8. IMPLEMENTACIÓN

En este capítulo se explican las herramientas utilizadas para codificar todo lo expuesto en la especificación y diseño y el estilo de implementación.

Por un lado se expone, sabiendo ya cual será la tecnología del SGBD (Sistema Gestor de Base de Datos) y el lenguaje de programación que herramientas adicionales se han usado para construir el sistema.

Por otro lado se expone cual ha sido el "modus operandi" en el desarrollo respectando la especificación y el diseño, y tomando como ejemplo la sencilla ampliación del sistema a la hora de añadir clases de objeto.

8.1 Tecnologías utilizadas para la implementación

En el desarrollo de un proyecto siempre se conjuntan varias tecnologías y herramientas para poder llegar al objetivo final, que es codificar el sistema descrito en la fase de especificación y diseño.

La descripción de las tecnologías utilizadas se centrará en:

- El sistema operativo utilizado para el desarrollo: Linux.
- El lenguaje de programación utilizado: Java
- El entorno de programación utilizado: Netbeans
- Las librerías adicionales utilizadas con Java: xStream

8.1.1 Linux

Puesto que el sistema final, por el momento, va a funcionar sobre un sistema Unix, se ha planteado desarrollar en un entorno muy parecido: Linux.

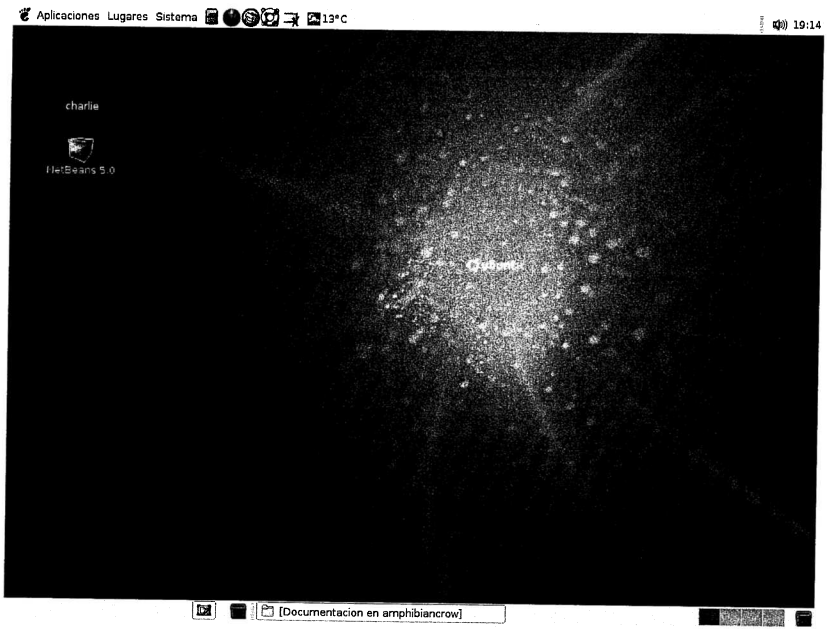
Linux es un sistema operativo de tipo Unix, que implementa el estándar POSIX (**P**ortable **O**perating **S**ystem **I**nterface y la **X**, que vendría de la herencia de las librerías de Unix.) y utiliza, principalmente, filosofías y tecnologías libres, encasilladas en el proyecto GNU (que hace referencia a un proyecto de construcción de un sistema operativo completo y libre desarrollado por Richard Stallman).

El sistema operativo Linux se utiliza con muchas utilidades, entre las cuales destacan:

- Como servidor
- Como alternativa a otros sistemas operativos en que la licencia no es gratuita. Este punto lo han explotado muchas administraciones públicas para ahorrar costes.
- Como entorno de programación

Además, Linux, ofrece variantes llamadas distribuciones. Algunas empresas privadas, como por ejemplo IBM, han sacado sus propias distribuciones.

En el caso de este proyecto la distribución utilizada es Ubuntu Linux 5.04, que es una variante de la distribución Debian.



8.1.2 Java

Java es un lenguaje de programación orientado a objetos que fue desarrollado por Sun Microsystems a principios de los años noventa.

A diferencia de otros lenguajes de programación, Java es un lenguaje compilado en un bytecode que es interpretado por una máquina virtual Java.

Tiene una sintaxis muy parecida a C y C++, aunque se diferencia de estos lenguajes en varios aspectos, uno de los cuales, de los más importantes, es que la gestión de memoria la realiza la máquina virtual Java mediante un sistema llamado Garbage Collector.

Fue creado con la siguiente filosofía:

- Debe usar la metodología de orientación de objetos.
- Debe permitir ejecutar un programa en diferentes plataformas hardware y software.
- Debe ser fácil de usar y debe recoger lo mejor de otros lenguajes.

A pesar de que los puntos clave de su filosofía tienen un buen propósito, Java ha sido un lenguaje que no ha sido muy usado, tradicionalmente, para aplicaciones de escritorio por varios motivos:

- Según como esté estructurada una aplicación Java, puede necesitar muchos recursos.
- Hay varias versiones de JRE (**J**ava **R**un **E**nvironment), que es el componente que representa la máquina virtual de Java. Si no se tiene instalada la adecuada, puede causar problemas de compatibilidad.
- La Interfaz Gráfica de Usuario es costosa de implementar y no sigue estrictamente la Guía para Interfaces Humanas.

En los últimos años han salido versiones que han ido solucionando todos estos motivos, aunque también es cierto que su uso se ha enfocado más a desarrollar sistemas webs.

Independientemente de ello, se ha utilizado este lenguaje en este proyecto por las causas anteriormente mencionadas.

Un ejemplo fácil de Java (el *helloworld.java*) podría ser el siguiente:

