4 Estructura del Programa
SISTEMA DE CONTROL DIFUSO (SCD)

En el presente capítulo se expone una breve noción del ámbito de programación Visual Basic 6.0 © [Brian99] como herramienta de desarrollo de aplicaciones informáticas bajo plataforma Windows. La información se estructura básicamente entorno al diseño de la *interfaz de usuario* compuesta por *formularios* (ventanas), *controles* (elementos visuales contenidos en un formulario) y el *código* que permite establecer la secuencia de funcionamiento de los mismos por medio de *módulos de código*. Una vez introducidos en el entorno de programación se expondrá la estructuración del programa SCD como una aplicación MDI *Multi-Document-Interface.* Se enumerarán aquellos tipos de estructuras, enumeraciones, variables, constantes, procedimientos, funciones, módulos de código, formularios, ficheros y demás elementos relevantes que conforman a SCD.

4.1 Introducción al Entorno de Programación - Visual Basic 6.0 ©

El objetivo al elaborar este apartado ha sido el de introducir al lector en el entorno de programación Visual Basic 6.0, es por ello que no se profundizará en muchos aspectos de Visual Basic pues creemos no es la finalidad de este documento pero si mencionar que esto y todo lo referente a Visual Basic 6.0 podrá encontrarlo en [Brian99] de forma clara, concisa y fiable.

A la hora de desarrollar una aplicación informática es necesario una herramienta de desarrollo que permita al programador generar el software que necesita. Para la elaboración del presente trabajo se ha optado por el producto Visual Basic como entorno de programación para la construcción de una aplicación sólida en un espacio (relativamente) corto de tiempo; Visual Basic a menudo se considera como herramienta de Desarrollo Rápido de Aplicaciones (RAD, Rapid Application Revelopment). El producto Visual Basic de Microsoft podría definirse como un *sistema de programación* utilizado para escribir programas basados en Windows, y crear herramientas de productividad personalizadas.

Debido a la naturaleza estructurada de la computación, resulta vital diseñar los programas *antes* de empezar a codificarlos, es por ello que se podría establecer unos pasos genéricos claves para la creación de un programa informático:

- Planificar las tareas del programa: *¿cómo debe funcionar?.*
- Diseñar la Interfaz de Usuario: *¿qué aspecto debe tener?.*

- Generar Código del programa: Implementar los dos pasos anteriores.
- Probar y Depurar el programa.
- Documentar y Distribuir el programa.

Un concepto clave de Visual Basic es la capacidad de crear y utilizar componentes u *objetos.* Un tipo de objeto es un *control* el cual va a permitir agregar funcionalidad a los programas sin tener que entrar en los detalles del modo de funcionamiento de dichas características (encapsulamiento). Los *objetos* tienen *propiedades* que sirven para definir su apariencia, *métodos* que les permiten ejecutar tareas y *eventos* que les permiten responder a acciones de usuario.

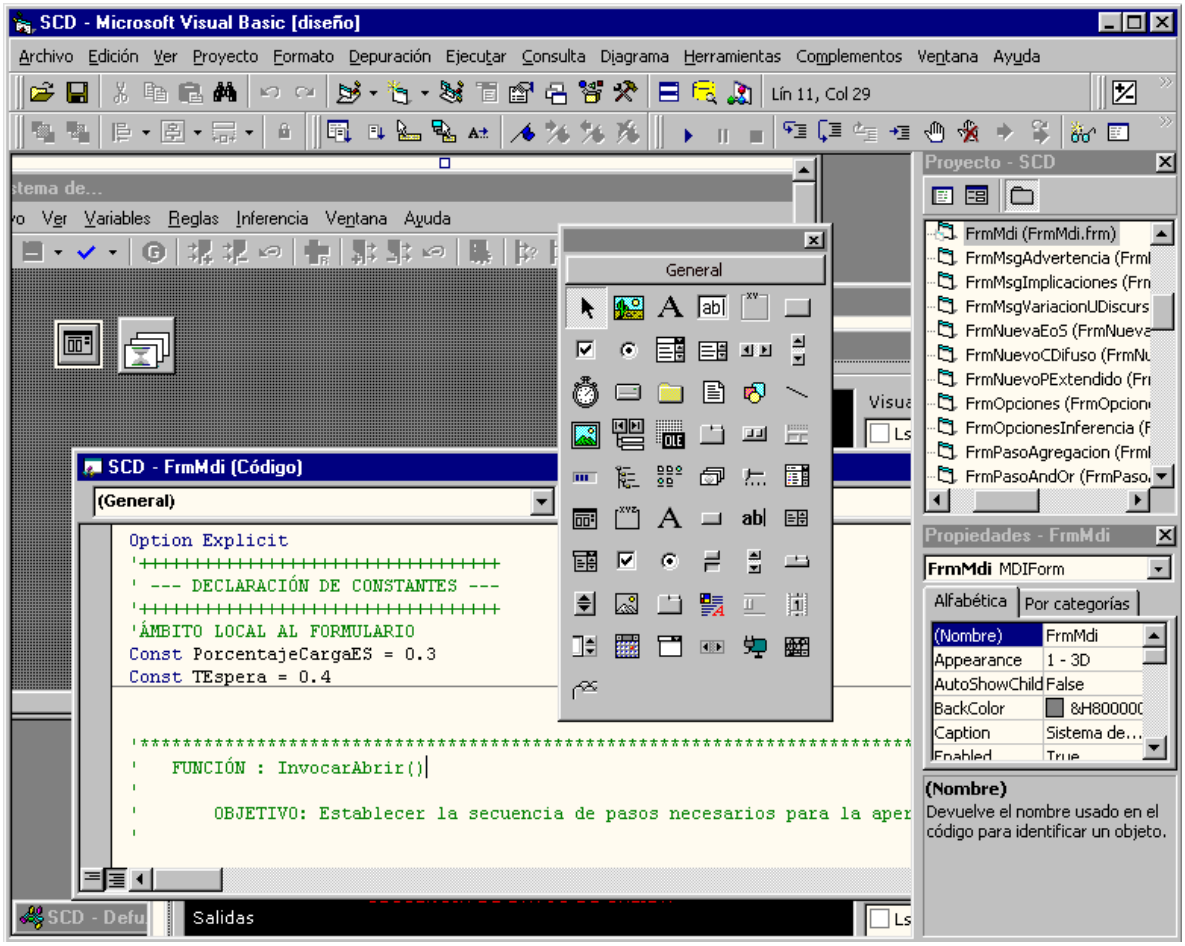

Figura 4.1: Entorno de Desarrollo Integrado (IDE) de Visual Basic 6.0.

Visual Basic ofrece varios tipos de *proyectos*, cada uno ofrece al usuario una determinada funcionalidad y características. El proyecto que más se utiliza y sobre el cual se ha desarrollado la presente aplicación es el tipo *EXE estándar*, proyecto que se emplea para crear un programa estándar de Windows (archivo EXE). En la Figura 4.1 se presenta el *Entorno de Desarrollo Integrado (IDE, Integrated Development Environment)* de Visual

Basic, que constituye la interfaz del propio Visual Basic. Tomándose como partida el proyecto, básicamente se pueden distinguir dos bloques de trabajo a la hora de desarrollar una aplicación:

- Interfaz del Usuario: Diseño de Formularios y Controles.
- Generación de Código: Módulos de Código.

