Definición de los casos de excepción.
- Definición del cálculo de datos.
- Definición de la interfaz de salida de usuario.

El siguiente esquema trata sobre el paradigma que sigue la operación:

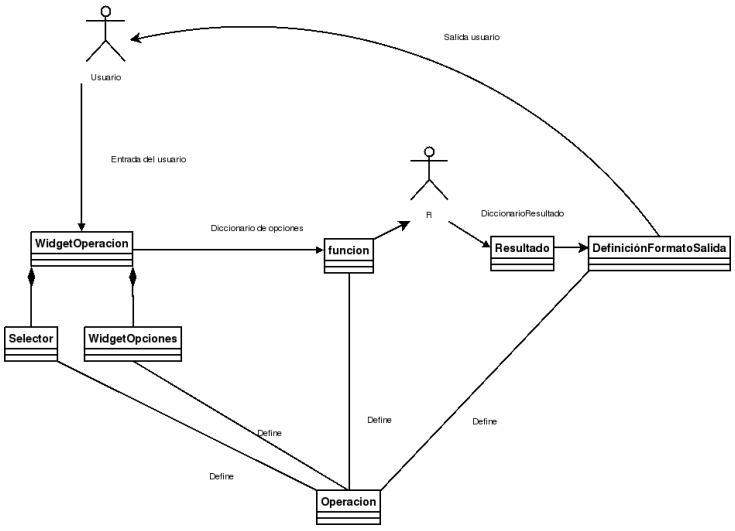


Figura 4.11: El ciclo de datos

El usuario tiene una interfaz de entrada, WidgetOperación. Ésta entrega un diccionario a la función, que solicita a R los cálculos. Los resultados que R entrega son interpretados a través de una definición del resultado, mostrando la salida al usuario.

El módulo de operaciones no incluye únicamente la clase Operación, sino que contiene gestores y especializaciones:

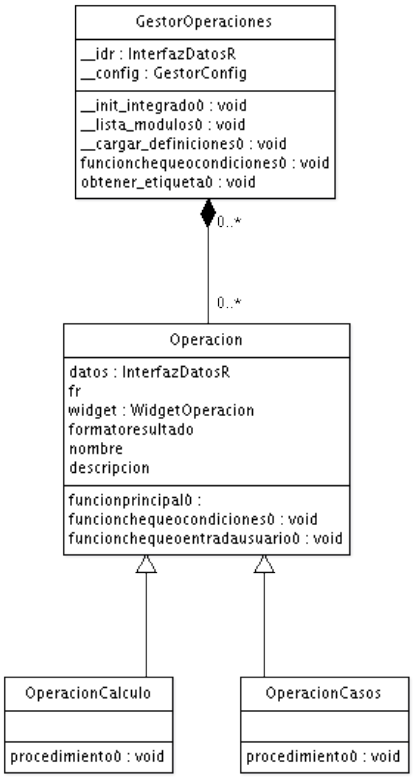


Figura 4.12: Módulo Driza.operaciones

GestorOperaciones: Gestiona todas las operaciones, cargándolas de ficheros y ofreciendo funciones accesoras.

Operacion: Clase que representa una operación. Importa su contenido de un fichero, y contiene la definición de los elementos que le componen. Es una clase abstracta, las especializaciones redefinen la forma de realizar los cálculos.

OperacionCalculo: Es una especialización de operación, para casos en los que se trabaja con una sola variable por vez.

OperacionCasos: Es una especialización de operación, para casos en los que se trabaja con una pareja variable-valor por vez.

Interacción con el resto de los módulos

Las operaciones son solicitadas por el diálogo de operaciones, ya que es esta la que se encarga de construir la interfaz y de solicitar las operaciones.

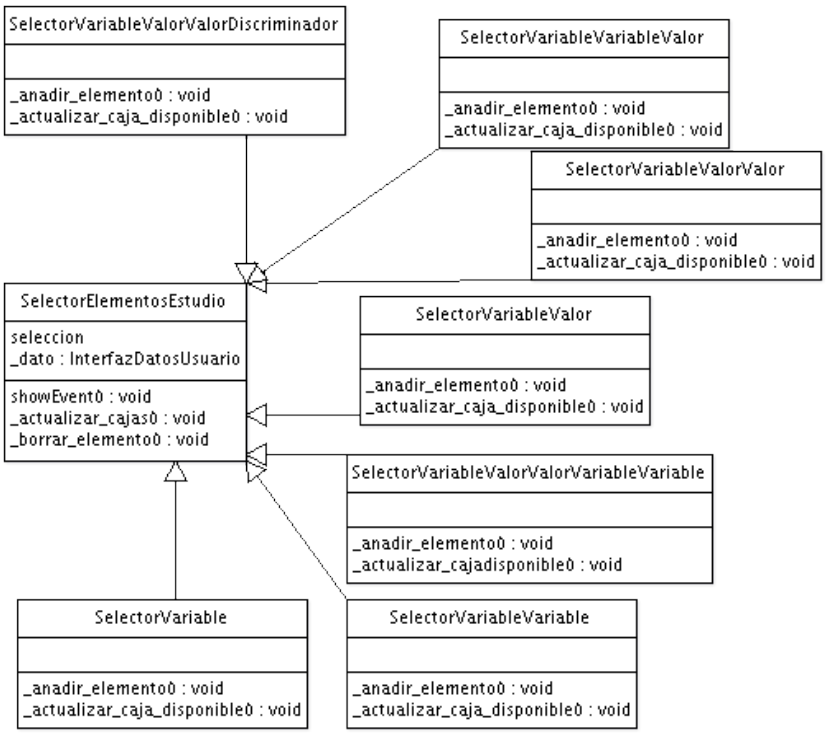


Figura 4.13: Módulo Driza.iuqt3.operaciones.seleccion

4.2.13. Selección [operaciones en interfaz de usuario]

Los selectores son los elementos gráficos que permiten al usuario elegir que variables y casos serán estudiadas en cada operación. Son integrantes de los widgets de operaciones (*WidgetOperaciones*). Hay varios tipos de selectores, todos ellos descendientes de la misma clase:

SelectorElementosEstudio: Es la clase base de todos los selectores. Es un *QWidget*, y contiene todos los elementos comunes a los selectores, como por ejemplo los botones de inserción y extracción de elemento, o la lista donde se guardan los resultados. También contiene un enlace a la interfaz de datos, ya que se ciertos selectores mostrarán la lista de variables disponibles.

Descendientes de SelectorElementosEstudio: Las clases descendientes son las implementaciones de los selectores. Lo que las diferencia funcionalmente es la cantidad y la forma de los datos a elegir. A nivel de código fuente, son los métodos de dibujado de caja los que cambian.

Variable simple: Permite escoger variables numéricas, una por vez. La disposición es trivial, ya que solamente requiere una caja con las variables que se pueden introducir.

Dos variables: Permite escoger parejas de variables numéricas. Requiere dos cajas, ambas con el listado de variables a escoger.

Un discriminador: El listado que muestra un discriminador es de variables discretas en vez de variables numéricas. Además, permite escoger un valor de dicha variable. A la hora de establecer como se debe mostrar, la variable aparecerá en una caja, mientras que la lista de valores aparecerá en un desplegable[1].

Un discriminador doble: La diferencia con el discriminador normal es que este permite escoger dos valores de la variable. Por tanto, tendrá dos desplegables en vez de uno.

Una variable y un discriminador: Consiste en la unión de un selector de variable simple y un discriminador.

Una variable y un discriminador doble: Consiste en la unión de un selector de variable simple y un discriminador doble.

Dos variables y un discriminador doble: Consiste en la unión de un selector de dos variables y un discriminador doble.

Interacción con el resto de los módulos

Los selectores son parte imprescindible en los widgets de operaciones generados dinámicamente. Es el Diálogo de operaciones el encargado de generarlos a partir de la definición de la operación.

[1] En la implementación los desplegables son llamados *ComboBox*

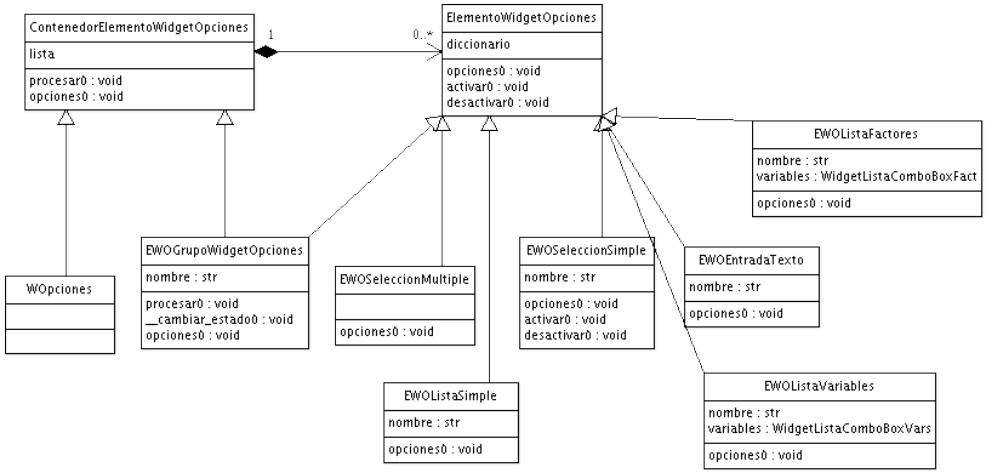


Figura 4.14: Módulo Driza.iuqt3.operaciones.wopciones

4.2.14. Widgets de opciones [operaciones en interfaz de usuario]

Los widgets de opciones aparecen en los widgets de operación junto a los selectores. Cada widget de opciones está compuesto de unos widgets, descendientes de la clase *ElementoWidgetOpciones*. Los componentes de este módulo son los siguientes:

ContenedorElementoWidgetOpciones: Esta clase es padre de todas las clases que contienen ElementoWidgetOpciones. Tiene una función para crear elementos a partir de diccionarios de definición.

WOpciones: Es la clase que hereda sus componentes de *QWidget* y que contiene diferentes *ElementosWidgetOpciones*. También es hija de *ContenedorElementoWidgetOpciones*.

ElementoWidgetOpciones: Clase padre de todos los elementos. Es abstracta, contiene todos los miembros que son reimplementados por cada una de las clases herederas.

Tipos de ElementoWidgetOpciones

Estos son las diferentes implementación de *ElementoWidgetOpciones*, junto a una pequeña explicación de su función:

Etiqueta: Muestra un mensaje de texto, sin permitir ninguna interacción con el usuario.

Entrada de texto: Muestra una caja en la que se permite al usuario introducir un texto. Este texto será el resultado enviado a la función.

Lista simple: Permite escoger una opción de una lista que se muestra en un elemento desplegable.

Lista de factores: Permite escoger una variable de tipo factor de entre todas las disponibles. La cadena de texto será el resultado enviado a la función.

Lista de variables: Permite escoger una variable cualquiera de entre todas las disponibles. La cadena de texto será el resultado enviado a la función.

Selección de un solo elemento: Se le pasa una lista de posibilidades, y permite escoger solamente una de éstas. Las muestra todas simultáneamente, con botones.

Selección de múltiples elementos: A la hora de crear este elemento, se le pasa una lista de cadenas. Al usuario se le permite escoger tantos elementos como desee, y estos se le pasarán a la función. Las muestra todas simultáneamente con botones.

Contenedor: Puede contener cualquier elemento de los anteriormente citados. A su vez, tiene un botón que permite activarlos y desactivarlos. Su utilidad principal es agrupar elementos de uso ocasional.

Interacción con el resto de los módulos

Al igual que selección, es instanciado en la creación de los widgets por parte del diálogo de operaciones.

4.2.15. Operaciones [interfaz de usuario]

En este módulo se engloba tanto el diálogo de operaciones como todos los elementos que lo forman.