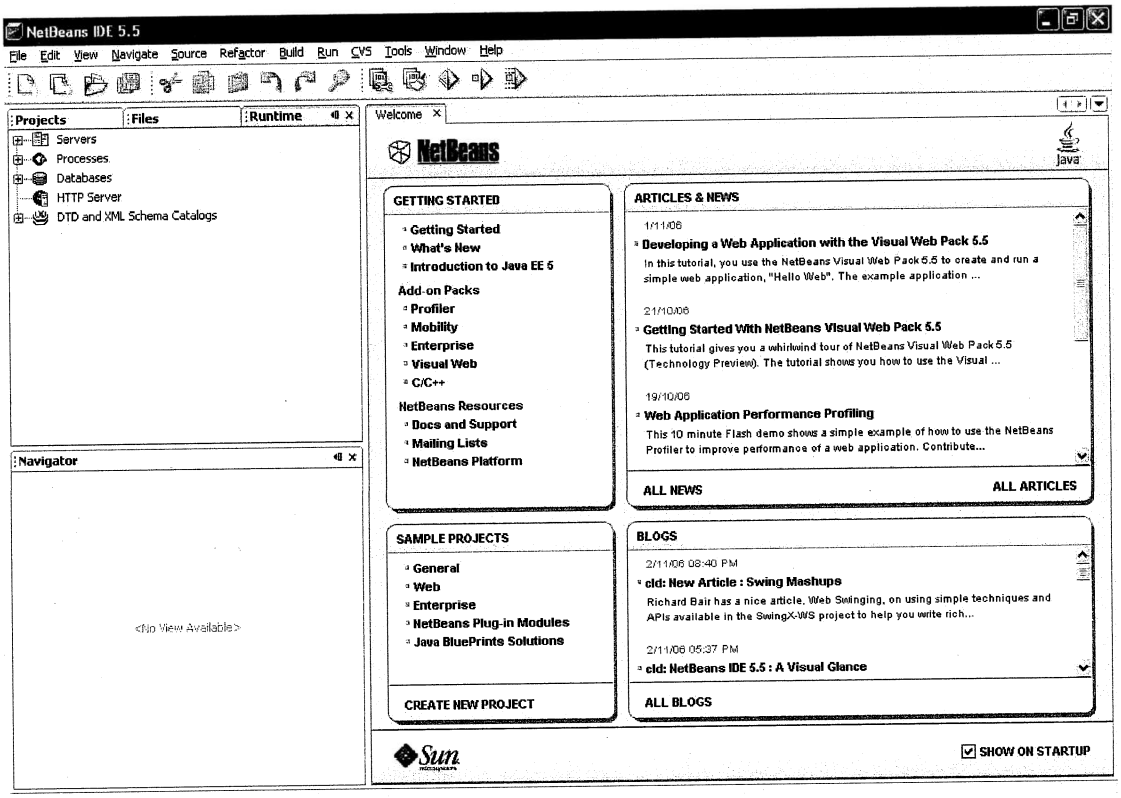
```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("hello world!");}
    }
```

8.1.3 Netbeans

Netbeans es una plataforma de desarrollo Java, implementada en este mismo lenguaje, que permite realizar las siguientes tareas:

- Administración y gestión de las interfaces de usuario
- Administración y gestión de la gestión de configuraciones de usuario.
- Administración y gestión de la carga y guardado de cualquier tipo de elemento.
- Administración de ventanas y frames.
- Asistentes predefinidos.
- Gestión de repositorios externos.

Una imagen de esta plataforma es la que se puede ver en la siguiente figura.



El origen de esta plataforma es el proyecto de unos estudiantes checos en 1996, con la finalidad de escribir un entorno de desarrollo para Java parecido al que ofrecía el del lenguaje Delphi.

Las versiones utilizadas para desarrollar este proyecto van desde la versión 4.1 hasta la última versión, la 5.5. Este entorno de desarrollo es gratuito y, actualmente, se puede descargar conjuntamente con el lenguaje de programación Java de la página de Sun Microsystems para diferentes plataformas.

8.1.4 xStream

xStream es una librería que permite serializar objetos Java al formato XML y viceversa. Además es una librería con licencia de código abierto desarrollada por Joe Walnes.

Alguna de las funcionalidades que ofrece esta librería son:

- XML sencillo: el código XML generado por esta librería es fácil de entender si se lee mediante un editor de textos.
- Fácil de usar: cargar y descargar un objeto es una tarea sencilla.
- No necesita modificar los objetos para guardarlos.
- Tiene un control de errores.

A continuación se ofrece la operación de lectura implementada en el código dentro del componente "DataManager":

```
/** Reads an object */
   public Object read (String path,Class c,int id)
   {
     /** Variables */
      Object result=null;
      String nombreArchivo;

      /** Nombre objeto a buscar */
      nombreArchivo = c.getSimpleName();

      /** Comprobación  existencia path */
      path = path + "/" + nombreArchivo + "_" + id+".xml";

      /** Carga del objeto */
      FileManager fm = new FileManager();
      String cadenaXml=fm.read(path);

      //alias
      x.alias(nombreArchivo,c);

      /** Parsear XML */
      result = x.fromXML(cadenaXml);

      return result;
   }//read
```

La parte señalada en rojo muestra la facilidad de uso de esta librería. La versión que se ha utilizado para desarrollar el sistema ha sido la 1.1.2.

8.2 Estilo de programación

El estilo de programación es importante no sólo de cara a la legibilidad del código sino también a la mantenibilidad.

Se ha intentado en este proyecto que la implementación siguiera una cierta guía de estilo:

- Se ha aplicado diseño descendente.
- Se ha intentado que el nombre de las variables fuese significativo
- Se ha indicado de qué tipo son las variables utilizadas.
- Cada función asume sus responsabilidades.
- Se ha comentado el código en idioma inglés, por poner un idioma estándar (en muchos proyectos donde participan personas de diferentes nacionalidades no todas tienen porqué saber el idioma de origen del proyecto).
- La mayoría de comentarios siguen el estándar de Java con la finalidad de poder usar la herramienta JavaDoc y generar automáticamente documentación sobre las clases.

8.3 Otras cuestiones de implementación

En este apartado se ve un ejemplo de una de las características del análisis de requerimientos, de la especificación y el diseño: la característica de escalabilidad.

Recordando lo que representa esta característica tenemos que:

"... el sistema debe ser construido sobre la base de un desarrollo evolutivo e incremental de manera tal que si aparecen o se desean añadir nueva funcionalidades y requerimientos relacionados puedan ser incorporados afectando al sistema existente de la menor manera posible. Se pretende, por tanto, incorporar aspectos de reutilización de código...".

Se va a ver esta característica con un ejemplo con afectación a la implementación (se supone que se ha seguido el ciclo de una iteración una reestructuración para llegar hasta este punto).

El ejemplo que se verá a continuación es un ejemplo sencillo para mostrar como se debería añadir una clase más en el dominio.

8.3.1 Ejemplo

Se supone que se ha llegado a un punto avanzado en la implementación del sistema y se quiere añadir una clase en el diagrama de clases, lo que implica añadir esa clase también en la implementación.

Se va a suponer que esa clase especifica la tipología de paso y se va a llamar "cTipoPaso", siguiendo la nomenglatura establecida.

Los pasos que se seguirían son:

- Se crearía un directorio más o carpeta en la estructura del dominio que se llamaría "cTipoPaso".

- Se añadiría de la misma forma a la estructura física de directorios que sirve como plantilla a la hora de definir un nuevo proyecto.

- Se define una nueva clase en el paquete "Domain", la clase "cTipoPaso" con sus atributos y procedimientos. Existen atributos que son obligatorios:
  - "int id": representa el identificador de una instancia de objeto de la clase.
  - "String description": representa la descripción de la instancia del objeto de la clase nueva.
  - "cList listContainers": lista que establece los contenedores de la instancia del objeto y que representa la relación con otros objetos.