Además de los dos bloques principales de trabajo comentados anteriormente se mencionan los siguientes elementos utilizados en el desarrollo de la aplicación:

- *Documentos Relacionados*: Esta carpeta dentro del explorador de proyectos (Figura 4.2 ventana derecha) va a contener todos aquellos elementos implicados en el proyecto de la aplicación que no sean ni elementos de la interfaz de usuario (formularios) ni módulos de código y algunos otros como módulos de clase... Dos elementos utilizados en el desarrollo de SCD son los *recursos* y los archivos identificados como *relacionados* con la aplicación.

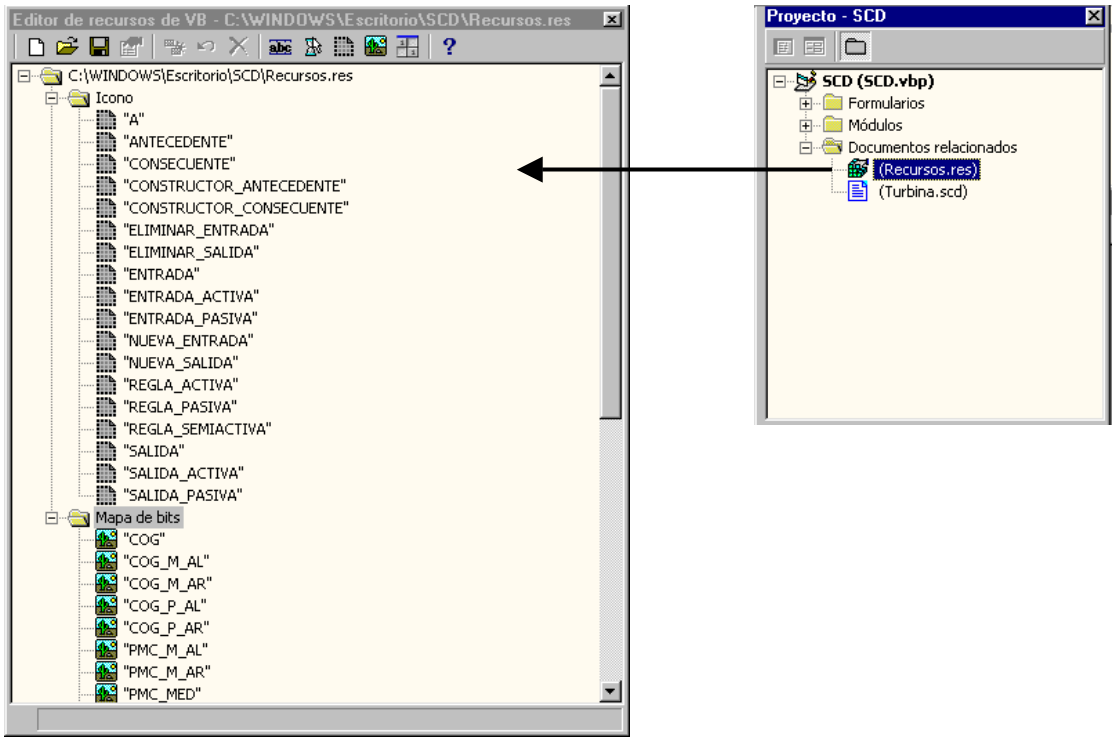

Figura 4.2: Editor de recursos y ventana explorador de proyectos.

Los *Recursos* (archivos de extensión .RES) van a permitir almacenar ficheros de iconos, mapas de bits, cadenas de texto o cualquier otra información de forma centralizada en un solo fichero. Además, permite establecer unos

identificadores a cada uno de estos datos introducidos a través del *Editor de recursos* (Figura 4.2 ventana izquierda. La utilidad que ofrece es la de permitir modificar contenidos en tiempo de ejecución, es decir, sin necesidad de recompilar nosotros podríamos tener almacenado en este archivo de recursos todas las cadenas de texto referentes por ejemplo al menú de nuestra aplicación pero no sólo en un idioma sino en varios y en función del valor de una opción que seleccionase el usuario al ejecutar la aplicación pudiera seleccionar el idioma en el que desea trabajar. En general, su utilidad es la de agrupar recursos necesarios para la aplicación en un solo fichero de forma que centraliza los datos.

Con respecto a los archivos identificados como *relacionados* van a ser aquellos ficheros que el programador considere necesarios para el funcionamiento de la aplicación sin ser en sí mismos de la propia aplicación. Por ejemplo, la distribución de una aplicación que desea que vaya siempre acompañada de algunos ejemplos de ilustración. Estos ficheros de ejemplos podrían ser considerados como *relacionados* de forma que se garantiza que en el proceso de empaquetado, dichos ficheros serán instalados junto a la aplicación.

- *Asistente de empaquetado y distribución*: A través de este complemento ofrecido en el entorno de Visual Basic se va a poder elaborar el paquete de instalación que permitirá instalar nuestra aplicación. Como puede observarse en la Figura 4.3 varias son las posibilidades que ofrece Visual a la hora empaquetar una aplicación. Si nuestro objetivo es el de generar un programa de instalación se opta por la primera opción *Empaquetar*. La ventanas que se suceden a la ya mencionada establecerán una serie de cuestiones que permitirán establecer los parámetros de instalación.

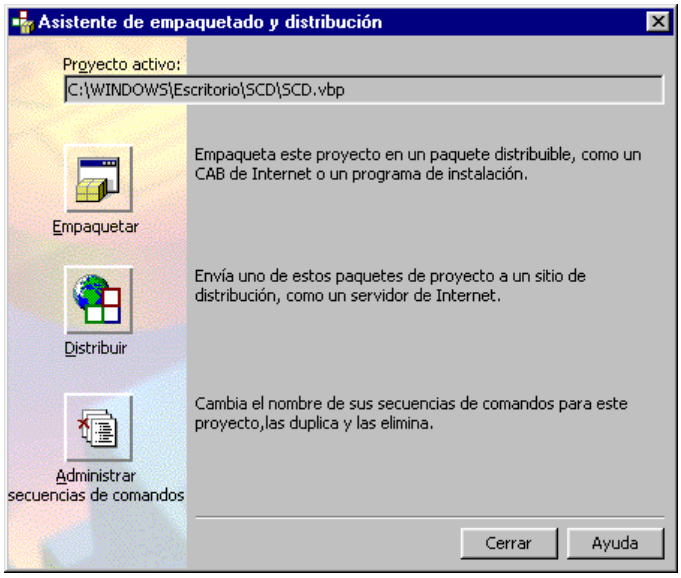

Figura 4.3: Ventana de Asistente de empaquetado y distribución.