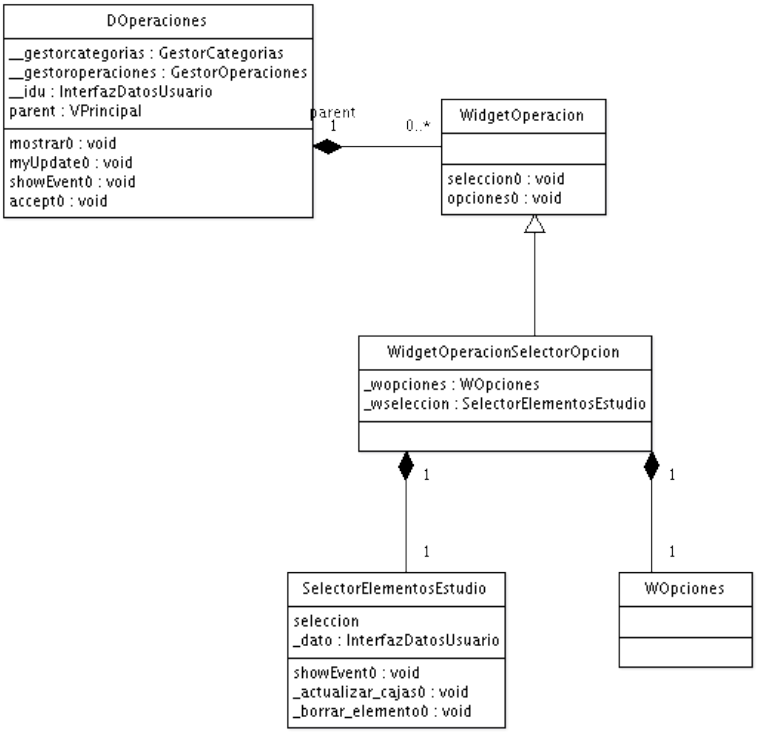


Figura 4.15: Módulo Driza.iuqt3.operaciones

Las funciones que empeñan las diferentes clases y módulos son las siguientes:

DOperaciones: En este módulo se define el diálogo de operaciones. Heredera de *QDialog*, tiene redefinidos los miembros de refresco y de aceptación.

WidgetOperacion: Clase abstracta de los diferentes tipos de *WidgetOperacion*. En el módulo woperaciones aparece junto a sus especializaciones.

WidgetOperacionSelectorOpcion: Es el único tipo de *WidgetOperacion* implementado. Incluye un *Selector* y un *WidgetOpciones* (ambos tratados en los capítulos anteriores).

Selectores y Widgets de opciones: Ambos módulos, comentados en secciones anteriores, son parte de los *WidgetOperacion*.

Interacción con el resto de los módulos

El diálogo de operaciones es creado por la clase *InterfazQt3* (en el módulo iuqt3). Además esta le pasa el gestor de categorías y el gestor de operaciones que son imprescindibles para poder obtener el listado de operaciones.

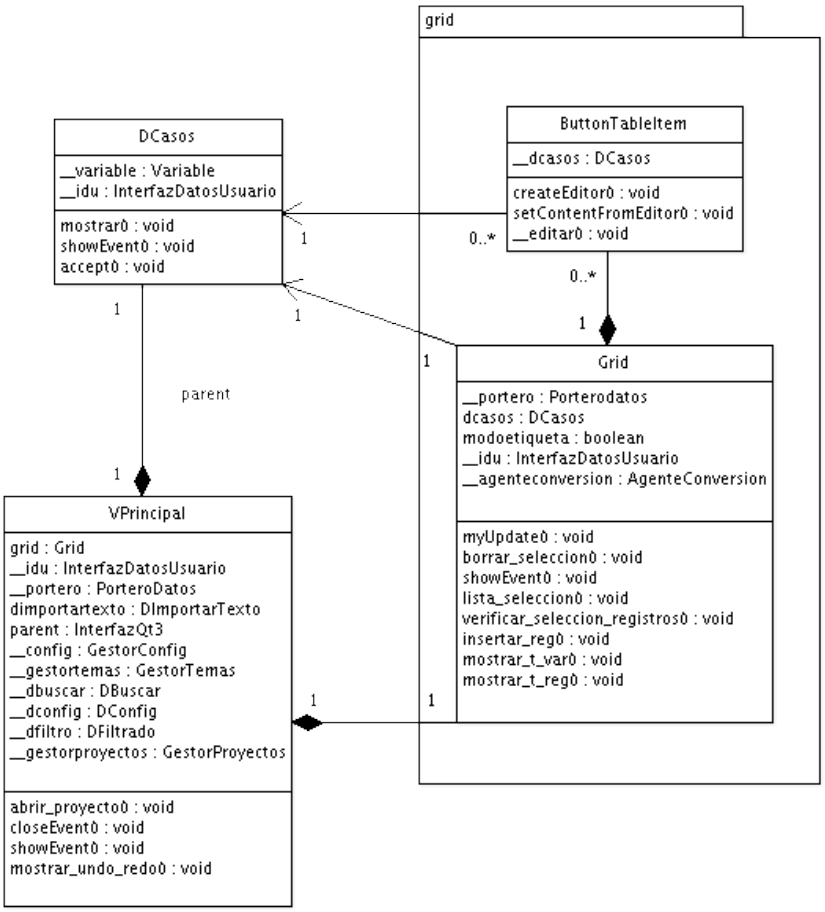


Figura 4.16: Módulo Driza.iuqt3.vprincipal

¡

4.2.16. Vprincipal [interfaz de usuario]

Este módulo contiene diferentes módulos que suministran las funcionalidades necesarias a la ventana principal. Los componentes incluidos son los siguientes:

DCasos: Es un diálogo que permite hacer asociaciones de valores con cadenas.

ButtonTableItem: Es un tipo de elemento que puebla los *QTable*, que hereda sus atributos de *QTableItem*. Aparece en la pestaña de variables de la ventana principal, y al hacer click sobre él muestra el Diálogo de Casos.

Grid: Es la clase que gestiona la zona donde se introducen los datos de la ventana principal. Tiene enlaces con los siguientes elementos:

- Interfaz de datos de usuario.
- Portero de datos.
- Agente de conversión.

Posee muchas funciones miembros, tanto públicas como privadas, para gestionar los eventos generados por el usuario, como introducir variables o registros, o el cortado y pegado. A nivel de funcionamiento interno, contiene dos *QTable*, que son accesibles a través de la pestaña de variables y datos.

VPrincipal: Hija de *QMainDialog*, es padre de prácticamente todos los diálogos. Además, tiene un menú con acceso a la mayoría de las funcionalidades, una toolbar, una instancia de *Grid* y muchas funciones miembros para gestionar los diferentes eventos.

Interacción con el resto de los módulos

Salvo la ventana de salida, todos los módulos de la interfaz de usuario están vinculados con el módulo de la ventana principal, ya que es esta la que muestra los diferentes diálogos, y es el centro de atención del usuario. Los datos son modificados por el usuario a través de la ventana principal.

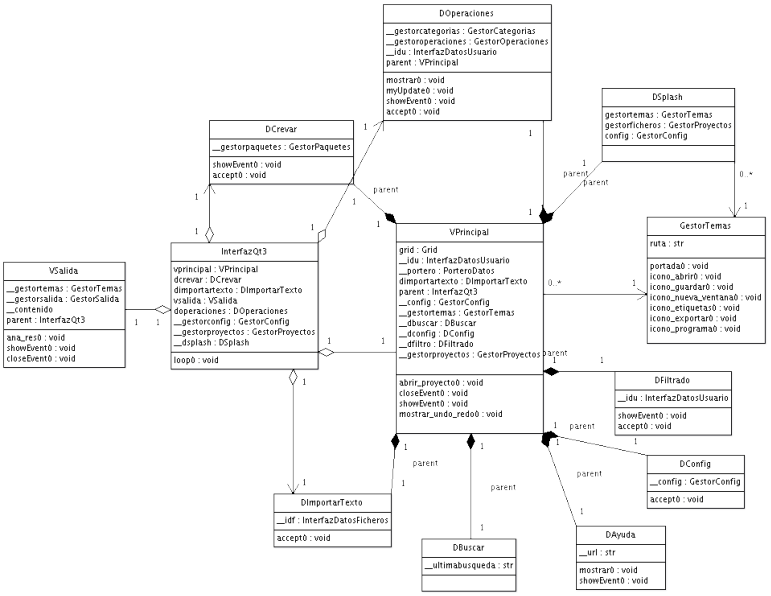


Figura 4.17: Módulo Driza.iuqt3

4.2.17. Interfaz de usuario

En este módulo se agrupa la funcionalidad de la interfaz de usuario basada en Qt3. En la figura 4.17 se puede observar la relación de los tres componentes principales de este módulo (*InterfazQt3*, *VPrincipal* y *VSalida*) con el resto de componentes.

Inspeccionando la relación entre clases y módulos

Uno de los aspectos más relevantes para comprender la relación entre clases es el orden de instanciación de las diferentes clases:

1. La clase *Driza* crea *InterfazQt3*
2. *InterfazQt3* crea la ventana principal y la ventana de salida
3. La ventana principal crea todos los diálogos
4. El diálogo de operación carga las operaciones generando los widgets que mostrará al usuario

Por lo general, es la ventana principal la que actúa como padre de los diálogos, de forma que todos los diálogos cuando terminan devuelven al usuario a la ventana principal.

Interacción con el resto de los módulos

InterfazQt3 es instanciado por la clase *Driza*, y por ninguna otra. Ello se debe a que, para separar la interfaz de usuario completamente y evitar referencias a módulos de Qt en otros módulos, se optó porque fuese el módulo de usuario el que consultase al resto de funciones del programa según la demanda del usuario.

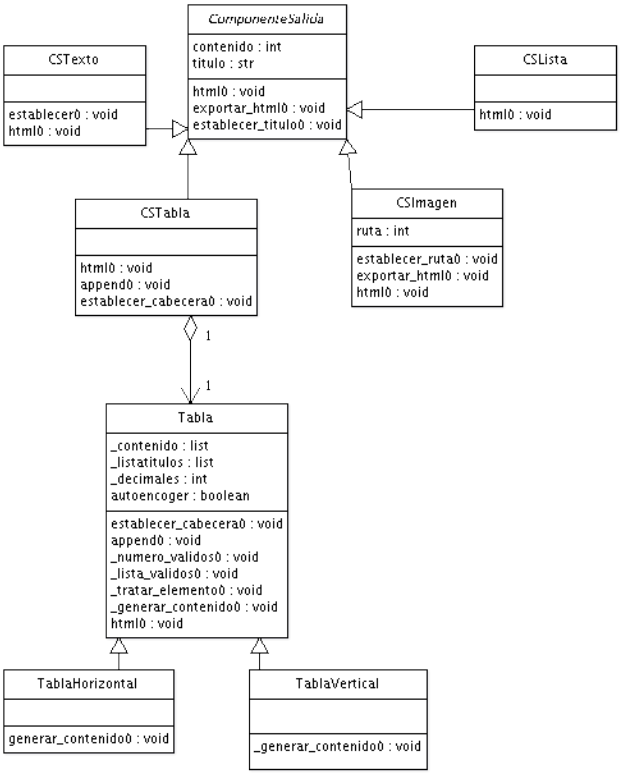


Figura 4.18: Módulo Driza.salida.componentesalida

4.2.18. Componentes de salida [salida]

Los componentes de salida son clases que encapsulan información y que tienen métodos de exportación a html. Estas son las clases del módulo de componentes de salida:

ComponenteSalida: Es una clase abstracta. Contiene todos los miembros que ha de tener un componente de salida, como las funciones que devuelven texto en formato html o exportan el contenido a un fichero.

Tabla: Clase que representa una tabla en html. Va almacenando los elementos que se le insertan, verificando que todos tienen el mismo tamaño. Las clases descendientes implementan las funciones de renderizado de html.

TablaHorizontal: Implementación de tabla que muestra su contenido de forma horizontal.

TablaVertical: Implementación de tabla que muestra su contenido de forma vertical.

Implementaciones de Componentesalida: Estos son los componentes de salida implementados:

CSTexto: Muestra un párrafo de texto.

CSTabla: Muestra una tabla. Utiliza alguna de las implementaciones de *Tabla* para manejar los datos. La implementación se decide con un parámetro en el constructor. También da la opción de que el tamaño de la tabla se ajuste al número de columnas existentes que tienen datos.

CSLista: Es un elemento que puede contener cualquiera de los Componentes de salida.

CSImagen: Guarda la ruta de una imagen, mostrándola en pantalla. Además, el método de exportación de html tiene la capacidad de manejar el fichero con el contenido de la imagen.