- Se define una nueva clase en el paquete "Interfaz", la clase "cNodoTipoPaso". Como se puede apreciar en el nombre conserva el sufijo "TipoPaso", otra norma de nomenglatura. Esta clase define el nodo que se muestra en el árbol de objetos, en la parte derecha de la pantalla de modificación de elementos del proyecto. Debe ser una copia de los anteriores nodos cambiando:
  - El "import" de la clase, que debe ser "cTipoPaso"
  - El tipo del atributo llamado "objeto", que debería ser de tipo "cTipoPaso"
  - El tipo del parámetro del constructor, que debe ser de tipo "cTipoPaso"

```
package Interfaz;

/** imports */
import Domain.cTipoPaso;

/**
 *
 * @author charlie
 */
public class cNodoTipoPaso extends cNodo{

   public cTipoPaso objeto;
```

```
    /** Creates a new instance of cNodoTipoPaso */
    public cNodoTipoPaso() {
    }//constructor

    /** Creates a new instance of cNodoTipoPaso */
    public cNodoTipoPaso(cTipoPaso v) {
       objeto = v;
    }//constructor

    public void setConcrete() {

    }//setConcrete

    public String toString(){
       return objeto.toString();
    }// toString


}//class
```

- Se define una nueva clase en el paquete "Interfaz", la clase "iTipoPaso". El nombre vuelve a mantener la nomenclatura. En esta clase se define el formulario que mantiene la instancia del objeto, por tanto, se definirán los procedimientos que mantienen el objeto. En el código mostrado debajo, se muestran las partes comunes con otras clases que implementan formularios que cambian. Obviamente, si se requiriera un formulario más complejo, por necesidad podrían haber más procedimientos o tratar más eventos:

```
package Interfaz;

/** imports
 */
import Domain.cTipoPaso;
import java.lang.reflect.*;

/**
 *
 * @author  charlie
 */
public class iTipoPaso extends javax.swing.JPanel {

   cTipoPaso fuente;

   /** Creates new form iTipoPaso */
   public iTipoPaso(Object o) {

      Field fiobjeto = null;
      try {
       fiobjeto = o.getClass().getField("objeto");
       fuente = (cTipoPaso)fiobjeto.get(o);
      } catch (Exception e) {
         System.out.println ("Hay problemas con la lectura del objeto");
      }//Excepcion

      initComponents();
      loadComponent();
   }

   // <editor-fold defaultstate="collapsed" desc=" Generated Code ">
   private void initComponents() {

   }
   // </editor-fold>
```

```
private void jTxtDescriptionFocusLost(java.awt.event.FocusEvent evt) {
  fuente.description=jTxtDescription.getText();
}


// Variables declaration
private javax.swing.JLabel jLabel1;
private javax.swing.JLabel jLabel2;
private javax.swing.JLabel jLabel3;
private javax.swing.JPanel jPanAcciones;
private javax.swing.JPanel jPanSuperior;
private javax.swing.JSeparator jSeparator1;
private javax.swing.JTextField jTxtDescription;
private javax.swing.JTextField jTxtId;
private javax.swing.JLabel jlabSep2;
private javax.swing.JPanel jpanEtiquetas;
private javax.swing.JPanel jpanPrincipal;
// End of variables declaration

/** loads component values */
public void loadComponent()
{
   //cargar la lista
   jTxtId.setText(String.valueOf(fuente.id));
   jTxtDescription.setText(fuente.description);

}//load component


}//class
```

- Sólo queda cargar la clase en el árbol desde la clase que gestiona los elementos de un proyecto "iMain". Se realiza desde el procedimiento "initData" y tiene el siguiente aspecto:

```
int resultv=insertTreeFile(cNodoVariable.class,conf.pathDatosDomain+"/cTipoPaso/");
```

Con estos pasos, lo que se consigue, es introducir instancias de un objeto de una clase más y poderla gestionar (dar de alta nuevas instancias, eliminar una instancia o guardar los cambios de una instancia).

Queda claro que si se quisiera que en el formulario de mantenimiento de la clase "cPaso" apareciera los tipos de pasos, modelizados por la clase "cTipoPaso", el formulario que se tendría que modificar es el de "cPaso". Bastaría con añadir una lista desplegable donde se mostraran las diferentes instancias de la clase "cTipoPaso" y se definiera el evento de cambio, para que registrara el cambio de tipo en la instancia del paso.

8.3.2 Parametrización mediante metalenguaje

Visto en el anterior ejemplo que al seguir los pasos, el mantenimiento de la instancia automáticamente ya estaba programada, es debido a que en la implementación se han utilizado elementos de metalenguaje.

Los lenguajes de programación que ofrecen herramientas para desarrollar teniendo en cuenta el metalenguaje facilitan la implementación de funciones y procedimientos parametrizados.

A modo de ejemplo se puede ver en el procedimiento que inserta un nuevo nodo en el árbol:

```
private DefaultMutableTreeNode insertTreeObject(String objeto)
  {
    DefaultMutableTreeNode result = null;
    Object o = null;
    Object n = null;
    Field fiObject = null;

    //Nombre de la clase del nodo
    String stNodo = "Interfaz.cNodo"+objeto;

    //Nuevo objeto
    int id = -1;
    try {
        Class clNodo = Class.forName(stNodo);
        fiObject = clNodo.getField("objeto");

        //Obtener nuevo id (mediante el IdManager)
        id = idm.getNewId(fiObject.getType());

        //Asignación del id al atributo de la clase
        String claseO=fiObject.getType().getName();
        Class clObjeto = Class.forName(claseO);
        o = clObjeto.newInstance();
        Field fiId = clObjeto.getField("id");
        fiId.set(o, (Integer)id);

    } catch (Exception e){
        System.out.println("Hay problemas para leer el objeto del árbol.");
    }//catch


    //Asignar el objeto al nodo
    try {
        Class clDefinicion=Class.forName(stNodo);
        n = clDefinicion.newInstance();

        //Asignar el objeto xml al objeto del nodo
        fiObject.set(n,o);

      } catch (Exception e){
          System.out.println("Hay problemas para leer el objeto del árbol.");
      }//catch

    //Cargar objeto en el árbol
    result = new DefaultMutableTreeNode(n);

    return result;
  }//insertTreeObject
```

8.4 Visión general

El código del sistema se organizado de una forma muy similar a las capas. Esta organización se ha hecho con la intención de mantener una similitud entre la organización de la especificación y la del código.