- *Examinador de Objetos*: El código de una aplicación de Visual Basic está organizado de forma jerárquica. Una aplicación típica está compuesta por uno o más módulos: un módulo de formulario por cada formulario de la aplicación, módulos estándares opcionales para el código compartido. Cada módulo contiene uno o más procedimientos que contienen código: procedimientos de evento, procedimientos *Sub* o *Function*, propiedades *Property*, declaraciones de constantes, variables, etc. A través del *Examinador de objetos* (Figura 4.4) del *Entorno de Desarrollo Integrado (IDE)* de Visual Basic se puede ver la estructura del proyecto, es decir, agrupar toda la información (procedimientos, funciones, enumeraciones, constantes...) de cada uno de los elementos (formularios y módulos de código) que forman la aplicación en una ventana común.

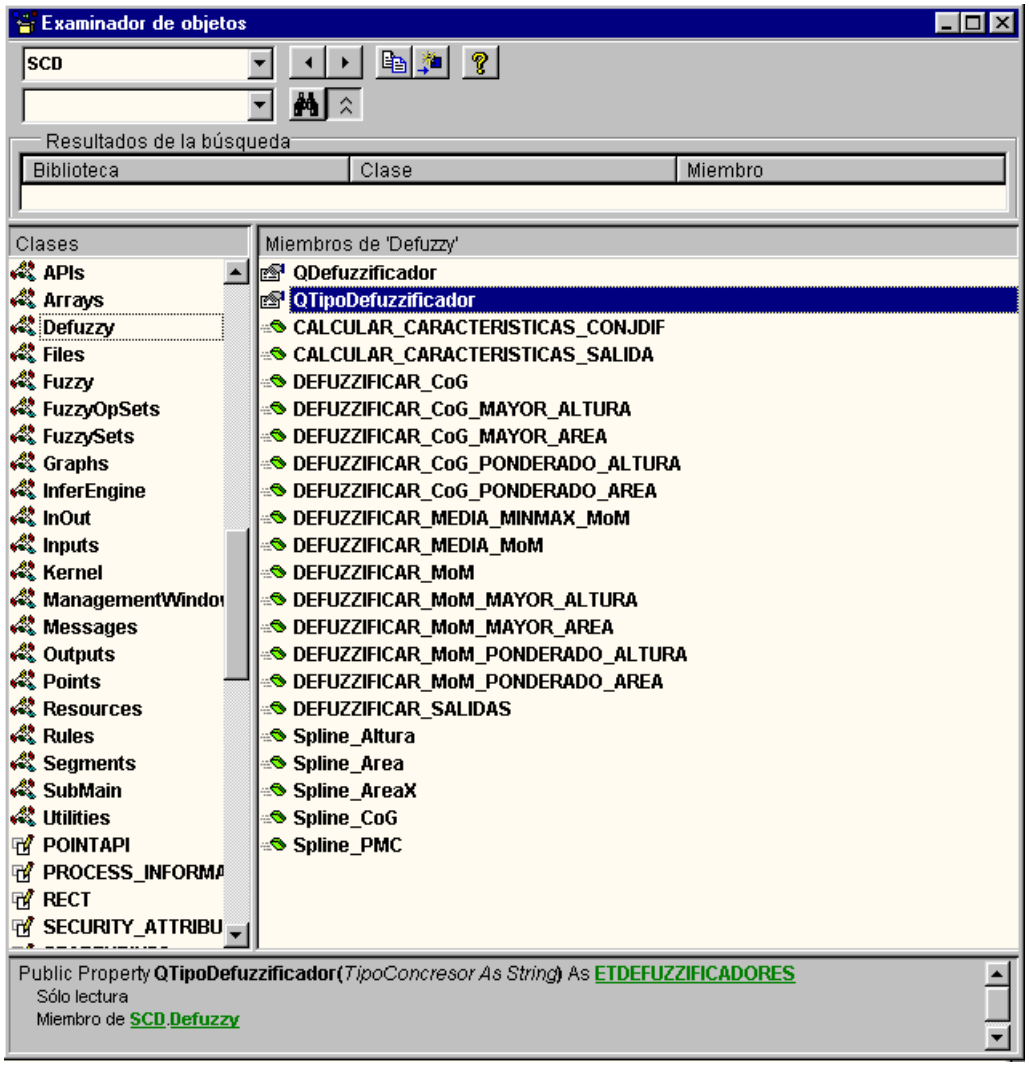

Figura 4.4: Ventana Examinador de objetos.

El control lista desplegable de la parte superior izquierda en el examinador de objetos permite seleccionar el proyecto o biblioteca de la cual se

desee mostrar los elementos. Una vez seleccionado nuestro proyecto, el examinador de objetos mostrará todos los elementos que contiene en la parte izquierda. A su vez, al seleccionar alguno de estos elementos se mostrará en la parte derecha todos los miembros que contiene. Aquellos elementos que hayan sido creados por el programador estarán resaltados en negrita diferenciándolos de los propios de cada control. En la parte inferior se muestra la información del miembro seleccionado. Así. si por ejemplo se selecciona un procedimiento este recuadro mostrará la cabecera del mismo.

Antes de comenzar comentando los *tipos* o *estructuras* más importantes de la aplicación y sobre la cual se sustenta la manipulación de información de la aplicación se comentan los tipos de datos definidos en Visual Basic (intrínsecos) utilizados y sobre los cuales se han formado los necesarios para la aplicación. Estos tipos están declarados en la Tabla 4.1.

TIPO INTRÍSECO	**DESCRIPCIÓN**
String	Hay dos clases de cadenas: cadenas de longitud variable y cadenas de longitud fija (string*). Las cadenas de longitud variable pueden contener hasta 2.000 millones de caracteres (2^{31}). Las cadenas de longitud fija que pueden contener de 1 a 64 KB (2^{16}) caracteres. El carácter de declaración de tipo para string es el signo de dólar (**$**).
Integer	Las variables Integer se almacenan como números de 16 bits (2 bytes) con valores que van de -32.768 a 32.767. El carácter de declaración de tipo para el tipo Integer es el signo de porcentaje (**%**).
Double	Las variables dobles (punto flotante de doble precisión) se almacenan como números IEEE de coma flotante de 64 bits (8 bytes) con valores de -1,79769313486232E308 a -4,94065645841247E-324 para valores negativos y de 4,94065645841247E-324 a 1,79769313486232E308 para valores positivos. El carácter de declaración de tipo para Double es el signo de número (**#**).
Single	Las variables Single (punto flotante de precisión simple) se almacenan como números IEEE de coma flotante de 32 bits (4 bytes) con valores que van de -3,402823E38 a -1,401298E-45 para valores negativos y de 1,401298E-45 a 3,402823E38 para valores positivos. El carácter de declaración de tipo para Single es el signo de exclamación (**!**).
Long	Las variables Long (enteros largos) se almacenan como números con signo de 32 bits (4 bytes) con un valor comprendido entre - 2.147.483.648 y 2.147.483.647. El carácter de declaración de tipo para Long es el signo **&**.
Boolean	Las variables tipo Boolean se almacenan como números de 16 bits (2 bytes), pero sólo pueden ser **True** o **False**.