Interacción con el resto de los módulos

Los componentes de salida aparecen en la ventana de salida, y son generados por la definiciones de resultado que serán explicadas en el próximo apartado.

4.2.19. Componentes de resultado [salida]

Los componentes de resultados son definiciones del formato de salida. Estas definiciones se obtienen a partir de la definición de operación. Las

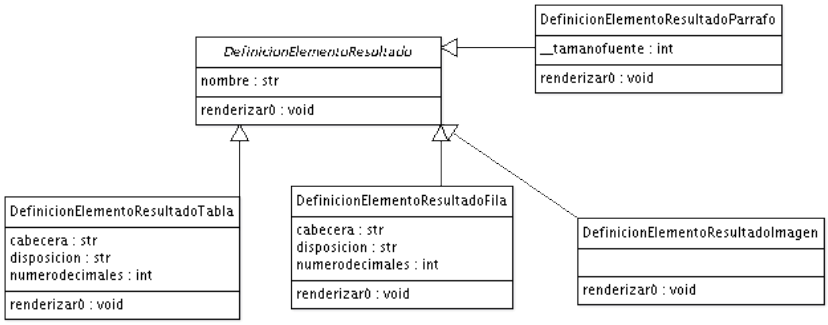


Figura 4.19: Módulo Driza.salida.componenteresultado

clases que forman el módulo de componentes de resultado son las siguientes:

DefinicionElementoResultado: Es la clase base del resto de elementos, siendo además abstracta. Tiene un método a implementar por el resto de clases, renderizar. Este método recibe un diccionario con la información que debe portar la salida.

Implementaciones de DefinicionElementoResultado: Las implementaciones de las definiciones responden a las necesidades de las operaciones. Son las siguientes:

DefinicionElementoResultadoTabla: Cada vez que la operación genere un resultado, esta definición generará una tabla en la salida.

DefinicionElementoResultadoFila: A diferencia de la tabla, cada vez que la operación genere un resultado, este será añadido como fila a una misma tabla.

DefinicionElementoResultadoParrafo: Muestra un párrafo de texto, al que se le puede modificar el tamaño de la fuente.

DefinicionElementoResultadoImagen: Genera una imagen en el resultado.

Interacción con el resto de los módulos

Las definiciones de resultados están relacionadas con los componentes de salida, ya que cada definición de resultado genera uno o más componentes de salida.

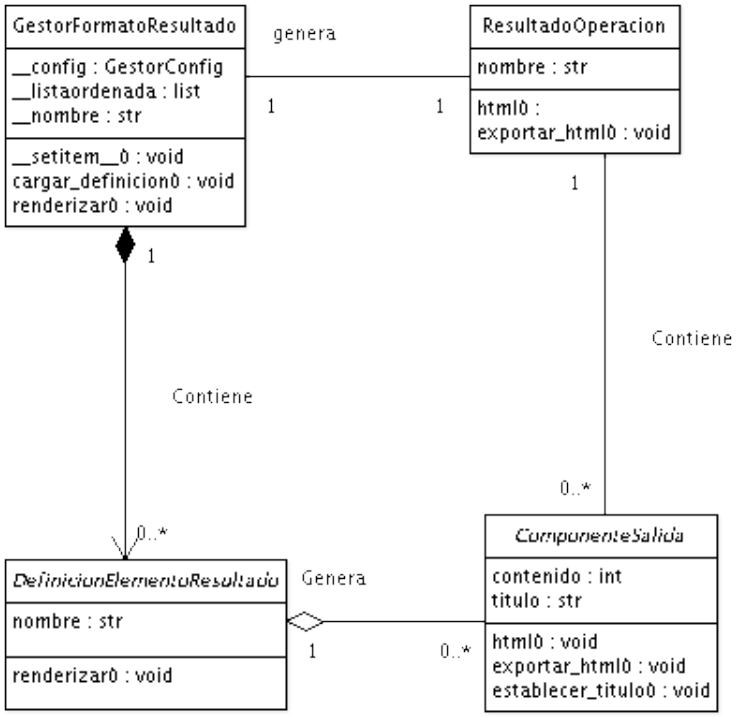


Figura 4.20: Módulo Driza.salida

4.2.20. Salida

Este módulo tiene como objetivo generar una salida de sencilla lectura para el usuario, a partir del resultado de cualquier operación. La clase principal, que actúa de interfaz del módulo, es el *GestorFormatoResultado*, que se encarga de generar resultados de operación. Cada *GestorFormatoResultado* contiene varias *DefinicionElementoResultado* que, a su vez, generan los componentes de salida. La clase *ResultadoOperacion* actúa como contenedor independiente, facilitando el manejo de operaciones en la ventana de salida.

Interacción con el resto de los módulos

La salida es solicitada por la interfaz de usuario, normalmente después de que el usuario realice una operación.

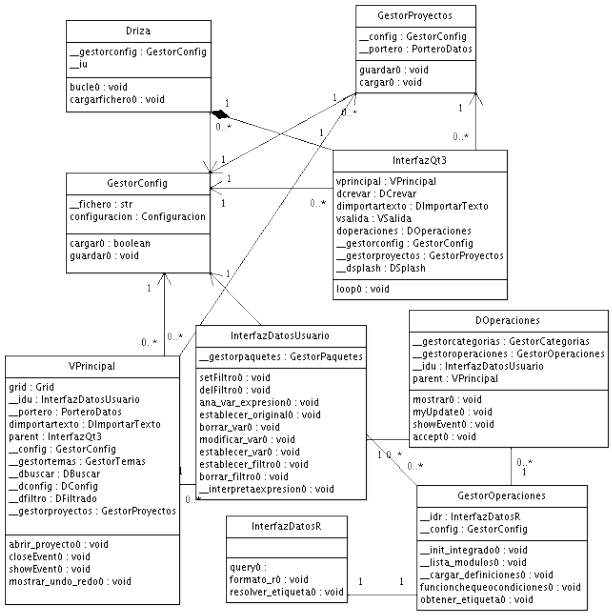


Figura 4.21: Módulo Driza

4.2.21. Driza

Sin duda, es el módulo principal del programa. La clase *Driza* es la que instancia el ejecutable, y es la que está vinculada a la ejecución del programa. Carga todos los componentes, es decir:

- Gestores
- Configuración
- Interfaz de usuario
  - Operaciones
  - Salida
- Datos

que a su vez lanzan a sus componentes. Todo ello ha sido explicado en los apartados anteriores, por lo que omitiremos repetir su descripción.

En este esquema, muy simplificado, se pretende dar una visión general del programa. Por ejemplo, el gestor de configuración esta enlazado con la mayoría de componentes, ya que es un elemento común del que se debe compartir una misma instancia.

Otro elemento reseñable es que varios elementos de la interfaz de usuario son bastante centrales en el esquema. Esto se debe a que la ventana principal tiene relación con una fracción importante del resto de módulos, y que el diálogo de operaciones es el que trabaja con el módulo de operaciones y con el módulo de salida.

Capítulo 5

Desarrollo

El desarrollo de *Diseño e implementación de una interfaz orientada a la docencia para el paquete estadístico R* ha tenido una larga duración. Ha habido varias etapas perfectamente diferenciadas, que intentaré detallar en este capítulo.

5.1. Historia del diseño

En esta sección abordaremos las diferentes etapas que pasó el diseño del programa. Debe tenerse en cuenta que lo expuesto en esta sección es complementario a la sección "Historia del desarrollo".

La complejidad del diseño de un software es proporcional a la complejidad de éste. En el caso de Driza, por cuantía de objetivos, existe una complejidad considerable. Algunas de las dificultades con las que ha contado el diseño han sido las siguientes:

- Es muy complicado plasmar un diseño extenso en un medio no visual.
- Describir todos los aspectos de un programa es imposible, ya que la explicación tendría que ser mas extensa que el código fuente.
- Existen referencias continuas dentro de la explicación, que la vuelven larga rebuscada e incomprensible.

La tendencia en el mundo de la programación es considerar que los métodos óptimos son aquellos que son capaces de expresar las relaciones de forma visual, y que explican las características importantes de forma concisa. A continuación detallaremos las diferentes etapas del diseño.

5.1.1. Un diseño desasistido

En los inicios del proyecto se intento evitar el uso del papel y, en consecuencia, evitó disponer de una forma visual para plasmar el diseño. Esto provocó que no existiese un verdadero diseño, ya que las decisiones que lo configuraban se iban tomando en función del estado del código.

Facilidad para los errores

A la hora de crear nuevas clases, estas tenían muchas veces muchas referencias innecesarias a otras clases. También aparecían atributos que referenciaban a otros atributos. En general, los roles de las clases no estaban bien definidos.

Falta de escalabilidad

A la hora de realizar cualquier cambio, suponía un tremendo esfuerzo para el programador decidir el sentido en el que hacerlo. Una vez tomada la decisión (que no siempre era la más acertada), la implementación de dicha transformación tenía una dificultad de difícil cálculo, ya que las referencias no estaban formalizadas en ningún documento, y las funciones de las clases cambiaban continuamente.

5.1.2. Diseño con árbol

Sin un diseño claro, era imprescindible que el diseño fuese germinando junto con el desarrollo. Era una necesidad conseguir mejorarlo. En consecuencia, para poder tomar mejores decisiones, comenzó el uso de los "mindmaps" (Mapas mentales).

Aportación al desarrollo

Los mapas mentales aportaron mucha claridad a la estructuración de ficheros. Al tener representado en un árbol que clases pertenecían a cada fichero, era posible tomar decisiones acerca de la distribución de ficheros y directorios. Como beneficio adicional, y de forma indirecta, fue forzando la aparición de directorios y ficheros con utilidad funcional, lo cual mejoró el mantenimiento.

Aspectos mejorables

La forma de utilización de los mapas mentales no hace referencia al contenido de clases e interacción entre estas, por lo que buena parte del diseño

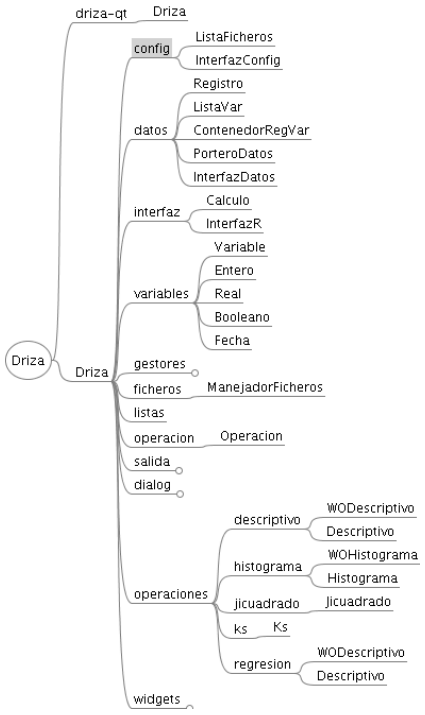


Figura 5.1: Ejemplo de esquema de organización realizado con la herramienta FreeMind

quedaba sin cubrir. Además, en cierta manera forzaba a que hubiese cambios estructurales con cierta frecuencia, alterando todas las referencias entre ficheros, y aumentando la cantidad de trabajo.

5.1.3. Diseño con esquema UML centralizado

Con un diseño por mapa mental, la escalabilidad se hizo difícil. La adición de nuevas características seguían teniendo un coste muy elevado y cierta impredictibilidad. Era imprescindible mejorarlo, así que realice un único esquema UML con todas las clases.