Si se parte de la base que Java permite estructurar las clases por paquetes, se han usado estos paquetes para organizar como se indica en el párrafo anterior. La estructura principal del código es la siguiente:

- CompileManager (Paquete)
  - CompileManager
- DataManager (Paquete)
  - DataManager
- Domain (Paquete)
  - cComando
  - cComandoSimple
  - cVariable
  - cTipoVariable
  - cPaso
  - cScript
  - cProyecto
- IdManager (Paquete)
  - IdManager
- FileManager (Paquete)
  - FileManager
- Interfaz (Paquete)
  - iProject
  - iMain
  - cNodo
  - cNodoComandoSimple
  - cNodoVariable
  - cNodoTipoVariable
  - cNodoScript
  - cNodoPaso
  - iComandoSimple
  - iVariable
  - iTipoVariable
  - iScript
  - iPaso
- MySystem (Paquete)
  - cList
  - cHashId
  - cId
  - cContainer
  - ConfiguracionXML2Java

Como se puede observar se sigue una estructura bastante similar a la que ofrecía la especificación.

9. CONCLUSIONES

Habiendo hecho un análisis y desarrollo del sistema en todos los términos establecidos hay que hacer una valoración de lo propuesto y lo realizado.

A partir de las conclusiones se establece que objetivos iniciales se han llevado a cabo y, si la planificación determinada en las fases iniciales del proyecto se han satisfecho.

Se dejan abiertas, en esta conclusión, después de haber visto el sistema desde varias perspectivas, posibles mejoras y líneas de ampliación del sistema realizado.

También se realizará una valoración personal del proyecto.

9.1 Objetivos propuestos y objetivos alcanzados

Antes de empezar a valorar los objetivos alcanzados, se hace un breve recordatorio de cuales eran los objetivos iniciales del proyecto:

- minimizar el esfuerzo de construir y gestionar scripts para realizar experimentos.
- facilitar la portabilidad y la mantenibilidad del sistema.
- evitar el uso de sistemas gestores de bases de datos

Una vez hecho el recordatorio de los principales objetivos del sistema, se comenta a continuación los objetivos alcanzados en base a los objetivos iniciales:

- minimizar el esfuerzo de construir y gestionar scripts para realizar experimentos.

  Se ha construido una herramienta visual cuyo objetivo era ese. Permite definir, modificar experimentos y gestionar todos los elementos que lo conforman. Como objetivo principal se ha conseguido, aunque aún es demasiado pronto para establecer si realmente facilita y minimiza la construcción de experimentos, ya que no ha sido utilizado aún por ningún usuario.

- Facilitar la portabilidad y mantenibilidad del sistema

  El sistema construido se ha hecho a partir de esas premisas. Todas las tecnologías utilizadas en su construcción lo permiten. Para poder hacerlo portable cien por cien conviene actualizar la forma de describir las rutas dentro del código, aunque cambiando este punto el sistema es plenamente portable.
  Por otro lado el sistema es fácilmente mantenible puesto que está organizado por componentes y, el añadir funcionalidades nuevas implica, o bien añadir esas funcionalidades a esos componentes (en función de sus responsabilidades) o bien añadir componentes nuevos. Puesto que la mayoría de componentes que integran el sistema son independientes este punto se ha satisfecho

- Evitar el uso de sistemas gestores de bases de datos

  Habiendo establecido este punto como un requerimiento inicial del sistema, ya que el usuario así lo pidió, este objetivo se ha cumplido. Se ha de referir de todas maneras que el hecho de no utilizar un sistema gestor de bases de datos ha hecho más difícil el desarrollo del sistema, ya que se han tenido que planificar puntos (como por ejemplo la gestión de identificadores) que un sistema gestor de bases de datos ya tiene implementado.

9.2 Objetivos personales

Los objetivos que tracé al principio del proyecto fueron los siguientes:

- Desarrollar un sistema que fuera fácilmente mantenible y extensible.
- Explorar el desarrollo parametrizado utilizando funciones de metalenguaje.
- Experimentar con otras herramientas y técnicas desconocidas para mí.
- Resolver un problema diferente, dentro de un ámbito diferente al de mi vida profesional.

Después de revisarlos quizá olvidé también que uno de los objetivos personales realizando este proyecto era cerrar un ciclo de mi vida.

De todas maneras, pienso que el resto de objetivos se han cumplido en un gran porcentaje ya que:

- El sistema desarrollado es un sistema fácil de mantener y extender por su modularidad.

- He intentado realizar funciones y procedimientos parametrizados utilizando las clases y métodos de persistencias que me ofrecía el lenguaje de programación. Sean o no sean los métodos más adecuados de parametrización, me han ayudado a mi vida laboral para darme cuenta qué se puede generalizar y qué no.

- El uso de las tecnologías en un entorno laboral no las escoges, si no que vienen impuestas por el cliente. Eso condiciona a usar tecnologías que pueden ser conocidas o desconocidas, nuevas u obsoletas, pero no las escoges. En cambio, el proyecto me ha brindado la posibilidad de inspeccionar tecnologías como Linux, Java y XML que no veo normalmente en la vida laboral.

- Mis tareas están muy encapsuladas al sector laboral al que me dedico (básicamente gestión). El hecho de plantear un sistema sobre términos diferentes a la gestión clásica es enriquecedor.

Por último, decir que en los objetivos no entraba el reto de compaginar un trabajo, el proyecto y mi propia vida personal, que ha sufrido muchos cambios. Quizá el objetivo último que me marqué fue realmente acabar lo que había empezado.

9.3 Líneas abiertas

El sistema realiza unas funciones correctas dentro de los objetivos marcados, aunque con un poco más de tiempo se podrían haber realizado más tareas como:

- Extender el sistema para que reconozca más lenguajes, no sólo el tcshell. Se podría realizar incluyendo dos clases más, la representación de un comando iterativo y otro condicional. En la clase controladora de montaje de código, tener en cuenta entonces el lenguaje.

- Permitir ejecutar partes del código, por ejemplo ejecutar un paso, un script o, incluso, un comando determinado.

- Realizar una compilación de código real, estableciendo que partes del código no son correctas. Esto podría incluso llevar otro proyecto, ya que supone implementar un compilador

- Hacer que el sistema sea distribuido. Sería una manera de permitir compartir los experimentos con otros investigadores.

APENDICE A

En el siguiente apéndice se muestra un sencillo manual de usuario, con el que poder realizar la mayoría de operaciones del sistema, aunque la interfaz gráfica es sencilla y no ofrece complicación.

A.1 Instalación

Para instalar el programa se necesita:

- JRE 1.5 o superior
- El archivo "gexp.jar"
- La carpeta de librerías "lib", adjuntada con el archivo "gexp.jar".
- El classpath de java bien configurado

A.2 Manual

En esta sección se describe el funcionamiento del sistema a nivel de usuario. Se partirá de que el sistema visualiza dos pantallas principales, la de pantalla de gestión de proyectos y la pantalla de gestión de elementos del proyecto.