Tabla 4.1: Tipos de datos definidos en Visual Basic.

4.2 Estructura General de SCD

La estructuración del programa SCD como una aplicación MDI *Multi-Document-Interface* se compone de una serie tipos de estructuras, enumeraciones, variables, constantes, procedimientos, funciones, módulos de código, formularios, ficheros y demás elementos que permiten gestionar el funcionamiento de la aplicación. Estos elementos son comentados a continuación.

4.2.1 Constantes

Las constantes, al igual que las variables, almacenan valores pero, como su nombre indica, estos valores permanecen constantes durante la ejecución de la aplicación. La utilización de constantes puede hacer más legible el código ya que proporciona nombres significativos en vez de números. Las constantes más relevantes de la aplicación están en la Tabla 4.2.

CONSTANTE	VALOR	DESCRIPCIÓN
SIGLA_ (SCD, SCD_INGLES…)	Cadena caracteres	Almacenan abreviaturas.
EXTENSION_ (_ARCHIVO_INICIALIZACION, _ARCHIVOS_COPIAS_SEGURIDAD…)	Cadena caracteres	Almacenan las extensiones de los ficheros generados por la aplicación.
NOMBRE_ (_APLICACIÓN_ESPAÑOL, _DESCRIPCIÓN…)	Cadena caracteres	Almacenan expresiones de texto.
SECCION_ (_GENERAL, _ENTRADA…)	Cadena caracteres	Almacenan los tipos de secciones de la estructura de ficheros SCD.
CLAVE_ (_ESTADO, _NOMBRE…)	Cadena caracteres	Almacenan los tipos de claves de la estructura de ficheros SCD.
NUMDEC	4	Número de decimales utilizados en los procesos de cálculo.
EPSILON	0.001	Valor mínimo de diferencia entre puntos de las funciones de pertenencia de las etiquetas lingüísticas.
NUM_ESTADOS	20	Número de valores en la secuencia de datos.
ICO_ (_ANTECEDENTE, _ENTRADA...)	Cadena caracteres	Almacenan el identificador de iconos en el archivo de recursos.
MB_SN_ (_F_FRANK, _MÁXIMO...)	Cadena caracteres	Almacenan el identificador de imágenes de expresiones de s-normas en el archivo de recursos.
MB_TN_(_MINIMO, PRODUCTO_ACOTADO...)	Cadena caracteres	Almacenan el identificador de imágenes de expresiones de t-normas en el archivo de recursos.
MB_D_ (_COG, _COG_M_AL…)	Cadena caracteres	Almacenan el identificador de imágenes de expresiones de Defuzzificadores en el archivo de recursos.
ALTO_ (_FA, _FB...)	Numérico	Almacenan la altura de los formularios.
ANCHO_ (_FA, FB...)	Numérico	Almacenan la anchura de los formularios.

Tabla 4.2: Algunas Constantes empleadas.

4.2.2 Enumeraciones

Las *enumeraciones* proporcionan una manera cómoda de trabajar con conjuntos de constantes relacionadas y de asociar valores de constantes con nombres. Algunas de las empleadas aparecen en la Tabla 4.3.

ENUMERACIÓN	DESCRIPCIÓN
ET_TNORMAS_SNORMAS	Indica el tipo de t-norma o s-norma seleccionada. Enumeración declarada con 9 tipos de t-normas y 8 de s-normas.
ET_IMPLICACIONES	Indica el tipo de Implicación seleccionada. Enumeración declarada con 3 Implicaciones Fuertes, 3 I. Residuales, 1 RS-Implicación, 1 QM-Implicación o de la Mecánica Cuántica y 3 Implicaciones con t-normas.
ET_CONJDIF	Indica el tipo de conjunto difuso o etiqueta lingüística y son 7: *Singleton, Intervalo, L, Triángulo, Trapecio, Gamma* y *Trapecio Extendido*.
ET_DEFUZZIFICADORES	Indica el tipo de Concresor o Defuzzificador. Hay declarados 6 tipos del *Centro de Gravedad CoG,* 9 del *Punto de Máximo Criterio PMC*, y 2 del *Centro de Área CoA*.
ET_DIFUSORES	Indica el tipo de difusor, hay declarados 5 tipos: *Singleton, Intervalo, Triángulo, Trapecio* y *Trapecio Extendido*.
ET_ESTADO	Indica los tres estados posibles de una regla en la base de conocimiento: *Activa, Semiactiva* y *Pasiva.*

Tabla 4.3: Algunas Enumeraciones empleadas.

4.2.3 Tipos

Cuando se programa, se almacenan datos y se manipulan mediante una serie de instrucciones. Los datos y los contenedores (variables, matrices,...) en los que se almacenan los datos constituyen la materia prima de la programación. Los tipos de datos determinan la manera en que se almacenan los datos y la forma en que se pueden utilizar tales datos. Los tipos de datos empleados en SCD aparecen en las tablas 4.4, 4.5, 4.6, 4.7, 4.8, 4.9, 4.10, 4.11, 4.12 y 4.13.

TSISTEMA		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Usuario	String * 50	Cadena de caracteres de longitud fija, almacena el *nombre* de usuario (máximo 50 caracteres).
Título	String * 80	Cadena de caracteres de longitud fija, almacena el *título* del sistema (máximo 80 caracteres).
Observaciones	String * 200	Cadena de caracteres de longitud fija, almacenan aquellas *observaciones* acerca del sistema (máximo 200 caracteres).
Entrada()	TEVARIABLE	Matriz dinámica (véase Tabla 4.6) de una dimensión. Cada posición contiene los datos de una variable de entrada en el sistema.
Salida()	TSVARIABLE	Matriz dinámica (véase Tabla 4.7) de una dimensión. Cada posición contiene los datos de una variable de salida en el sistema.
Regla()	TBCONOCIMIENTO	Matriz dinámica (véase Tabla 4.8) de una dimensión. Cada posición contiene los datos de una regla en la base de conocimiento del sistema.
Inferencia	TINFERENCIA	Contiene los parámetros de inferencia del sistema (véase Tabla 4.5).

Tabla 4.4: Tipo TSISTEMA.

TINFERENCIA		
CAMPO	**TIPO**	**DESCRIPCIÓN**
OpImplicacion	ET_IMPLICACIONES	Enumeración (véase Tabla 4.3) que indica las posibles *implicaciones* disponibles.
OpAgregacion	TSNorma	Tipo de datos (véase Tabla 4.11). Almacena los parámetros del operador de *agregación*.
OpAnd	TSNorma	Tipo de datos (véase Tabla 4.11). Almacena los parámetros del operador *and*.
OpOr	TSNorma	Tipo de datos (véase Tabla 4.11). Almacena los parámetros del operador *or*.
OpDifuminación	TSNorma	Tipo de datos (véase Tabla 4.11). Almacena los parámetros del operador de *difuminación*.

Tabla 4.5: Tipo TINFERENCIA.