Aportación al diseño

Los esquemas UML permitieron ver el contenido de los miembros de las clases, y dar una idea de los enlaces con el resto de clases. Durante la conversión a esquema UML, fueron descubiertos muchos errores anteriormente ignorados. Por ejemplo, había muchas referencias innecesarias entre módulos

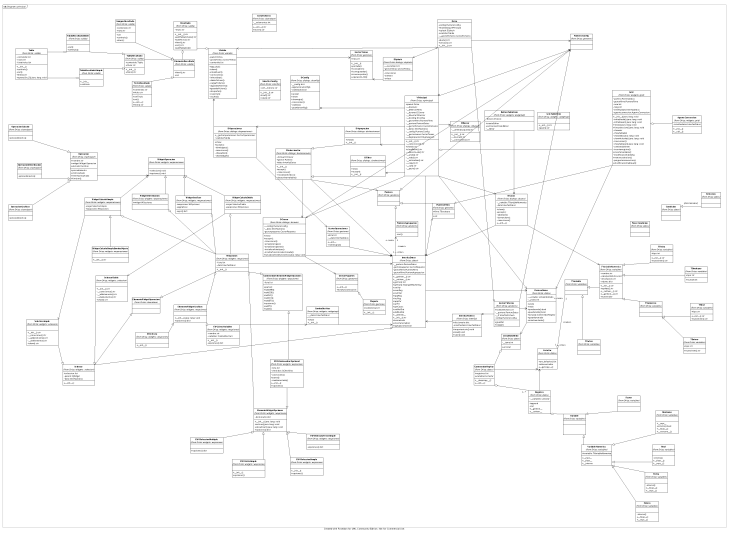


Figura 5.2: Ejemplo de esquema de organización centralizado realizado con la herramienta Posseidon

que dificultaban la independencia de estos. Dio pie a la especialización en muchas áreas.

Inconvenientes de los esquemas centralizados

El principal obstáculo para el uso de estos esquemas es que mostrar en un espacio bidimensional tantas relaciones se volvía un poco confuso, obligando en parte a separar el esquema en secciones, y haciendo perder visibilidad general, que era precisamente el objetivo principal. Además, los clientes de UML que se manejaron no están pensados para mostrar simultáneamente muchas clases, con lo que el rendimiento disminuía considerablemente a medida que aumentaba el tamaño del esquema.

5.1.4. Diseño con esquema UML descentralizado

Finalmente, y como adaptación, se fueron separando los esquemas UML imitando la estructura modular del programa, diseñando cada parte por separado (respetando la filosofía de orientación a objetos). Aparecía también un conjunto de esquemas que relacionaban a los módulos, claves para entender el funcionamiento general del programa.

Ventajas de la descentralización

Este método se ha mostrado como el más eficaz para tomar decisiones y tener una representación clara y sencilla de mantener. También aporta al diseño una tendencia a identificar en cada módulo las clases que actúan de interfaces, ocultando el resto.

Los esquemas utilizados con este método han sido los utilizados en la sección de componentes.

5.2. Historia del desarrollo

El desarrollo de este proyecto consta de varias etapas, que abordaremos en los próximos apartados.

5.2.1. Análisis inicial

Durante esta etapa, fueron analizadas las diferentes opciones. Las fuentes de consulta eran variadas, ya fueran desarrolladores, fuentes de Internet, como algunos libros. Las decisiones iniciales se tomaron a partir de la información inicial. De éstas destaca la decisión del lenguaje de programación, Ruby, y de librerías gráficas: Gtk.

5.2.2. Desarrollo en Ruby

Una vez tomada la decisión del lenguaje de programación pudo darse comienzo al desarrollo. Debe tenerse en cuenta que aunque el desarrollador tenía experiencia en Ruby, era prácticamente la primera vez que trabajaba con una librería gráfica

El aprendizaje Gtk

El aprendizaje de una librería gráfica lleva tiempo, ya que estas utilizan muchas clases y convenciones. Fueron utilizados diversos tutoriales de Ruby+Gtk en Internet, que conllevaron el aprendizaje sobre como realizar el programa completo.

5.2.3. Una mal análisis conlleva malas elecciones

Durante el aprendizaje, fue necesario comprender el funcionamiento de los *layouts*, saber como manejaba Gtk las señales y eventos, colocar un menú y otras muchas cuestiones.

El aprendizaje avanzaba a buen ritmo, considerando las limitaciones de la documentación. Pero hubo un problema de difícil solución. La librería Gtk no soporta las "rejillas" de datos. Esta limitación no apareció en el análisis, puesto que existen varios programas que la implementan (Gnumeric). Ante esta contingencia, un nuevo análisis tendría que encontrar alguna alternativa.

Obtener el código de otro proyecto

Aprovechando las virtudes del software libre, la solución más sencilla era reutilizar código. Encontré dos alternativas

PyGrid: Es un wrapper para Python de GtkGrid, una implementación de grid para Gtk en el lenguaje C. No está disponible para Ruby, así que no podía ser utilizado. Tampoco se disponía de los conocimientos para la realización de un wrapper para Python.

Gnumeric: Es un programa en C de hojas de cálculo. La evaluación del código reveló que era bastante complejo, y que la rejilla tenía muchas funcionalidades integradas que requerían trabajo a bajo nivel en C, que no son triviales en la implementación en un lenguaje de más alto nivel.

Crear una nueva rejilla

Una posibilidad era realizar el widget con las herramientas básicas de Gtk. Esta solución requería un gran conocimiento de la librería gráfica, no sólo de uso, sino de funcionamiento interno.

Parada del proyecto y análisis

La creación de la rejilla fue descartada ya que estaba volviendo el código fuente inmanejable. Al superar las 600 líneas, y ver que no estaba ni siquiera obteniendo las funcionalidades básicas, fue necesario parar el desarrollo para establecer una nueva línea de trabajo

El siguiente paso fue preguntar a otros desarrolladores que tuviesen experiencia en programación de librerías gráficas, y recabar testimonios que encontré por la red. Ante la ausencia de una solución poco drástica, me vi obligado a cambiar la librería y el lenguaje de programación. Tendría que recomenzar el proyecto en un lenguaje que no conocía: Python.

5.2.4. Vuelta al desarrollo

Una vez terminado el nuevo análisis, el panorama del desarrollo era el siguiente:

- El código fuente estaba escrito en otro lenguaje.
- La librería gráfica iba a ser cambiada
- Buena parte del código estaba destinado a una característica que ya estaba implementada en la librería gráfica.

Aprendizaje del lenguaje Python

Uno de los factores que había tenido en cuenta para la toma de la decisión del lenguaje de programación era la experiencia. Abordar el uso de un lenguaje nuevo es muchas veces complicado, resta productividad y además se corre el riesgo de tener malos hábitos de programación en ese lenguaje. Esto obligó a la lectura de abundante documentación, destacando Dive into python[2].

Aprendizaje de la librería Qt

También fue necesario aprender el uso de una nueva librería gráfica. En este caso, la documentación que se puede encontrar por Internet es de bastante calidad, y la estructura de la librería es más simple que en el caso de Gtk.

Uso de subversion

Aprovechando el reinicio en la implementación del código, comenzó el uso de subversión de forma local, para tener un control adecuado de los cambios del proyecto.

5.2.5. Desarrollo en Python

Fue en esta etapa en la que se planteó la publicación del proyecto en Internet. Cuando la publicación fue realizada, ya había transcurrido algún tiempo desde el cambio de lenguaje de programación. El proyecto avanzaba en características, y durante este tiempo ocurrió buena parte de los avances en el diseño.

La forja

El proyecto paso a estar publicado en la forja. A partir de la publicación fue necesario realizar los siguientes cambios

- Crear una página web.
- Acondicionar los textos de licencia.
- Crear un método de instalación.
- Hacer énfasis en la documentación y autodocumentación.

Con la inclusión de la forja, el control de versiones del proyecto paso a ser externo. Esto significa el acceso de cualquier persona al código, tanto a su resultado como a su evolución.

5.2.6. Finalizando el desarrollo

Después de llevar un tiempo en la forja, el desarrollo fue convergiendo a los objetivos. Los métodos de diseño, y el uso de herramientas fue perfeccionándose.

Corrección de errores

Para mejorar la calidad del código, y hacerlo mas estándar, utilice una herramienta llamada Pylint. Esta herramienta realiza una serie de comprobaciones, detectando fallos sintácticos o de estructura.

Traducción de elementos del código

En esta etapa también hubo una traducción de ciertos elementos ya que algunos, por costumbre, estaban escritos en ingles.

Separación del proyecto en ramas

En el repositorio del subversion aparece una rama de entrega del proyecto, y otra de desarrollo. Esto se debe a que existen características experimentales, como la interfaz Qt4, que no son estables todavía, por lo que no son incluidas como código del proyecto.

5.2.7. Contribuciones

Debido a la licencia del software, y a la distribución del código fuente, ha sido posible que personas colaboren realizando partes funcionales del proyecto.

Carlos Mestre González realizó la escritura y lectura del fichero de configuración con la librería *ConfigParser*.

Luis de Bethencourt Guimerá se encargó de empaquetar el proyecto para la distribución Ubuntu.

5.3. Estudio gráfico del desarrollo

En el estudio gráfico del desarrollo pretendemos mostrar algunos aspectos del trabajo de forma visual.

5.3.1. Número de líneas de código

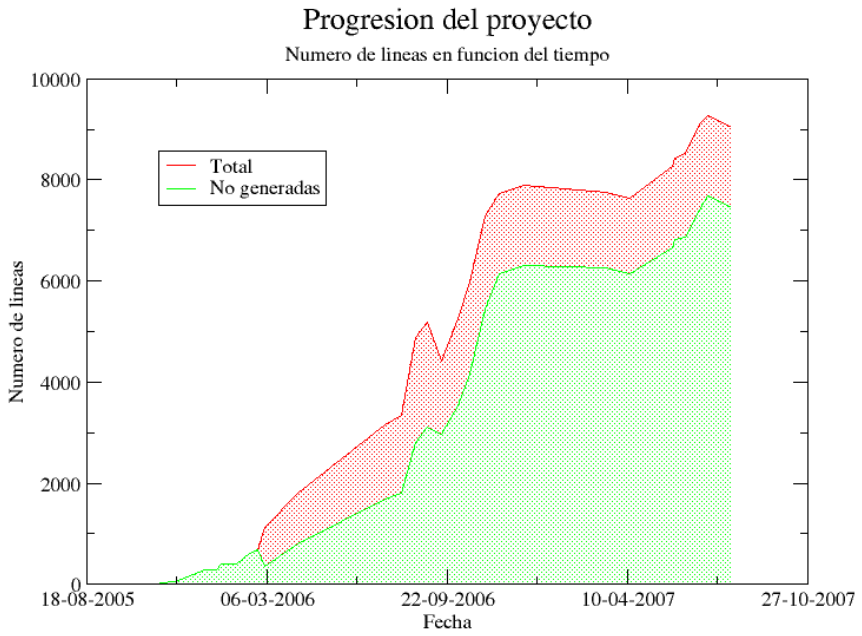


Figura 5.3: Número de líneas de código con respecto al tiempo

El número de líneas no es un estimador muy bueno de la magnitud del trabajo. En lenguajes con mucha sintaxis superflua, como C, puede haber mucho código redundante. Aunque en los lenguajes dinámicos la relación es más directa, sigue sin ser un buen estimador.

De la figura 5.3 cabe destacar el cambio de lenguaje de programación (Aproximadamente Marzo de 2006). Con el cambio de lenguaje, apareció el código generado.

Otra tendencia remarcable es la disminución del código generado a lo largo del tiempo. También cabe destacar los períodos en los que no se incluyeron nuevas características, en los que la cantidad de líneas de código se vio reducida.

5.3.2. Número de clases, ficheros y directorios

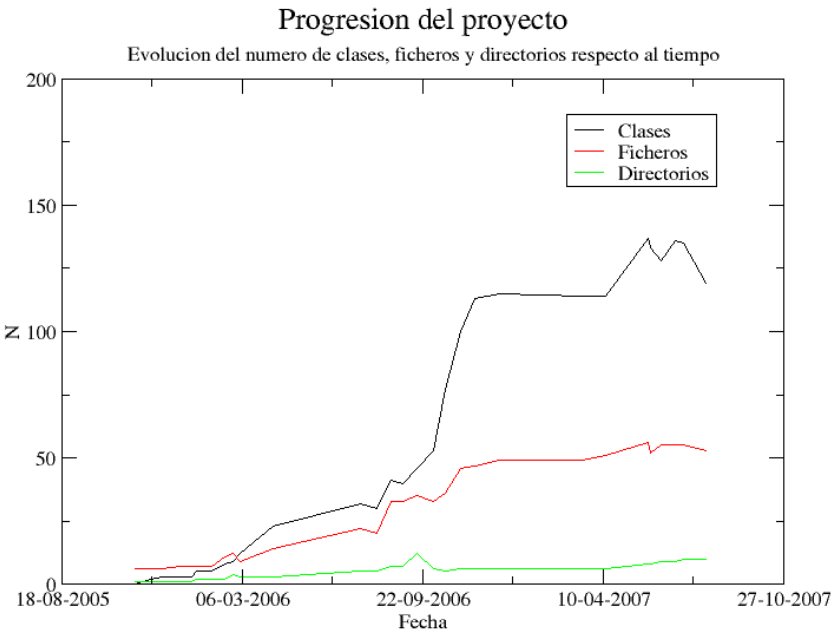


Figura 5.4: Número de clases, ficheros y directorios con respecto al tiempo

La cantidad de clases es una mejor medida de la magnitud del proyecto. En el esquema se puede apreciar las épocas de máximo crecimiento y de estancamiento del desarrollo.