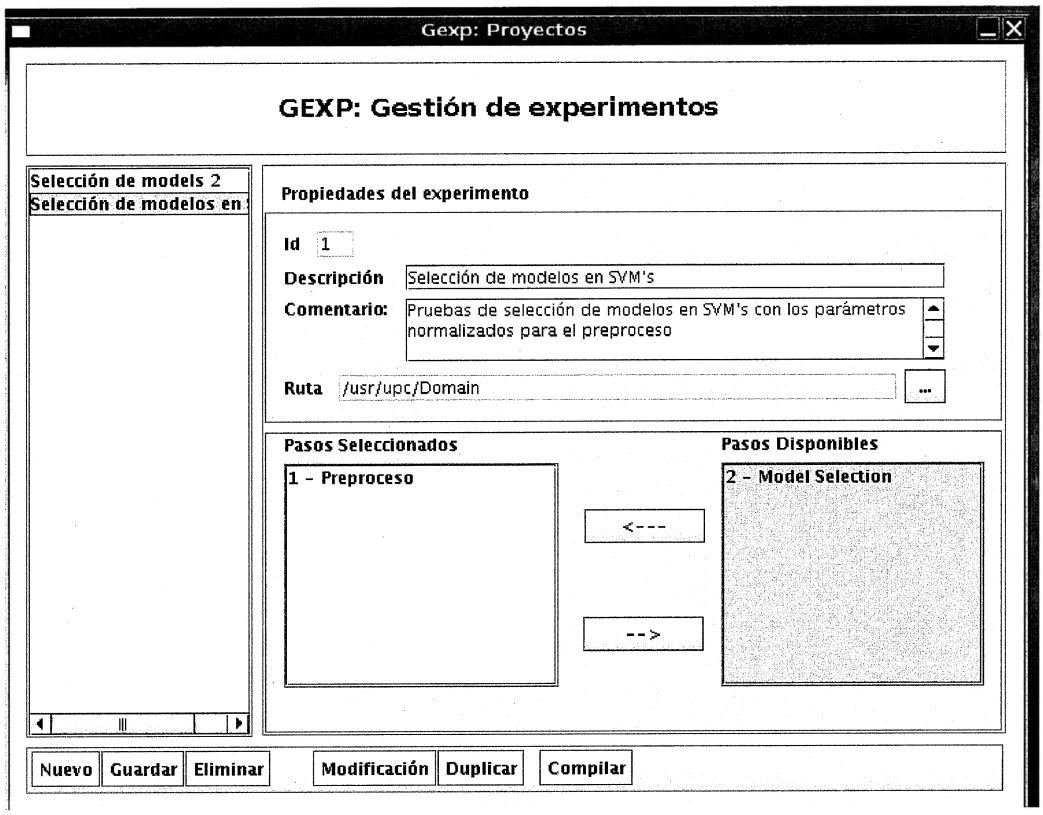
A.2.1 Pantalla de gestión de proyectos

Mediante esta pantalla se realizan las siguientes tareas:

- Se define un proyecto, que representa un experimento.

- Se modifican las características propias de un proyecto.
- Se selecciona dónde va a estar ubicado el proyecto.
- Se seleccionan los pasos de los que está formado el proyecto.
- Se puede duplicar un proyecto.
- Se puede generar el código de un proyecto.

La pantalla inicial que muestra el sistema es parecida a la siguiente:



En la parte izquierda de la pantalla se muestra una lista con todos los proyectos disponibles. En la parte derecha, se muestra las características del proyecto seleccionado. En la parte inferior hay una barra de herramientas que establece las funcionalidades a realizar con el proyecto.

Para ver las características de un proyecto, basta con pulsar sobre el proyecto en la lista de proyectos, situada a la izquierda. Al seleccionar el proyecto, se cargará en la parte derecha, las características de ese proyecto.

Las propiedades se pueden modificar directamente. Por ejemplo, si se quisiera cambiar el comentario del proyecto, habría que situarse en el cuadro de texto del "Comentario" y cambiar el comentario.

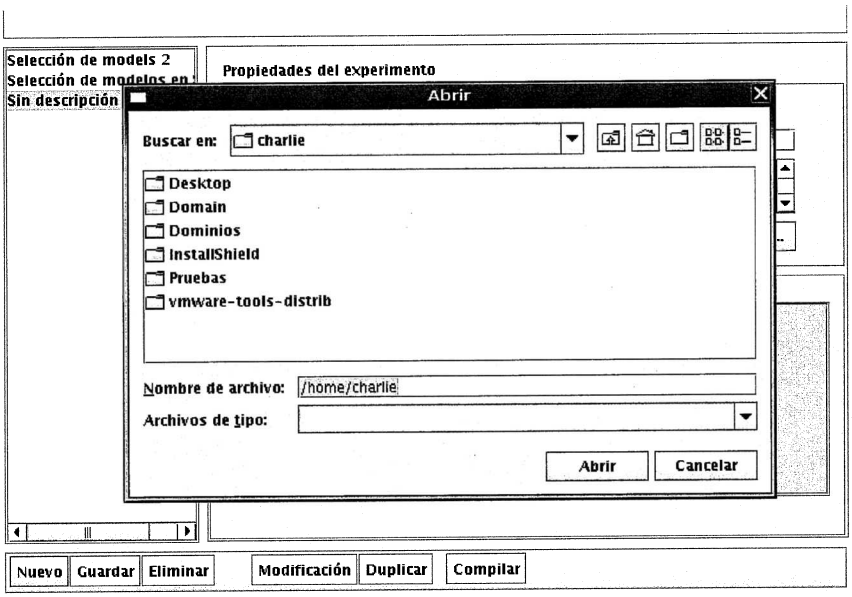
Para cambiar la ruta donde están ubicados los elementos del proyecto, se pulsará sobre el botón "...", situado en la parte derecha de la propiedad "Ruta".

Los pasos seleccionados también se pueden cambiar. Para añadir un paso al proyecto se deberá pulsar el botón "←-". Por el contrario si se desea quitar un paso del proyecto, se deberá pulsar sobre el botón "-→".

Es importante remarcar que los cambios realizados sobre las propiedades de un proyecto no tendrán efecto hasta que se pulse sobre el botón "Guardar".

Los botones de la barra de botones en la parte inferior realizan las siguietnes funciones:

- Nuevo: permite definir un nuevo proyecto. Nada más pulsar el botón aparecerá en pantalla un cuadro de diálogo para establecer donde se guardarán los elementos conformantes del proyecto. Si en la carpeta seleccionada ya existe una carpeta "Domain", que es la que guarda los elementos de un proyecto, cargará automáticamente en la lista de "Pasos Disponibles", los pasos disponibles que hubiera para ese proyecto.