TEVARIABLE		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Nombre	String * 40	Cadena de caracteres de longitud fija, almacena el *nombre* de la variable de salida (máximo 40 caracteres).
Unidades	String * 20	Cadena de caracteres de longitud fija, almacena la expresión del *sistema de unidades* de la variable de salida (máximo 20 caracteres).
UndSigla	String * 10	Cadena de caracteres de longitud fija, almacena la expresión del *sistema de unidades* de la variable de salida (máximo 40 caracteres).
Observaciones	String * 200	Cadena de caracteres de longitud fija, almacena las *observaciones* de la variable de salida (máximo 200 caracteres).
Activa	ETACTIVADO	Enumeración que indica el estado de activación de la variable.
UnivDiscurso (0 To 1)	Double	Matriz de dos elementos que almacenan el valor *mínimo* (0) y *máximo* del universo de discurso.
Difusor	TDIFUSORES	Tipo de datos (véase Tabla 4.12) que almacena los datos que definen un difusor.
ConjDif()	TCONJDIFUSO	Tipo de datos (véase Tabla 4.9) que contiene los parámetros que definen a una etiqueta lingüística o conjunto difuso en el sistema.
Vcrisp()	Double	Matriz dinámica que contiene la secuencia de valores actuales o crisps.

Tabla 4.6: Tipo TEVARIABLE.

TSVARIABLE		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Nombre	String * 40	Cadena de caracteres de longitud fija, almacena el *nombre* de la variable de salida (máximo 40 caracteres).
Unidades	String * 20	Cadena de caracteres de longitud fija, almacena la expresión del *sistema de unidades* de la variable de salida (máximo 20 caracteres).
UndSigla	String * 10	Cadena de caracteres de longitud fija, almacena la expresión del *sistema de unidades* de la variable de salida (máximo 40 caracteres).
Observaciones	String * 200	Cadena de caracteres de longitud fija, almacena las *observaciones* de la variable de salida (máximo 200 caracteres).
Activa	ETACTIVADO	Enumeración que indica el estado de activación de la variable.
UnivDiscurso (0 To 1)	Double	Matriz de dos elementos que almacenan el valor *mínimo* (0) y *máximo* del universo de discurso.
Defuzzificador	ETDEFUZZIFICADORES	Enumeración (véase Tabla 4.3) que almacena los tipos de defuzzificadores o concresores disponibles.
ConjDif()	TCONJDIFUSO	Tipo de datos (véase Tabla 4.9) que contiene los parámetros que define una etiqueta lingüística o conjunto difuso en el sistema.
CoordXAgreg()	Double	Matriz dinámica que almacena las coordenadas del eje de X de cada uno de los puntos que definen al conjunto difuso resultante del proceso de inferencia sobre la variable.
CoordYAgreg()	Double	Matriz dinámica que almacena las coordenadas del eje de Y de cada uno de los puntos que definen al conjunto difuso resultante del proceso de inferencia sobre la variable.
Vcrisp()	Double	Matriz dinámica que contiene la secuencia de valores actuales o crisps.

Tabla 4.7: Tipo TSVARIABLE.

TBCONOCIMIENTO		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Estado	ETESTADO	Enumeración (véase Tabla 4.3) que indica los tres posibles estados de la regla: activa, semiactiva y pasiva.
Antecedente()	TTERMINO	Matriz dinámica (veáse Tabla 4.10) de una dimensión. Cada posición almacena los datos que definen las características de cada término de los antecedentes en una regla.
GACTIVACION	Double	Valor resultado del proceso de difuminación. Almacena la *posibilidad (POSS)* de cada término de entrada.
Operador	ETOPERADOR	Enumeración que indica el operador seleccionado para la concatenación de términos del antecedente (operador *and* u *or*).
Consecuente()	TTERMINO	Matriz dinámica (veáse Tabla 4.10) de una dimensión. Cada posición almacena los datos que definen las características de cada término de los consecuentes en una regla.
Observaciones	String * 200	Cadena de caracteres de longitud fija, almacena las *observaciones* de la cada regla (máximo 200 caracteres).

Tabla 4.8: Tipo TBCONOCIMIENTO.

TCONJDIFUSO		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Etiqueta	String * 50	Cadena de caracteres de longitud fija, almacena el *nombre* de la etiqueta o conjunto difuso (máximo 50 caracteres).
Tipo	ETCONJDIF	Enumeración (véase Tabla 4.3) que indica el tipo de función de pertenencia asociado a cada etiqueta lingüística o conjunto difuso de la variable.
Características	TCARACTERISTICAS_CONJDIF	Tipo (véase Tabla 4.13) que sirve de almacenamiento de las cuatro características de cálculo para el proceso de defuzzificación: *área, centro de gravedad, altura, punto de máximo criterio.*
Poss	Double	Valor real que indica la comparación entre conjuntos difusos – difusor y etiquetas -.
GACTIVACION_MAX	Double	Almacena el grado de verdad de mayor valor asociado para una misma etiqueta de una variable de salida dado que en la base de conocimiento existirán distintos grados de activación sobre una misma etiqueta de salida.
PBasicos(0 to 3)	Double	Matriz de cuatro elementos que almacena los cuatro parámetros principales que definen a cada tipo de conjunto difuso.
Pextendidos()	Double	Matriz dinámica de una dimensión que contiene en las posiciones *pares* la coordenada X (universo discurso) de cada punto extendido en un conjunto difuso del tipo *trapecio extendido* y en los índices impares las coordenadas Y (grado de verdad en el intervalo [0,1]) de dichos puntos.
CoordXGActiv()	Double	Matriz dinámica de una dimensión que almacena las posiciones respecto al eje X (universo de discurso) del conjunto difuso resultante de aplicar el proceso de implicación sobre el conjunto difuso.
CoordYGActiv()	Double	Matriz dinámica de una dimensión que almacena las posiciones respecto al eje Y (grado de verdad en el intervalo [0,1]) del conjunto difuso resultante de aplicar el proceso de implicación sobre el conjunto difuso.

Tabla 4.9: Tipo TCONJDIFUSO

TTERMINO		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Activa	ETACTIVADO	Enumeración que indica el estado de activación del término.
OpNot	Boolean	Indica si está habilitado el operador *not* sobre el término, *True* o *False*.
NombreEoS	String * 40	Cadena de caracteres de longitud fija, almacena el *nombre* de la variable de entrada o salida al que hace mención el término si es parte del antecedente o del consecuente respectivamente (máximo 40 caracteres).
EtiquetaConjDif	String * 50	Cadena de caracteres de longitud fija, almacena el *nombre* del conjunto difuso de entrada o salida al que hace mención el término si es parte del antecedente o del consecuente respectivamente (máximo 50 caracteres).

Tabla 4.10: Tipo TTERMINO.

TSNORMA		
CAMPO	**TIPO**	**DESCRIPCIÓN**
TipoNorma	ET_TNORMAS_SNORMAS	Enumeración (véase Tabla 4.3) que permite seleccionar el tipo de t-norma o t-conorma disponibles.
P	Double	Valor real necesario en la expresión de algunas t-normas y t-conormas.

Tabla 4.11: Tipo TSNORMA.