El número de ficheros y de directorios dan una idea de la distribución y de la proporción de éstos.

5.3.3. Proporción de clases respecto al número de ficheros

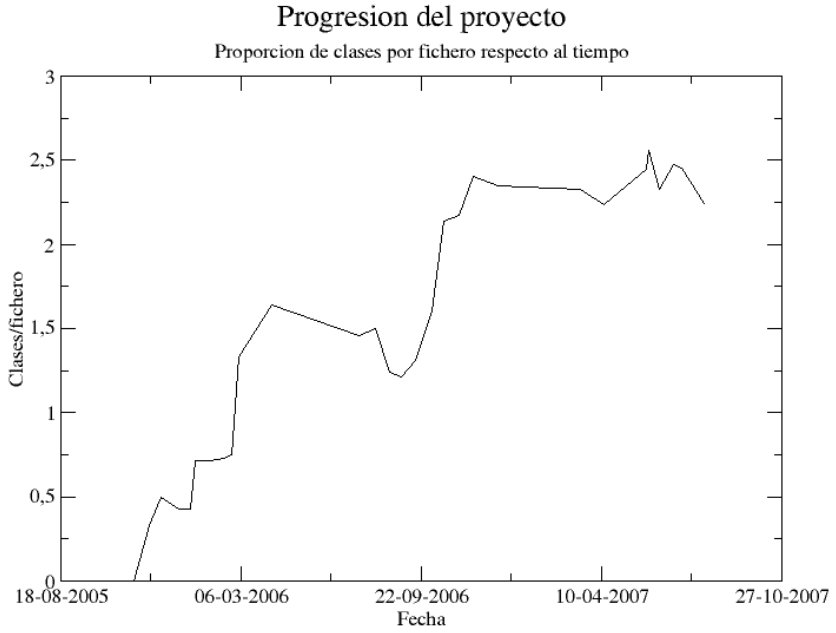


Figura 5.5: Fracción de clases y ficheros con respecto al tiempo

En el lenguaje de implementación no hay una cantidad mínima ni máxima de clases por fichero. Por tanto, es decisión del programador que aproximación tomar. En este proyecto se ha intentado aproximar los ficheros a unidades funcionales con significado, lo cual no fija ninguna tendencia acerca del número de clases por ficheros. No obstante, el paso del tiempo parece mostrar un límite asintótico en torno a 2.5.

5.3.4. Proporción de ficheros respecto al número de directorios

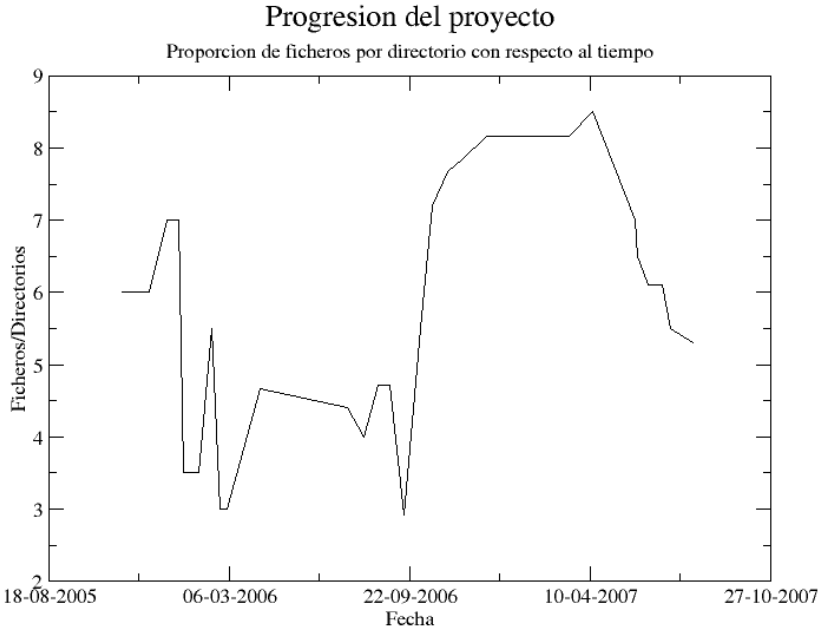


Figura 5.6: Fracción de ficheros y directorios con respecto al tiempo

Nuevamente, el lenguaje es una influencia determinante. En este caso, hay total libertad, y al igual que en el número de clases respecto al fichero se ha intentado llevar un esquema funcional. En la gráfica se puede apreciar los bruscos cambios debido a las restructuraciones, sin que parezca haber convergencia alguna.

5.3.5. Número de líneas de código por secciones

Figura 5.7: Líneas de código por secciones

En esta gráfica se puede apreciar la distribución de las líneas de código según secciones. Llama la atención el que la interfaz de usuario prácticamente suponga la mitad del código fuente del proyecto.

5.4. Convenciones empleadas

Convenciones que ayudan a la homogeneización y a la compatibilidad con entornos.

5.4.1. Convenciones de código

Las convenciones de código se han seguido y verificado gracias a un programa Pylint. Este programa muestra un listado de errores de estilo que tiene el código. Sirve además como ayuda al diseño, ya que detecta cuando un miembro de una clase puede ser una función independiente o redundante en el código. Algunas de las convenciones de código utilizadas son las siguientes

- Espacios en vez de tabuladores para la indentación.
- Una sentencia por línea.

- Después de un operador o una coma hay un espacio.
- Nombres de métodos y variables en español.
- Separación de nombres y variables compuestos con "_".

5.4.2. Autodocumentación

Python permite documentar las clases y funciones en el propio código fuente. Esta documentación puede ser consultada por el usuario en el intérprete. Por ejemplo:

```
>>> from Driza.config import GestorConfig
>>> help(GestorConfig)

Help on class GestorConfig in module Driza.config:

class GestorConfig
 |  Gestor de configuración. Se encarga de controlar si
 |  esta cargada la configuracion
 |  si el formato es el más actual, y mantiene una referencia al propio objeto
 |  de configuración
 |
 |  Methods defined here:
 |
 |  __init__(self)
 |
 |  cargar(self, fichero=None)
 |      Carga un fichero de configuración, por defecto ~/.driza
 |
 |  guardar(self, fichero=None)
 |      Guarda el fichero de configuración en un fichero
```

Esta característica facilita muchísimo el mantenimiento del código, y ayuda a que pueda ser entendido por otros desarrolladores.

5.4.3. La utilidad logging

Aunque sea poco habitual en los desarrollos de software, no se ha utilizado programa de depurado. La ventaja de los lenguajes dinámicos es que se puede ejecutar parte del código en el intérprete y ver el resultado y los efectos con

otros módulos. Sin embargo, para facilitar la corrección de errores, se ha utilizado un mecanismo de comunicación llamado *logging*.

Funcionamiento Los métodos envían mensajes de las acciones que realizan, y con el nivel adecuado a la importancia de la acción. Esta utilidad muestra mensajes que superen cierto umbral configurable por el usuario en la línea de comandos.

5.4.4. Convenciones de interfaz

En cuanto a la interfaz, no se ha seguido ningún documento sobre diseño de interfaces humanas, pero si que se ha intentado mantener cierta homogeneidad, a través de reglas sencillas:

- Árboles o elementos de selección en la interfaz en el lado izquierdo (operaciones,configuración, salida).
- Utilización de la tecla suprimir para borrar lo seleccionado.
- La tecla "Esc" fuerza la salida de cualquier diálogo.
- Toolbars en la zona superior del diálogo, con los elementos de uso mas frecuente.

Es previsible que en las próximas revisiones del software aparezcan mejores convenciones.

5.5. Pruebas

Las pruebas aseguran la calidad del software. En el caso de este proyecto, debido a la continua evolución, se debe mantener un régimen de pruebas continuo. Se ha utilizado tres metodologías de pruebas:

Pruebas de caja blanca Las pruebas se centraron, tras el estudio del código en las zonas más propensas a error. Concretamente, se aplicaron pruebas en las siguientes áreas

- Áreas de interpretación de expresiones.
- Carga dinámica de módulos, especialmente de operaciones.
- Manejo de ficheros

Pruebas de caja negra Las pruebas de caja negra consisten en considerar el software como algo cuyo funcionamiento no puede ser entendido. Como consecuencia de este enfoque, se centran en probar la entrada y salida. Los aspectos probados fueron los siguientes

- Entrada de texto en la ventana principal.
- Entrada de texto en las operaciones.
- Entrada de texto en los diálogos con expresiones.
- Argumentos del programa en la consola.

Pruebas hechas por usuarios Como el código fuente estaba disponible en Internet, algunas personas lo descargaron y utilizaron. De estas personas, algunas reportaron errores a la página de la forja, sirviendo como fuente de pruebas continuamente.

Capítulo 6

Resultado

En este capítulo se dará una pincelada general de la apariencia y funcionalidad del proyecto.

6.1. Ventanas y diálogos

La apariencia de un programa viene determinada por la apariencia de sus diálogos de interacción con el usuario. La diferencia entre ventanas y diálogos es que, mientras las primeras son entes independientes, los segundos dependen de una ventana para su existencia. Los diálogos tienen métodos de aceptación y cancelación, que realizan ciertas acciones y después devuelven el control a la ventana padre.

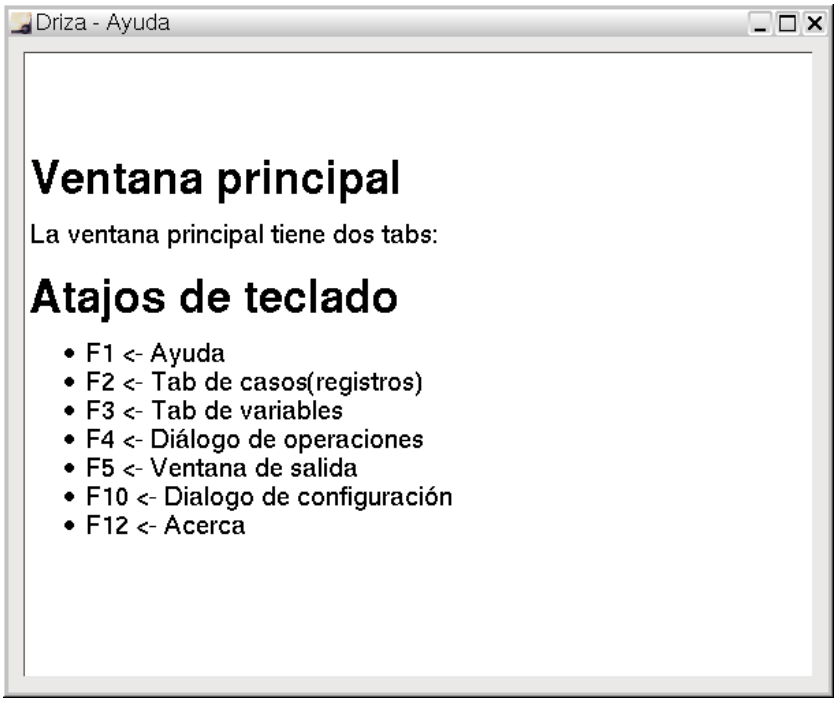


Figura 6.1: Diálogo de ayuda

6.1.1. Diálogo de ayuda

El diálogo de ayuda tiene como objetivo servir de explicación para un determinado contexto del programa. Su contenido es dinámico, se obtiene de plantillas en html, que son generadas por un lenguaje de descripción de contenidos llamado rst.