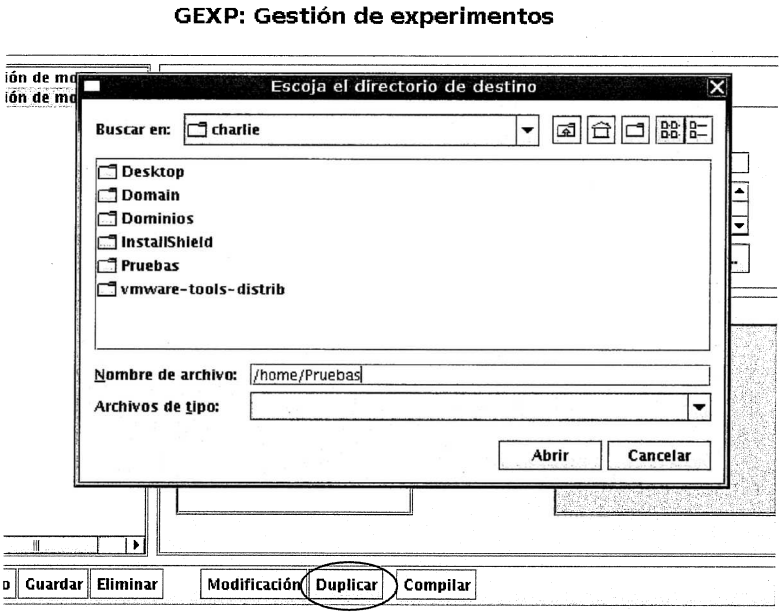


Para acabar la definición del nuevo proyecto se deben rellenar las propiedades restantes del proyecto y para registrar el proyecto, pulsar sobre el botón "Guardar".

- Guardar: permite registrar los cambios que ha sufrido el proyecto seleccionado en la lista de proyectos, tanto cuando se define uno nuevo como cuando se modifica uno existente.

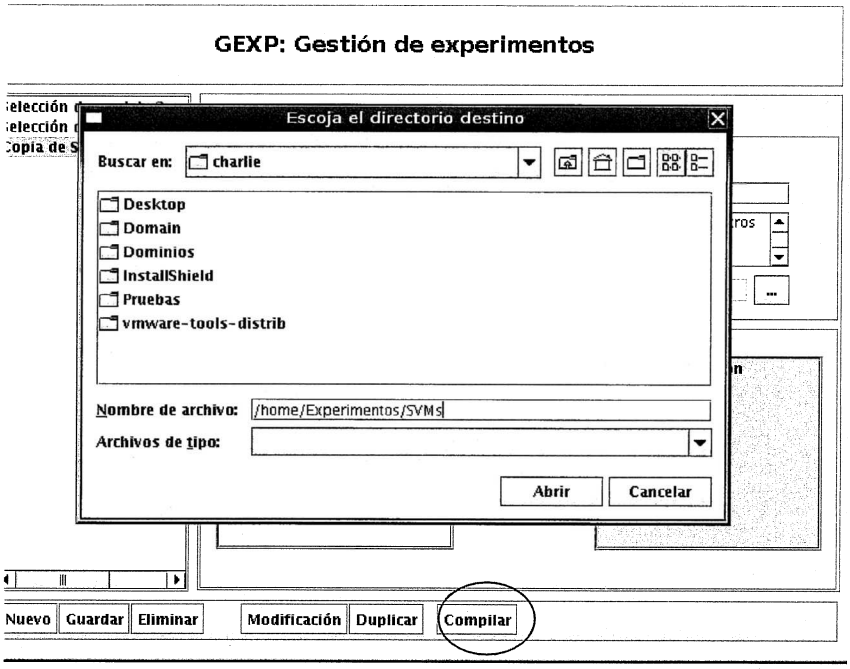
- Eliminar: elimina un proyecto del sistema. El eliminar un proyecto del sistema, también implica eliminar, físicamente, toda la estructura física de carpetas del disco.

- Duplicar: mediante este botón se puede duplicar un proyecto ya existente. Esta opción, replicará la estructura física que contiene los elementos de los que podrá estar formado el proyecto.



La copia resultante ya queda registrada en el sistema (no hace falta pulsar sobre el botón guardar a no ser que se realicen cambios en la copia) y tiene la descripción "Copia de <descripción del proyecto copiado>".

- Compilar: genera toda la estructura de código definida a partir de los pasos seleccionados y la deja en una carpeta que el usuario escoge mediante un cuadro de diálogo.

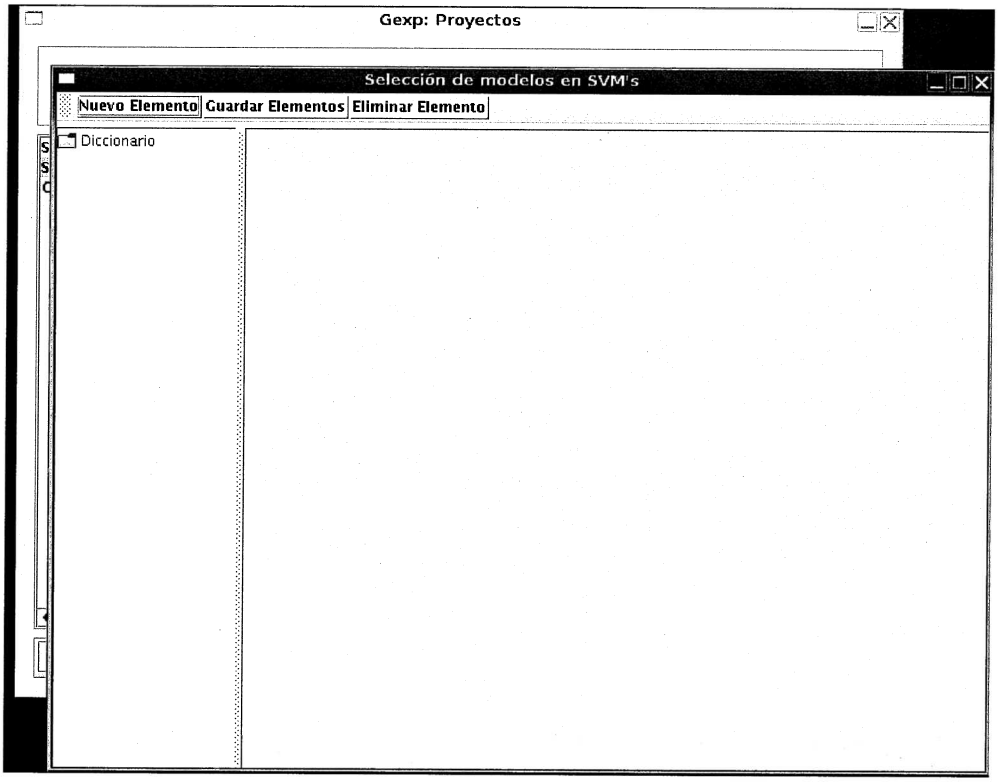


- Modificación: el botón modificación permite modificar y gestionar todos los elementos que forman un proyecto, incluídos los pasos. En el siguiente apartado se ve con más detalle.

A.2.2 Pantalla de gestión de elementos de un proyecto

Es la pantalla que aparece al pulsar sobre el botón "Modificar" de la pantalla de gestión de proyectos.