TDIFUSORES		
CAMPO	**TIPO**	**DESCRIPCIÓN**
Tipo	ETDIFUSORES	Enumeración (véase Tabla 4.3) que indica el listado de difusores disponibles en la aplicación.
PBasicos(0 to 3)	Double	Matriz de cuatro elementos que almacena los cuatro parámetros principales que definen a cada tipo de difusor
Pextendidos()	Double	Matriz dinámica de una dimensión que contiene en las posiciones *pares* la coordenada X (universo discurso) de cada punto extendido en un difusor del tipo *trapecio extendido* y en los índices impares las coordenadas Y (grado de verdad en el intervalo [0,1]) de dichos puntos.

Tabla 4.12: Tipo TDIFUSORES.

TCARACTERISTICAS_CONJDIF		
CAMPO	**TIPO**	**DESCRIPCIÓN**
G	Double	Almacena el valor real tras el cálculo del *punto de máximo criterio* del conjunto difuso.
S	Double	Almacena el valor real tras el cálculo del *área* del conjunto difuso.
W	Double	Almacena el valor real tras el cálculo del *centro de gravedad* del conjunto difuso.
H	Double	Almacena el valor real tras el cálculo de la *altura* del conjunto difuso.

Tabla 4.13: Tipo TCARACTERISTICAS_CONJDIF.

4.2.4 Variables

Las variables son marcadores de posición que se utilizan para almacenar valores; tienen nombre y un tipo de dato. El tipo de dato de la variable determina cómo se almacenan los bits que representan esos valores en la memoria del equipo. Las variables globales de mayor relevancia en la aplicación se comentan en la Tabla 4.14.

VARIABLE	TIPO	DESCRIPCIÓN
SistActual	TSISTEMA	Variable principal del sistema. Almacena toda la información referente al sistema que se esté diseñando a través de la aplicación. En ella se contienen todos los datos del sistema tanto de diseño como de cálculos. En esta variable se almacena la información del fichero .SCD que cargue el usuario.
VentanaEntrada()	FrmEntrada	Matriz dinámica que almacena los datos de cada una de las ventanas de variables de entrada que estén visualizándose.
VentanaSalida()	FrmSalida	Matriz dinámica que almacena los datos de cada una de las ventanas de variables de salida que estén visualizándose.

Tabla 4.14: Algunas variables empleadas.

4.2.4.1 Dimensionado Dinámico: Instrucción *Redim*

En este apartado se expone la utilización de una potente instrucción de Visual Basic, *Redim* como medio para poder dimensionar de forma *dinámica* nuestra aplicación. La aplicación ha sido elaborada de forma que en tiempo de ejecución puedan ser *redimensionadas* las variables de matriz dinámica comentadas anteriormente.

Para ello se ha recurrido a la utilización de la instrucción *Redim()* al nivel de procedimiento para poder reasignar espacio de almacenamiento. La ayuda de Visual Basic (MSDN) aporta la siguiente información acerca de la instrucción:

Sintaxis

ReDim [**Preserve**] *nombre_variable***(***subíndices***)** [**As** *tipo*] [**,** *nombre_variable***(***subíndices***)** [**As** *tipo*]] **. . .**

La sintaxis de la instrucción **ReDim** consta de las partes que figuran en la Tabla 4.15.

Parte	Descripción
Preserve	Opcional. Palabra clave utilizada para conservar los datos de una matriz existente cuando se cambia el tamaño de la última dimensión.
nombre_ variable	Requerido. Nombre de la variable; sigue las convenciones estándar de nombre de variable.
Subíndices	Requerido. Dimensiones de una variable de matriz; se pueden declarar hasta 60 dimensiones múltiples. El argumento *subíndices* utiliza la sintaxis siguiente: [*inferior* **To**] *superior* [**,**[*inferior* **To**] *superior*] **. . .** Cuando no se declara explícitamente en *inferior*, el límite inferior de una matriz se controla mediante la instrucción **Option Base**. El límite inferior es cero si no hay ninguna instrucción **Option Base**.
Tipo	Opcional. Tipo de datos de la variable; puede ser Byte, Boolean, Integer, Long, Currency, Single, Double, Decimal (no admitida actualmente), Date, String (para cadenas de longitud variable), **String** * *length* (para cadenas de longitud fija), Object, Variant, un tipo definido por el usuario, o un tipo de objeto. Utilice una cláusula **As** *tipo* distinta para cada variable que defina. Para una **Variant** que contiene una matriz, *tipo* describe el tipo de cada elemento de la matriz, pero no cambia la **Variant** a algún otro tipo.

Tabla 4.15: Partes de la instrucción Redim.

Comentarios

La instrucción **ReDim** se utiliza para asignar o cambiar el tamaño de una matriz dinámica que ya se ha declarado formalmente mediante las instrucciones **Private**, **Public** o **Dim** con paréntesis vacíos (sin subíndices de dimensiones).

Puede utilizar la instrucción **ReDim** repetidamente para cambiar el número de elementos y dimensiones de una matriz. Sin embargo, no puede declarar una matriz de un tipo de datos y luego usar **ReDim** para cambiar la matriz a otro tipo de datos, a menos que la matriz esté contenida en una **Variant**. Si la matriz está contenida en una **Variant**, el tipo de los elementos se puede cambiar mediante una cláusula **As** *tipo*, a menos que esté utilizando la palabra clave **Preserve**, en cuyo caso no se permiten cambios al tipo de datos.

Si utiliza la palabra clave **Preserve** sólo puede cambiar el tamaño de la última dimensión de la matriz y no es posible cambiar el número de dimensiones. Por ejemplo, si la matriz sólo tiene una dimensión, puede cambiar el tamaño de esa dimensión porque es la última y única dimensión. Sin embargo, si la matriz tiene dos o más dimensiones, sólo puede cambiar la dimensión de la última y todavía conservar el contenido de la matriz. El ejemplo siguiente muestra cómo puede aumentar el tamaño de la última dimensión de una matriz dinámica sin borrar ninguno de los datos existentes contenidos en la matriz.

```
ReDim X(10, 10, 10)
. . .
ReDim Preserve X(10, 10, 15)
```

De modo parecido, cuando utiliza el argumento **Preserve** puede cambiar el tamaño de la matriz sólo cambiando el límite superior; cambiar el límite inferior produce un error.

Si hace que una matriz sea más pequeña de lo que era, perderá los datos de los elementos eliminados. Si transfiere una matriz a un procedimiento por referencia, no puede cambiar el tamaño de la matriz dentro del procedimiento.

Cuando se inicializan las variables, una variable numérica se inicializa a 0, una cadena de longitud variable se inicializa a una cadena de longitud cero ("") y una cadena de longitud fija se rellena con ceros. Las variables **Variant** se inicializan a Empty. Cada elemento de una variable de un tipo definido por el usuario se inicializa como si fuera una variable distinta. A una variable que se refiere a un objeto se le debe asignar un objeto existente mediante la instrucción **Set** antes de que se pueda usar. Hasta que se asigna a un objeto, la variable de objeto declarada tiene el valor especial **Nothing**, el cual indica que no se refiere a ninguna instancia en particular de un objeto.