Zona principal

Zona donde se renderiza el html de la ayuda. Permite navegar a través de los enlaces del documento, y scroll cuando el contenido es mayor que el tamaño de la ventana.

Figura 6.2: Diálogo de Bienvenida

6.1.2. Diálogo de bienvenida

El diálogo de bienvenida es la portada del programa. Da a elegir al usuario entre las acciones que probablemente quiera hacer de forma inicial. El diálogo tiene dos zonas diferenciadas, la imagen de bienvenida y el área de trabajo, que es la única zona interactiva. Estas son las acciones que permite hacer:

- Comenzar un nuevo proyecto.
- Abrir un proyecto.
- Acceso a proyectos recientemente abiertos.
- Acceso a la ventana de salida.

Los enlaces a proyectos recientemente abiertos se generan dinámicamente, consultando a la configuración del programa. Una característica de este diálogo es que puede ser desactivado en la configuración.

6.1.3. Diálogo de búsqueda

Permite las búsquedas en la ventana principal. Solamente posee una entrada de texto y un botón de aceptación. El diálogo devuelve el control a la ventana principal en todos los casos, señalando aquel registro que contenga el texto a buscar. Es posible manejar el diálogo con el teclado, ya que se

Figura 6.3: Diálogo de Búsqueda

accede a él con la combinación de teclas "Control+F", se busca el texto con la tecla "Intro", y se cierra con la tecla "Escape"

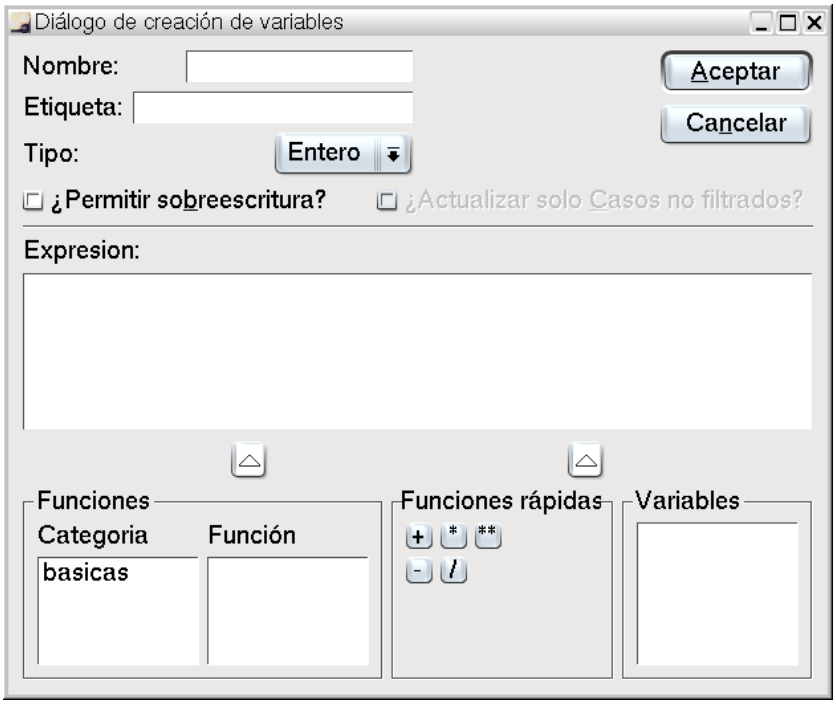


Figura 6.4: Diálogo de creación de variables

6.1.4. Diálogo de creación de variables

Este diálogo permite crear nuevas variables. Para ello pregunta al usuario por el nombre a introducir, la descripción, y la expresión que determinara sus valores. También permite controlar la sobreescritura de variables, y el comportamiento de escritura de los nuevos registros.

Área de introducción de expresiones

En este área introduciremos una expresión. La expresión estará formada por variables y funciones que relacionan a éstas. Si la expresión no es correcta, el programa generará un aviso con el error.

Botones de asistencia

Los botones facilitan la introducción de expresiones, ya que solamente es necesario utilizar el ratón. La asistencia que permite los botones son las siguientes:

Variables: Permite escoger una variable del mismo tipo e introducirla en la caja de texto.

Funciones: Permite, agrupándolas en categorías, escoger cualquiera de las funciones incluidas en el programa.

Funciones rápidas: Da acceso a las funciones rápidas, es decir, aritméticas.

El diálogo se comunica con usuario a través de ventanas de aviso en caso de error.

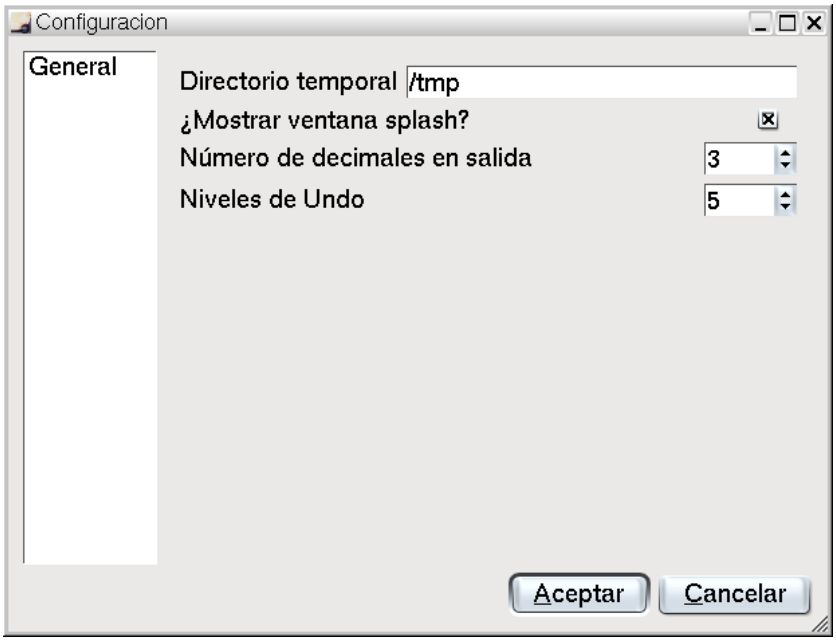


Figura 6.5: Diálogo de configuración

6.1.5. Diálogo de configuración

El diálogo de configuración permite al usuario configurar diferentes aspectos del programa. Separa en secciones los diferentes aspectos a configurar, para evitar complicar demasiado la interfaz. El diálogo está dividido en dos áreas:

Lista de secciones: Muestra una lista con las secciones disponibles para configurar, permitiendo escoger la que desee el usuario.

Widget de configuración: Muestra los elementos que se pueden configurar en la sección seleccionada. Las secciones disponibles son las siguientes:

General: Esta compuesto de los elementos de configuración general, que son los siguientes:

- Directorio temporal
- Mostrar Ventana splash
- Número de decimales de salida
- Niveles de Deshacer

Botones: Permiten guardar los cambios y aceptar o cerrar el diálogo.

6.1.6. Diálogo de filtrado

Con este diálogo podemos escribir la expresión que actuará de filtro para los resultados, o desactivarlo.

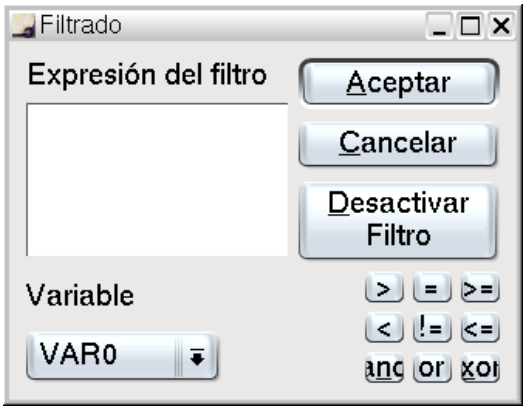


Figura 6.6: Diálogo de filtrado

Se compone de un área de introducción de expresiones, y varios elementos que facilitan la labor al usuario.

Área de introducción de expresiones: En esta zona el usuario puede introducir la expresión que determinará el filtro. Si el usuario introduce una expresión errónea, aparecerá una ventana de aviso.

Botones: Los botones facilitan la introducción de expresiones.

- Botones de funciones lógicas (comparadores, and, or, xor).
- Pestaña con las variables disponibles

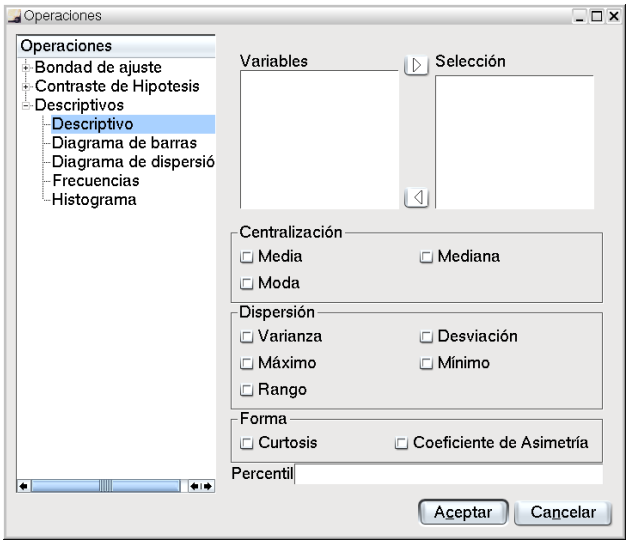


Figura 6.7: Diálogo de operaciones

6.1.7. Diálogo de operaciones

El diálogo de operaciones es el encargado de permitir elegir al usuario una operación, y de insertar los valores necesarios a ésta. Solamente tiene dos componentes:

Árbol de operaciones

Muestra, respetando la jerarquía, las diferentes operaciones. Cada elemento puede colapsarse o expandirse. El elemento seleccionado es el que se muestra en el widget de operaciones.

Widgets de operaciones

El widget de cada operación. Se renderiza a partir de la definición de la operación, mostrando todas sus opciones. Las diferentes operaciones aparecen detalladas en la sección Operaciones de este capítulo.

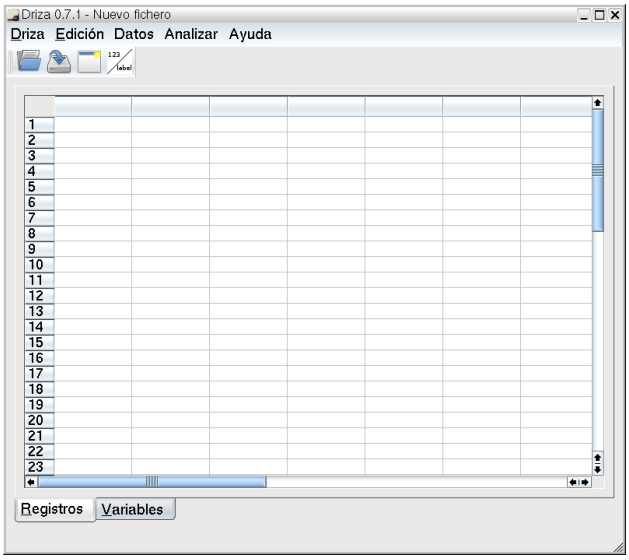


Figura 6.8: Ventana principal

6.1.8. Ventana principal

Este es el diálogo principal. Los elementos visuales principales son los siguientes

1. Menú
2. Toolbar
3. Área de trabajo
4. Pestaña de selección de tabla

El menú

La estructura del menú es estática, salvo el submenu de operaciones. Esta es su estructura, con una descripción de cada elemento:

Driza: Están las acciones de manipulación de proyecto, la importación de ficheros de texto, la configuración, y la salida del programa

Edición: Las funcionalidades básicas de edición, es decir, copiar, cortar, pegar, deshacer, rehacer y buscar

Datos: Funcionalidades un poco más complejas. Filtrado y creación de nuevas variables.

Analizar: Generado dinámicamente con todas las operaciones disponibles

Ayuda: Acceso a la ayuda, y al diálogo "Acerca de"

Diálogos de ficheros

Los diálogos de ficheros son relativamente sencillos. Permiten navegar por el sistema de fichero filtrando el tipo de fichero que se desea.