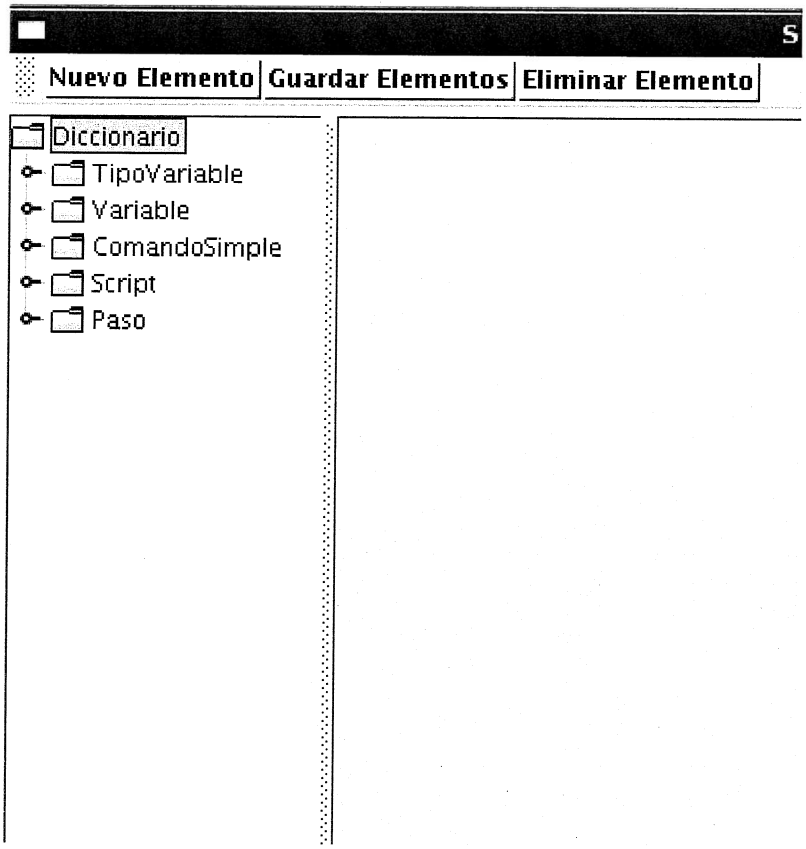
Permite realizar el mantenimiento de todos los elementos que conforman un proyecto. La pantalla inicial tiene el siguiente aspecto:



En la imagen se puede apreciar la estructura de esta pantalla:

- En la barra de título, el campo de descripción del proyecto.
- En la parte superior, una barra de botones para poder gestionar los elementos.
- En la parte izquierda, un árbol que tiene como raíz la palabra "Diccionario".
- En la parte derecha, un formulario vacío, donde se cargarán los formularios de los elementos a gestionar.

A continuación se muestra una figura, con los elementos del árbol desplegados.

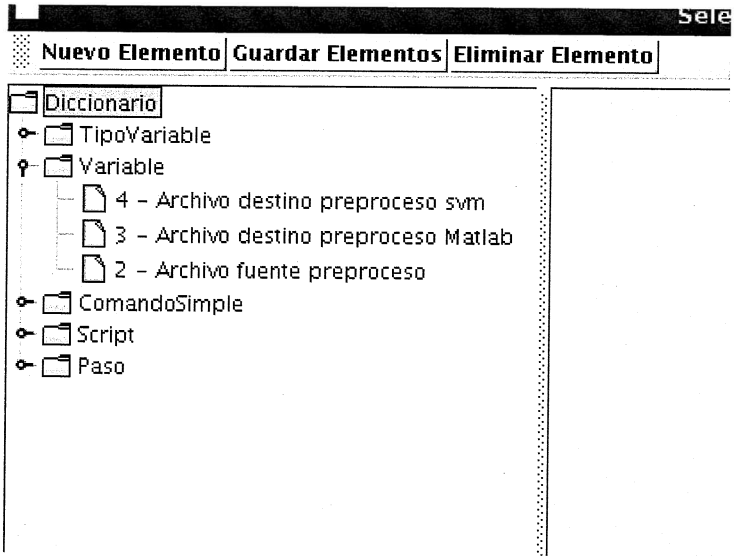


Los elementos conformantes de un proyecto se muestrán como nodos dentro del árbol. Los nodos siempre se muestran en el orden indicado en la figura. Además están ordenados de manera intuitiva en función de las relaciones entre ellos:

- Un TipoVariable aparece en una Variable.
- Las Variables aparecen en los ComandosSimples
- Los ComandosSimples se seleccionan en los Scripts
- Los Scripts se escogen en los Pasos.

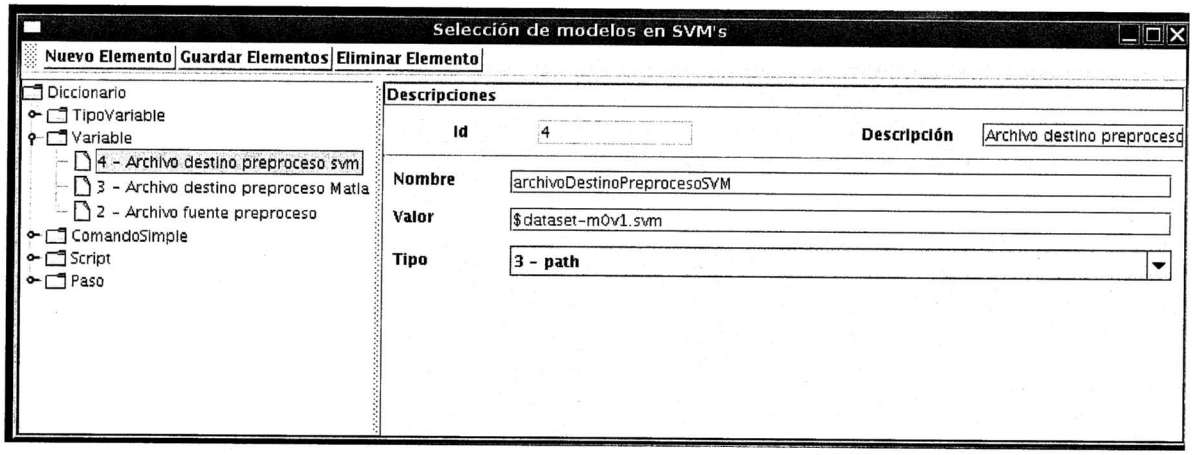
La organización, sigue, por tanto, un orden lógico de qué es lo que se tiene que ir definiendo.

Se tomará un elemento como ejemplo para mostrar su gestión, ya que todos se gestionan de una manera homogénea. Se parte pues, eligiéndolo de manera arbitraria el elemento "Variable". Al desplegarlo obtenemos una imagen parecida a:



Como se aprecia en la imagen al desplegar la "Variable" dentro del árbol se muestran todos los elementos definidos. Los elementos muestran, como descripción del nodo, el identificador más la descripción del objeto.