Precaución: La instrucción **ReDim** actúa como una instrucción declarativa si la variable que declara no existe en el nivel de módulo o nivel de procedimiento. Si más tarde crea otra variable con el mismo nombre, incluso con un alcance mayor, **ReDim** hará referencia a la creada más tarde y no causará necesariamente un error de compilación, incluso aunque **Option Explicit** esté en efecto. Para evitar estos conflictos, **ReDim** no se debe utilizar como una instrucción declarativa, sino sólo para cambiar el tamaño de las matrices.

Nota: Para cambiar el tamaño de una matriz contenida en una **Variant**, debe declarar explícitamente la variable **Variant** antes de intentar cambiar el tamaño de su matriz.

4.2.5 Módulos de Código

Además de formularios se han realizado un total de 21 módulos de código, cada uno de ellos contiene un conjunto de enumeraciones, constantes, variables, procedimientos, funciones... de alcance público o privado que permiten desempeñar las distintas operaciones necesarias para la gestión de la aplicación.

Esta separación de código en distintos módulos tiene como objetivo agrupar el código según funcionalidad de forma que cada módulo de código contengan todos los procedimientos, funciones, variables..., comunes a cada bloque de funcionamiento de la aplicación. El propio nombre designado a cada módulo resume el objetivo del código que contiene. A continuación se exponen las características de cada uno de estos módulos, en un recuadro cada uno.

MÓDULO DE CÓDIGO: **SUBMAIN.BAS**

- Precarga en memoria de ventanas: *Sub Main*
- Opciones de inicialización: *SCD.INI.*
- Registro de Incidencias: *SCDREGISTRO.TXT.*
- Impedir el arranque de copias extras de la aplicación.

MÓDULO DE CÓDIGO: **APIS.BAS**

- Marco de trabajo para poder manipular las APIs de Windows, o Interfaz de Programación de la Aplicación.
- Declaración de tipos y constantes necesarios para la llamada a las funciones APIs de Windows.
- Declaración de funciones de la API Windows , es decir, establecer un vínculo de nuestro programa con la DLL Externa (librería) en la que se encuentra almacenada la función.
- Funciones de Ajuste: Establecen un paso intermedio entre las funciones APIs de Windows y las necesidades del programador.
- Funciones para Leer y Escribir Archivos según formato .INI: *LeerINI, EscribirINI*
  - [Sección n]
  - Clave m = Valor m
- Visualización de ventana en primer plano: *EnPrimerPlano*

MÓDULO DE CÓDIGO: **FILES.BAS**

- Gestion de la lectura / almacenamiento de información estructurada de los archivos específicos de la aplicación: *GuardarDatoSistema, LeerVector...*

MÓDULO DE CÓDIGO: **FUZZY.BAS**

- Contener el conjunto de procedimientos y funciones necesarios para realizar el proceso de fuzzificación o difuminación sobre las variables de entrada. A tal efecto se han desarrollado:
  - *DIFUMINAR_ :* Procedimientos que realizan la intersección entre coordenadas de difusor (posicionado respecto valor actual) y conjunto difuso y devuelven con o sin interfaz gráfica de representación del propio proceso dicho conjunto difuso resultante y el valor de *posibilidad*.
  - *COMPROBAR_METODO_DIFUMINACIÓN:* Si el difusor y conjunto difuso (etiqueta) no se solapan en ningún punto se evita proceso de intersección, se reduce tiempo de computación al realizar ésta comprobación.
  - *POSIBILIDAD:* Obtener del conjunto difuso resultante de la intersección, el valor más representativo del mismo (máximo).

MÓDULO DE CÓDIGO: **FUZZYSETS.BAS**

- Gestión y reestructuración de los datos que definen a difusor y conjuntos difusos para el tratamiento por parte de los procedimientos de cálculo DIFUMINAR_, DEFUZZIFICAR_ ...
- *VECTORPTOSDIFUSOR:* Devuelve matriz dinámica con las coordenadas de los puntos posicionados respecto al valor crisp actual seleccionado.
- *VECTORPTOSCDIFUSO:* Devuelve matriz dinámica con las coordenadas de los puntos que forman el conjunto difuso seleccionado.
- *SPLINES_INPUT:* Agrupación de llamada de los procedimientos VECTORPTOS...
- *SPLINES_OUTPUT:* Obtención de la función polinómica definida a trozos (spline) o coordenadas del conjunto difuso de altura el grado de activación sobre uno de los conjunto difusos de dicha variable de salida junto a las coordenadas del mismo.

MÓDULO DE CÓDIGO: **ARRAYS.BAS**

- Contener el conjunto de procedimientos y funciones necesarias para la gestión de los distintos vectores y matrices que se manipulan en los restantes módulos de código: *IgualarLimitesVectores, IntroducirValorMatriz, SepararCoordXY.*

MÓDULO DE CÓDIGO: **DEFUZZY.BAS**

- Contiene el conjunto de funciones y procedimientos necesarios para el proceso de concreción o defuzzificacion sobre el resultado de la fase de inferencia en la variables de salida del sistema de control difuso:
  - *CALCULAR_CARACTERÍSTICAS_ :* Procedimientos que calculan los parámetros característicos (W, G, H, S) de un conjunto difuso o salida.
  - *DEFUZZIFICAR_ :* Procedimientos que calculan, sobre una variable de salida junto a un identificador de valor actual, el valor de concreción según un método.
  - *PROPIEDADES (Qdefuzzificador,...):* Devuelven para la enumeración ETDEFUZZIFICADORES una cadena de texto y viceversa.
  - *SPLINE_ :* Calcula, sobre las coordenadas de un conjunto difuso [CoordX(), CoordY()], los parámetros de defuzzificación.

MÓDULO DE CÓDIGO: **GRAPHS.BAS**

- Procedimientos y funciones que tienen como objetivo la representación a través de controles y métodos gráficos de las características de las variables *difusor, conjuntos difusos, secuencias ...*
- *DIBUJAR_SECUENCIA_ :* Visualizan sobre un control PictureBox los valores almacenados en matrices dinámicas X e Y permitiendo opciones como resaltado de punto, color, grosor de la línea de unión entre puntos...
- *DIBUJARGRAFICOSVARIABLE:* Representa sobre un control PictureBox en una cuadrícula y según un eje de coordenadas cartesianas las etiquetas lingüísticas y difuminador si se trata de una variable de entrada que contienen cada variable del sistema.