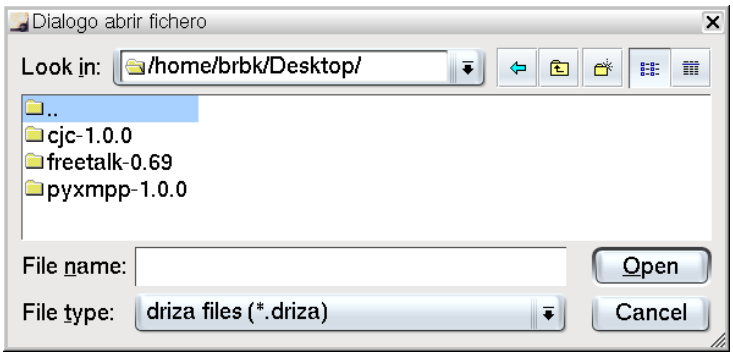


Figura 6.9: Diálogo de apertura de proyecto

ToolBar

El toolbar es un conjunto de botones que permiten el acceso rápido a las funcionalidades más usuales de la ventana. Las operaciones que permite la toolbar son las siguientes:

- Abrir un proyecto
- Guardar el proyecto
- Acceso a la ventana de salida
- Cambiar al modo "etiquetas"

Área de trabajo

En este área el usuario puede introducir los datos y las variables. También puede seleccionar, cortar y pegar. El mecanismo para cambiar entre las vistas de la ventana de trabajo es una pestaña situada en la parte inferior.

En esta zona se ha de ver el contenido exacto de los datos internos, sea cual sea el estado en el que se encuentre.

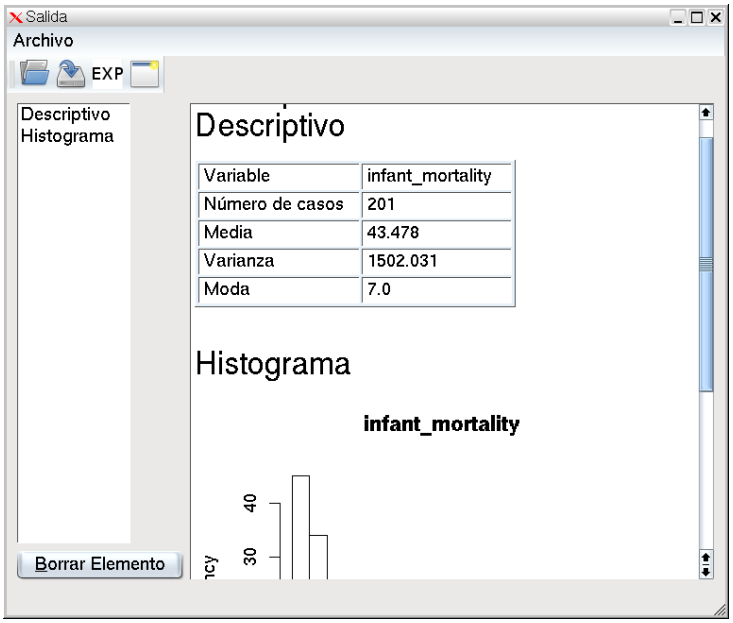


Figura 6.10: Ventana de salida

6.1.9. Ventana de salida

En esta ventana aparecen los resultados de las operaciones seleccionadas por el usuario. Es completamente independiente a la ventana principal. Sus componentes son los siguientes:

El menú

Solamente se controla la gestión de archivos a través del menú:

- Abrir un fichero.
- Guardar un fichero.
- Exportar a html.

Diálogos de ficheros

Son equivalentes a los de la ventana principal, salvo en el filtro utilizado para los ficheros.

ToolBar

Son un acceso rápido a las funciones más frecuentes de la ventana.

- Abrir un fichero de salida
- Guardar un fichero de salida
- Acceso a la ventana principal
- Exportar a html

Lista de resultados

Es una lista donde aparecen los nombres de los resultados. Su objetivo es permitir borrar un resultado no deseado, acción que es posible realizar a través del botón de la parte inferior (y con la tecla "Suprimir")

Área de visualización de resultados

En este área se puede visualizar los resultados según van apareciendo. Los resultados anteriores pueden ser consultados con la barra de desplazamiento.

6.2. Operaciones

En las próximas secciones abordaremos la descripción de las operaciones implementadas.

6.2.1. Descriptivos

Los estudios descriptivos se realizan para cada variable de forma independiente. Hay varios cálculos disponibles, pudiendo ser solicitados de forma simultánea. Por último, los percentiles requieren introducir una cantidad por parte del usuario.

Componentes: Al ser necesario solamente una variable por vez, un selector de Variable Simple es suficiente. Respecto a las opciones, la operación requiere dos elementos:

- Una selección de múltiples elementos.
- Una entrada de texto, para introducir el percentil.

Requisitos de funcionamiento: Esta función requiere que exista al menos una variable numérica.

Salida a mostrar: Al tener únicamente resultados numéricos, y ser varios por variable, parece que el elemento óptimo de representación es la tabla.

6.2.2. Tabla de frecuencias

La tabla de frecuencias examina todos los casos de la variable. Obliga por tanto a que ésta sea discreta, ya que no se puede hablar de casos en las variables continuas

Componentes: Como necesita una variable por vez, el selector más correcto es el simple. No requiere opciones.

Salida a mostrar: Como el nombre de la operación indica, el formato que debe tener es una tabla.

6.2.3. Histograma

Un histograma de frecuencias es un estudio gráfico. Permite que la variable sea tanto discreta como continua

Componentes: Como necesita una variable por vez, el selector más correcto es el simple. La única opción a introducir es el número de segmentos que tendrá la gráfica. Al ser un campo a introducir por el usuario, se utilizará una entrada de texto

Salida a mostrar: Al ser un estudio gráfico, requiere mostrar una imagen generada por R.

6.2.4. Diagrama de barras

Es un estudio gráfico para variables discretas en el que se muestra la frecuencia de cada valor.

Componentes: Como necesita una variable por vez, el selector más correcto es el simple. No requiere opciones.

Salida a mostrar: Al igual que en el resto de estudios gráficos, solamente se muestra una imagen generada por R.

6.2.5. Tests de bondad de ajuste

Las contrastes de hipótesis χ^2 y Kolmogorov-Smirnov sólo difieren en el funcionamiento interno. En ambas operaciones se busca contrastar si una muestra se ajusta a una distribución.

Componentes: Lo que se ha de seleccionar, por tanto, es una sola variable. Como opción aparecerá la distribución a escoger.

Salida a mostrar: La salida se compone de tres tablas:

- Una tabla con descriptivos
- Un contraste que verifique la igualdad de varianzas
- Una tabla con dos filas
    - Se asume varianzas iguales
    - Se asume varianzas distintas

6.2.6. Comparación de medias de muestras independientes (en la misma variable)

Dada dos poblaciones, se pretende probar la hipótesis nula:

$$H_0 : \bar{x}_0 = \bar{x}_1$$
$$H_1 : \bar{x}_0 \neq \bar{x}_1$$

Para ello utilizamos la T de student (Bilateral)

Componentes: En este caso se pretende comparar la media de dos muestras pertenecientes a la misma variable. Por tanto, hace falta una segunda variable que actúe como discriminador, y dos de sus valores. El selector más apropiado es "una Variable y un Discriminador Doble"

Requisitos de funcionamiento: Requiere que exista al menos una variable numérica y una variable discreta.

Salida a mostrar: Para cada pareja de casos se ha de mostrar una tabla con los siguientes datos:

- Valor t obtenido asumiendo varianzas iguales.
- Valor t obtenido asumiendo varianzas distintas.
- Rango t en el que se verifica la hipótesis nula.

Capítulo 7

Difusión

Los mecanismos de difusión son parte del desarrollo del software libre. En este proyecto, como se podrá apreciar en las próximas secciones, no hubo demasiada repercusión. Aún así, es posible que esta tendencia cambie con el tiempo.

7.1. El blog

Un blog es un sitio en Internet donde una persona u organización publican notas o posts. Hay dos blogs que han servido como mecanismo de difusión.

7.1.1. El blog suministrado por la OSL

La OSL brindó a los proyectos un espacio web en el que escribir noticias acerca del desarrollo. [1].

Inconvenientes: Uno de los inconvenientes del blog fue la falta de mantenimiento. Tenía problemas con las hojas de estilo y edición. Otro de los inconvenientes que impidieron el uso del blog fue el hecho de no tener las condiciones de compartición de antemano. Por último, aparecieron mensajes de spam en los comentarios, dejando la funcionalidad de comentar inservible.

Así que finalmente lo traslade a otro blog de blogger (Registro gratuito), aunque no era un blog exclusivo del proyecto.

[1] La url del blog de la OSL es http://osl.ulpgc.es/plog/index.php?blogId=2

7.1.2. El blog de blogger

En este blog[2], con mejor presentación, fueron publicados algunos posts. También fue publicado en el planeta dis, que muestra las entradas de varios blogs simultáneamente. Pese a todo, el blog no recogió demasiada atención, así que no fue mantenido.

7.2. La forja

La forja de rediris ofrece un espacio web para proyectos de software libre de ámbito universitario. El 17 de Junio de 2006 fue solicitado el registro del proyecto en la forja, con el nombre *Driza*. Es un instrumento muy completo. Estas fueron las características empleadas:

Página web

La forja rediris suministra a todos los proyectos un espacio web. En este espacio web el administrador puede colocar la información que desee. En el caso del proyecto driza, las categorías introducidas fueron las siguientes:

Página principal: En esta zona publiqué los enlaces a las entidades responsables, junto a una descripción de las noticias

Motivación e historia: Esta sección comenta el origen del proyecto. En su realización colaboró Carlos Mestre González.

Sección de desarrolladores: En esta sección hay una descripción de las tareas en las que puede colaborar un desarrollador. También existe una copia comentada de todo el código, junto a esquemas autogenerados con una herramienta llamada Doxygen

Screenshots: Por lo general, cuando alguien puede estar interesado en un programa echa un vistazo a la sección de screenshots para hacerse una idea de lo que puede ofrecer.

Sección de descargas: Es un enlace rápido a la descarga de versiones de la forja.

Enlace a la forja: Enlaza con la sección del proyecto en la forja.

[2]La url del blog de blogger es http://nesaro.blogspot.com

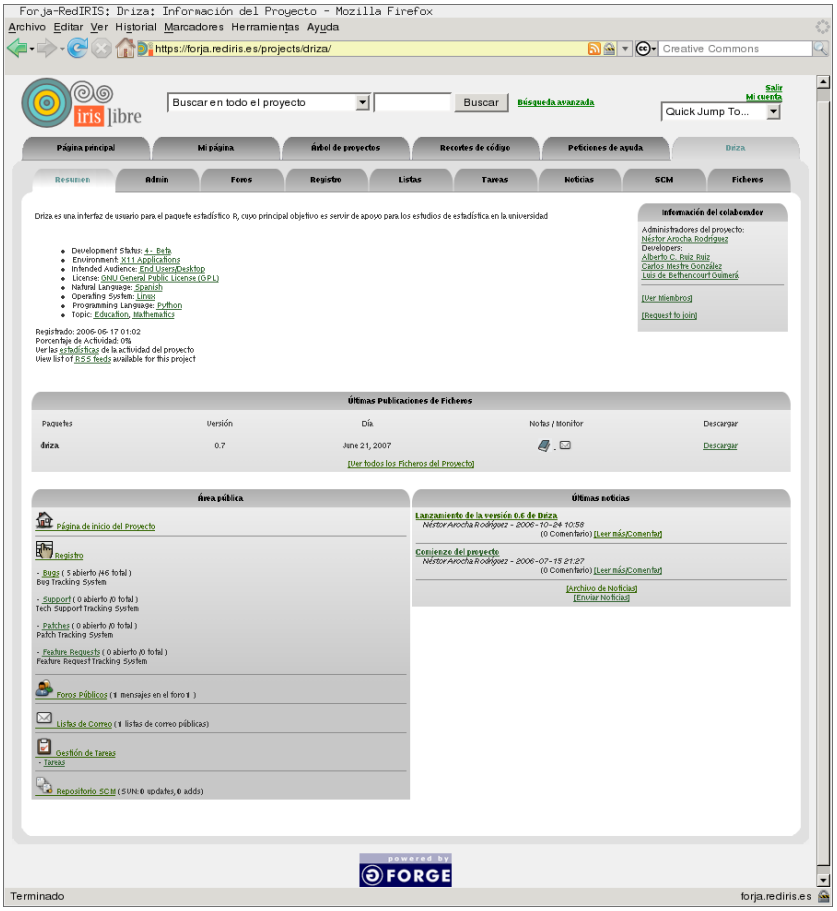


Figura 7.1: Sección Driza en la forja Rediris

Los registros

La sección de registros engloba cualquier petición registrable, esto es, bugs, soporte, parches y peticiones de características. La única característica utilizada fue la de bugs, donde varios colaboradores pudieron notificar diversos errores que encontraron.