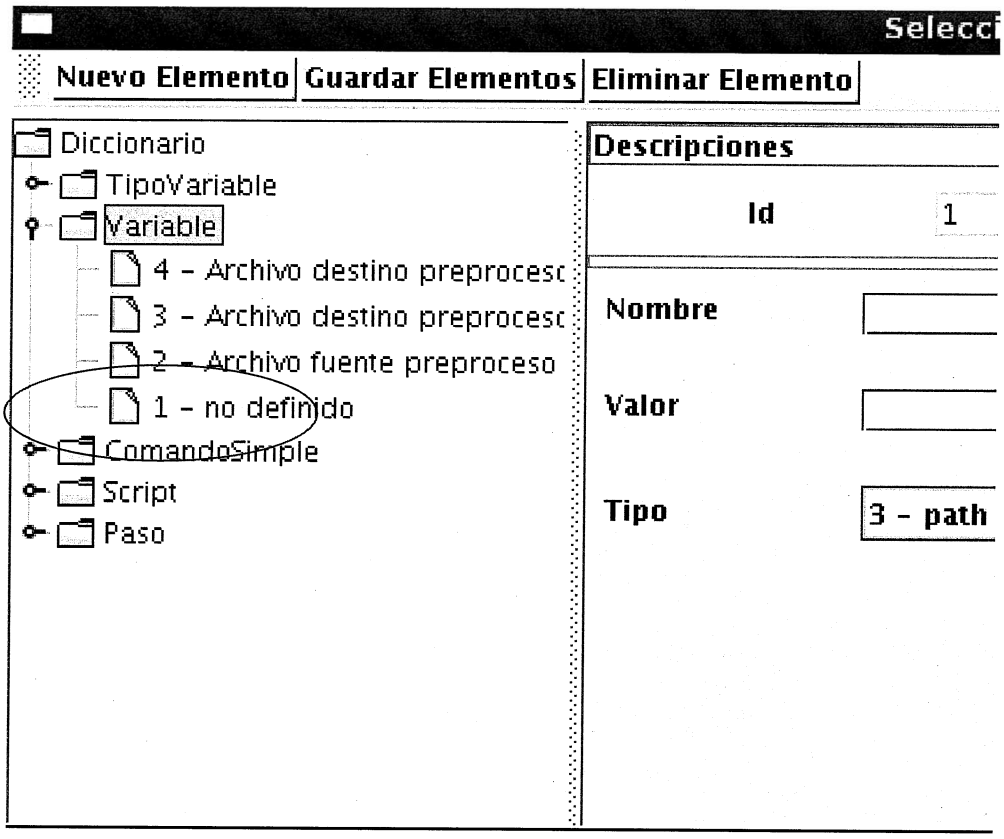
Para visualizar uno de los elementos se pulsará sobre el elemento. Al pulsar sobre el elemento, se cargará en la parte derecha, en el lugar reservado para los formularios, el formulario correspondiente.



En la parte superior del formulario siempre aparecerá, para cualquier elemento, el identificador y la descripción. El identificador es un campo fijo y no se puede modificar, ni asignar, ni eliminar.

Se pueden realizar ahora tres operaciones sobre este elemento:

- Modificar: para modificar un elemento seleccionar el campo de sus propiedades en el formulario y modificar.
- Eliminar: para eliminar un elemento, seleccionar el elemento en el árbol y pulsar sobre el botón "Eliminar Elemento".
- Definir uno nuevo: para definir un nuevo elemento, seleccionar en el árbol el padre del elemento del cual se quiere dar de alta (en este caso "Variable") y pulsar sobre el botón "Nuevo Elemento".

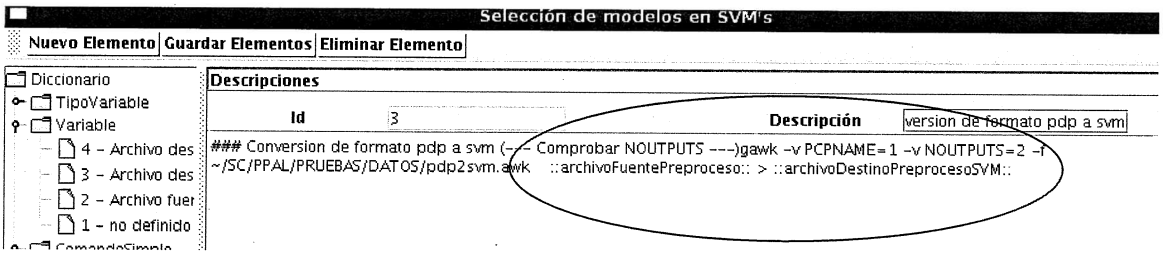


El nuevo elemento aparece en el árbol con un identificador nuevo y la descripción "no definido".

Todos los elementos se comportan de la misma manera a la descrita. Algunos elementos, sin embargo, tienen particularidades.

Un caso muy claro de estas particularidades es el elemento "ComandoSimple". La manera que tiene este elemento de referenciar las variables es diferente a la de los otros elementos.

Un ejemplo de referencia de variable se ve en la figura siguiente:



Las variables se referencian con la nomenglatura "::<nombreVariable>::".

BIBLIOGRAFIA

- [Brown, 1999] Brown, A. (1999). Constructing Superior Software, cap´itulo Building Systems from Pieces: Principles and Practice of Component-based Software Engineering. Macmillan Technical Publishing.

- [Ojeda, 2003] Charte Ojeda, Francisco (2003). GNU/Linux. Anaya Multimedia.

- [McLaughlin, 2000] McLaughlin, Brett (2000). Java and XML. O'Reilly & Associates, Inc.

- [Eckstein, Loy, Wood, 1998] Eckstein, Robert y otros (1998). Java Swing. O'Reilly & Associates, Inc.

- [Purcell, 1997] Purcell, J. (1997). Linux Complete Command Reference. Red Hat Software, Inc.

- [Apostol, 1967] Apostol, Tom M. (1967). Calculus. Xerox Corporation.

- [Guyon et alt, 2002] Guyon, Isabelle y otros (2002). Gene Selection for Cancer Classification using Support Vector Machines. Kluwer Academic Publishers.

- [Iribarne, 2003] Iribarne Martínez, Luis F. (2003). Desarrollo de Software basado en componentes COTS. Capítulo 1. http://www.cotstrader.com/thesis/chapter-1.pdf

- [W3C, 2003] W3C (2003). Extensible Markup Language. http://www.w3.org/XML/
- [Java Sun, 2006] Java Sun Microsystems (1997-2007). http://java.sun.com/
- [Walnes, 2006] Walnes, Joe (2003-2006). xStream, Two Minute Tutorial. http://xstream.codehaus.org/tutorial.html
- [Ubuntu, 2006] Ubuntu-es (2006). Documentación. http://www.ubuntu-es.org/index.php?q=documentacion