Listas de correo

Durante el desarrollo fue abierta una lista de correo para que quien quisiera pudiera hacer cualquier aportación. Fue complementario a los foros, y permitió hacer anuncios de desarrollo (Tal como la salida de una versión)

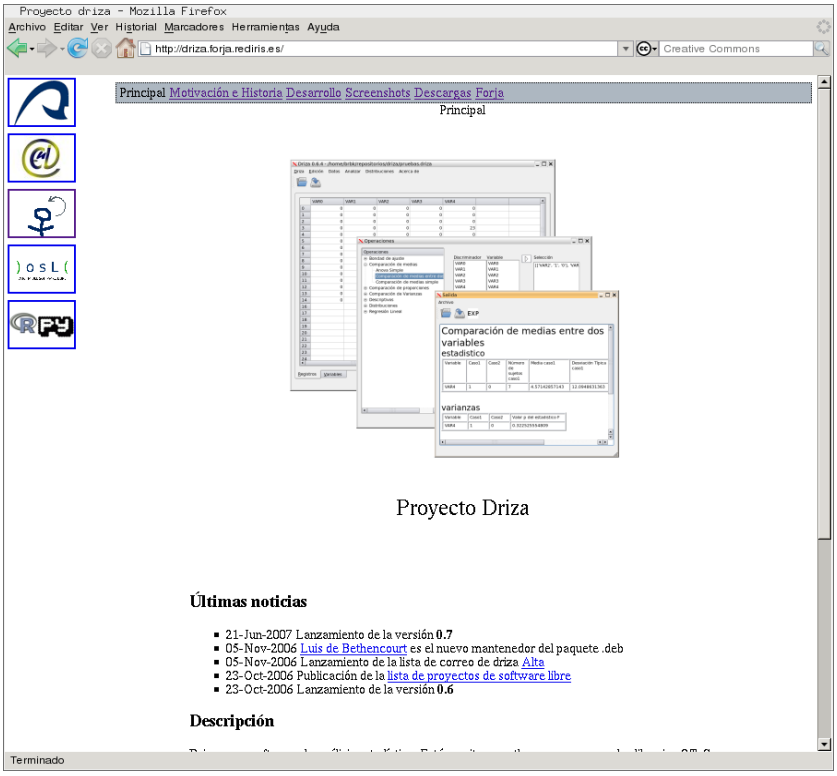


Figura 7.2: Página principal del proyecto Driza

Emisión de noticias

Es un medio de comunicación con el que se puede dar una noticia e, incluso, ser mostrada en la portada de la forja.

Subversión

Subversión es un sistema de control de versiones que permite a cualquier persona ver el estado actual del código. Es una de las ventajas de la forja. Además de tener el repositorio externo, permite la navegación por web a través de su contenido.

Gestión de ficheros

Esta sección permite gestionar las diferentes versiones del programa. Cada cierto tiempo, coincidiendo con el ingreso de características en el programa, se lanzaban nuevas versiones para que la gente pudiera probar el programa.

En el gestor de ficheros aparece reflejada la fecha, las características que incluye cada una y las diferencias con las anteriores.

7.3. Comentarios en otros sitios web

Durante el desarrollo, varias personas publicaron información en foros sobre el proyecto. En este sentido, cabe destacar la labor de la OSL, ya que en varios de sus documentos aparece mencionado.

Lista de Software Libre de la OSL: También fue publicada una nota en la lista de software libre mantenida por la OSL.

Freshmeat: Freshmeat es un directorio de software mayoritariamente libre. En este directorio publiqué una entrada sobre el proyecto, para mejorar la repercusión.

7.4. Información en buscadores de código

Google code search es un servicio de Google que escanea los sitios donde esta publicado el código de proyectos (como la forja), indexandolo según licencia, lenguaje de programación y otros valores. Con el servicio de búsqueda de código, es posible que programadores con intereses similares encuentren información acerca de este proyecto.

Capítulo 8

Conclusiones

El valor del proyecto es relativo. Como programa podría ser usado por mucha gente, quizá por nadie. Como créditos, pues ahí están, pero no tienen gran valor fuera del contexto de final de carrera. El verdadero valor del proyecto es la experiencia que queda a aquellos que hayan participado. Los próximos apartados comentarán por qué, a juicio del autor, hubo contrariedades en la realización del proyecto.

8.1. Los retos del proyecto

La gran duración del proyecto no es casual, se debe a que han habido diferentes retos que no siempre han tenido fácil solución.

8.1.1. Un proyecto demasiado extenso

Sin duda alguna, este es el gran problema del proyecto. Podría considerarse que el requería el trabajo de más de una persona. Tal vez haber contado con la reducción de objetivos desde el principio habría ayudado a no haberlo hecho tan extenso. Quizá haya sido algún factor sin análisis. Lo que es un hecho es que el número de horas de desarrollo no se corresponden con el número de horas pactadas.

8.1.2. Problemas en las etapas de análisis y diseño

El principal problema del análisis fue la mala elección de la librería gráfica y el lenguaje de programación. Hizo aumentar el número de horas de trabajo innecesariamente, ya que obligó el estudio de dos librerías gráficas y un lenguaje de programación nuevo. En el caso del diseño, la falta de experiencia

en desarrollos grandes hizo que el diseño estuviese un paso por detrás del desarrollo.

Es decir, el diseño surgía como necesidad cuando el desarrollo se complicaba demasiado. Como consecuencia de la precariedad del diseño, el desarrollo fue más lento de lo deseable.

8.1.3. Mala metodología

La falta de experiencia redundo en peor calidad en la comunicación entre los componentes del proyecto. El uso extendido de artefactos como los casos de uso habría simplificado la complejidad del proyecto.

8.1.4. Las peculiaridades del software libre

El software libre es un modelo muy diferente al modelo tradicional de software. La competencia en el software libre no es una cuestión económica, sino de dedicación por parte de los programadores y de atención por parte de los usuarios. El interés por mantener a los usuarios atados a un formato concreto o a una forma de hacer las cosas no redunda en beneficio para el programador.

También existe mucho más espacio para las iniciativas independientes. Un programador puede decidir hacer un nuevo proyecto, o modificar uno existente por diferencia con los mantenedores de ese proyecto.

Todas estas características hacen que el software libre sea un mundo lleno de alternativas y de formas de hacer las cosas.

Pero este planteamiento tiene varios inconvenientes. En primer lugar, los programadores de software libre trabajan por intereses propios en su mayoría. Esto provoca que por ejemplo, la documentación de los proyectos sea por lo general pobre.

Al no existir un mercado de competencia económica, no existe una necesidad real de que las cosas estén acabadas al 100 %, o que tengan funcionalidad plena.

Todos estos argumentos redundan en una serie de ventajas e inconvenientes:

Ventajas

- Alta reutilización del código de terceros.
- Muchas alternativas para resolver un mismo problema.
- Bajo coste y alto grado de colaboración.

Inconvenientes

- Alto grado de incompletitud en muchos proyectos.
- Dificultad de elección por exceso de oferta.
- Incertidumbre asociada a la dependencia de la iniciativa no económica de terceros.
- Documentación pobre en algunos casos.

8.1.5. Conseguir el éxito con un blog

Recoger atención es una actividad que requiere habilidad. En Internet, cualquiera puede arrancar cualquier proyecto. Pero conseguir que ese proyecto tenga relevancia requiere moverse con habilidad entre los foros y los usuarios, notificar su existencia en los sitios adecuados y saber mantener la expectación. Durante este proyecto, la aparición en Internet no ha cuajado, ni se ha convertido en un gran revuelo.

Probablemente sea necesario que el proyecto aumente sus funcionalidades y que tenga cierta repercusión para poder atraer más atención.

8.2. Futuro del proyecto

Aún quedan muchos aspectos para que *Diseño e implementación de una interfaz orientada a la docencia para el paquete estadístico R* se convierta en una alternativa que mejore las ya existentes. Nuevos frentes de desarrollo deberán cubrir las funcionalidades que falten. Estos nuevos frentes de desarrollo podrían mejorar considerablemente la calidad del proyecto. Quizá incluso dar pie a nuevos proyectos.

8.2.1. Portabilidad a otras librerías gráficas

Hay varias librerías gráficas en las que sería interesante tener una interfaz alternativa. Sin ir más lejos, las librerías Qt, creadas y mantenidas por Trolltech, tienen una nueva versión. Las mejoras son varias, pero la más relevante para el proyecto es la aparición del código GPL para Windows. Este hecho facilitará razonablemente la portabilidad a Windows. También podría existir en el futuro una interfaz para texto, por ejemplo en Ncurses.

8.2.2. Instalación en otros sistemas operativos

Una vez satisfecho el cambio de librería, para aumentar el número de usuarios del programa, y favorecer la difusión, sería conveniente soportar correctamente la plataforma Microsoft Windows, así como crear un instalador.

8.2.3. Nuevas operaciones de cálculo

El repertorio de operaciones actual es muy básico. La implementación de nuevas operaciones haría el programa más atractivo, mejorando la difusión del proyecto.

8.2.4. Importación de otros formatos

Quizá sea la asignatura pendiente del proyecto. El poder obtener información de Excel, Openoffice, del SPSS o de cualquier otro programa, ya sea a través de ficheros o de portapapeles, aumentaría bastante la utilidad del programa. No obstante, para el caso de formatos cerrados para los que el trabajo no esta hecho, podría requerir mucho esfuerzo.

8.2.5. Soporte para bases de datos

No es infrecuente que los datos de interés se encuentren en alguna base de datos mantenidas por un tercer programa. Al igual que otros programas de estadística, suministrar conectores para bases de datos SQL permitiría tener sin intermediario acceso a los datos.

8.2.6. Soporte de idiomas

Junto con el instalador de Windows sería la medida que podría atraer mayor atención. Tener soporte de otros idiomas (especialmente de inglés) aumentaría el público potencial de forma exponencial.

Capítulo 9

Anexos

9.1. GNU Free Documentation License

Version 1.2, November 2002

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image

format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements".

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in

an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will

be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright © YEAR YOUR NAME. Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.2 or any later version published by the Free Software Foundation; with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts. A copy of the license is included in the section entitled "GNU Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with ... Texts." line with this:

> with the Invariant Sections being LIST THEIR TITLES, with the Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

Bibliografía

[1] C. Freedesktop. Clipboard specs. http://standards.freedesktop.org/clipboards-spec/clipboards-latest.txt, 2005. [En linea, accedido el 2 de Agosto de 2007].

[2] M. Pilgrim. Dive into python, 2004.

[3] R. M. Stallman. Various licenses and comments about them. http://www.gnu.org/philosophy/license-list.html, 1999. [En linea, accedido el 29 de Julio de 2007].

[4] R. E. Walpole, R. H. Myers, and S. L. Myers. *Probability and Statistics for Engineers and Scientists*. Prentice-Hall, Inc., 1998.

[5] C. Wikipedia. Interfaz gráfica de usuario. http://es.wikipedia.org/wiki/Interfaz_gr%C3%A1fica_de_usuario, 2007. [En linea, accedido el 29 de Julio de 2007].

[6] C. Wikipedia. Multipurpose internet mail extensions. http://es.wikipedia.org/wiki/Multipurpose_Internet_Mail_Extensions, 2007. [En linea, accedido el 2 de Agosto de 2007].

[7] C. Wikipedia. Patrón de diseño. http://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o, 2007. [En linea, accedido el 29 de Julio de 2007].

[8] C. Wikipedia. Software libre. http://es.wikipedia.org/wiki/Software_libre, 2007. [En linea, accedido el 1 de Agosto de 2007 (??)].