MÓDULO DE CÓDIGO: **FUZZYOPSETS.BAS**

- Implementación de algoritmos que realicen por aplicación de T-Normas o S-Normas las operaciones de intersección o unión respectivamente sobre dos conjuntos difusos.
  Nota: Se entiende como conjunto difuso desde el punto de vista del tratamiento de código a dos vectores de matrices dinámicas que almacenan las coordenadas X e Y de los puntos que lo forman. Luego, estos procedimientos reciben cuatro matrices de coordenadas y devuelven dos matrices de coordenadas como resultado de la operación.
- Están implementados los siguientes algoritmos:
  - *T-NORMA DEL MÍNIMO.*
  - *S-NORMA DEL MÁXIMO.*

MÓDULO DE CÓDIGO: **INFERENGINE.BAS**

- Contiene los procedimientos, funciones y propiedades necesarias para realizar el proceso de inferencia.
- *QNORMA* y *QIMPLICACIÓN:* Propiedades que devuelven, en forma de cadena, el tipo de norma e implicación de una lista enumerada.
- *TySNORMAS:* Implementación de la T-Normas y S-Normas.
- *GACTIVACION_REGLAS:* Obtiene el grado de activación de cada regla.
- *OBTENER_CDIF_GACT_:* Procedimientos para la obtención, con o sin interfaz gráfica ,del conjunto difuso resultante de la aplicación del operador de implicación sobre la variable de salida.
- *OBTENER_CDIF_AGREG_SAL:* Devuelve el conjunto difuso resultante de aplicar el operador de agregación a las etiquetas que previamente han sido modificadas por el operador de implicación.
- *INFERIR_REGLAS:* Procedimiento global que agrupa todos los anteriores.

MÓDULO DE CÓDIGO: **INOUT.BAS**

- Gestión de los índices, números de elementos, contenidos... COMUNES a las variables de entrada y salida del sistema.
- *MODIFICARPTOSUNIVERSO_:* Procedimientos que permiten mantener el escalado proporcional al inicial ante variaciones de los límites en el universo de discurso.
- *ESTA_:* Propiedades que permiten comprobar el estado de campos referentes a una variable de entrada o salida tomando como parámetros de entrada un índice que apunte a la posición de los datos de esa variable en su correspondiente matriz.
- *Q_:* Propiedades que devuelven el valor del campo solicitado en su llamada tomando como parámetros de entrada un índice de posición de los datos de la variable en su matriz.

MÓDULO DE CÓDIGO: **INPUTS.BAS**

- Gestión de las propiedades, elementos ... SÓLO de las variables de entrada.
- *SUPRIMIDAENTRADA*: Variable global que almacena la última variable de entrada eliminada para permitir la opción de recuperación.
- *_ENTRADAMATRIZ:* Procedimientos que gestionan la adición y eliminación de una variable de entrada en la estructura del sistema.
- *NENTRADAS_:* Propiedades que indican el número de entradas, entradas activas y pasivas en el sistema.

MÓDULO DE CÓDIGO: **OUTPUT.BAS**

- Gestión de las propiedades, elementos... SÓLO de las variables de salida.
- *SUPRIMIDASALIDA*: Variable global que almacena la última variable de salida eliminada para permitir la opción de recuperación.
- *_SALIDAMATRIZ:* Procedimientos que gestionan la adición y eliminación de una variable de entrada en la estructura del sistema.
- *NSALIDAS_:* Propiedades que indican el número de salidas, salidas activas y pasivas en el sistema.

MÓDULO DE CÓDIGO: **KERNEL.BAS**

- Contiene la definición de las estructuras, enumeraciones, variables y constantes más importantes sobre las que se sustenta toda la información del sistema.
- *LIBERARDEMEMORIA:* Procedimiento que como su nombre indica libera de memoria todos los datos almacenados por la variable principal del sistema *SISTACTUAL.*

MÓDULO DE CÓDIGO: **MANAGAMENTWINDOWS.BAS**

- Procedimientos y funciones que permitan gestionar las ventanas de variables de entrada y salida de la aplicación.
- *CARGARVENTANA_:* Carga la variable de entrada o salida según índice de posición en matriz dinámica Entrada() o Salida() respectivamente sobre VentanaEntrada() o VentanaSalida() de forma dinámica.
- *DESCARGARVENTANA_:* Operación inversa a la de carga sobre VentanaEntrada() o VentanaSalida()
- *QVENTANA_TIENE_:* Determina que índice de posición en VentanaEntrada() o VentanaSalida() contiene la variable de entrada o salida respectivamente cargada.
- *REESTRUCTURARVECTORVENTANA_:* Tras la eliminación de una posición del vector dinámico VentanaEntrada() o VentanaSalida() se ha de proceder a su redimensionado.

MÓDULO DE CÓDIGO: **MESSAGES.BAS**

- Contener los procedimientos y funciones necesarias para la formación de cuadros de mensajes utilizados en la aplicación sobre un formulario común.

MÓDULO DE CÓDIGO: **POINTS.BAS**

- Gestiona la manipulación de los puntos que definen los conjuntos difusos.
- *BUSCARLIMITADORESPTO:* Devuelve los valores mínimo y máximo que puede adoptar un punto sobre el universo de discurso.
- *INTRODUCIRPEMATRIZ:* Introduce en la matriz dinámica de puntos extendidos del *Trapecio Extendido* el nuevo punto según su valor sobre el universo de discurso.
- *ESVALIDOPTO:* Comprueba si tras la modificación de la posición inicial del mismo el nuevo valor es correcto.
- *QP_:* Devuelve la cadena de puntos ya sean básicos o extendidos de un conjunto difuso dado.
- *REORDENARPBYE:* Unifica la matriz dinámica de puntos extendidos si fuese un *Trapecio Extendido* con la puntos básicos de forma tal que: A, E_1, ..., E_n, B, C, E_{n+1}, ..., E_m, D.

MÓDULO DE CÓDIGO: **RESOURCES.BAS**

- Contiene los identificadores necesarios de los distintos recursos (iconos, mapa bits...) disponibles en el fichero "RECURSOS.RES".

MÓDULO DE CÓDIGO: **RULES.BAS**

- Contiene el código (propiedades, funciones, procedimientos, variables, constantes...) necesarias para procesar la información acerca de las reglas del sistema.
- *NUEVAREGLA, SUPRIMIDAREGLA:* Variables que contienen respectivamente los datos de la nueva regla a introducir en el sistema y los datos de la última regla eliminada para poder restaurarla si fuese necesario.
- *ELIMINARTERMINOS_:* Procedimientos que eliminan de la base de conocimiento todos aquellos términos que hagan mención a un determinado conjunto difuso o variable.
- *ELIMINAREGLAMATRIZ:* Procedimiento que elimina de la base de conocimiento una regla y la almacena temporalmente en *suprimidaregla*.
- *ESTA_:* Propiedades que permiten averiguar si existen en la base de conocimiento referencias a una determinada etiqueta, o a variables de entrada o salida.
- *EXPRESIÓN_:* Procedimientos que devuelven cadenas de texto con las expresiones del antecedente, consecuente o ambos de una regla.
- MOD_: Procedimientos que buscan y sustituyen el contenido de un término de una regla, ya sea referente al nombre de la variable a la que haga mención, o bien de la etiqueta que menciona.
- *NREGLAS_:* Propiedades que determinan el número de reglas *activas, semiactivas* y *pasivas*.
- *Q_:* Propiedades que buscan según un identificador de regla y término (antecedente o consecuente) una característica del mismo.
- *VERIFICAR_:* Procedimiento que verifica el estado de cada una de las reglas de la base de conocimiento y comprueba que dicho estado es coherente con el estado de cada una de las variables a las que hace mención. Si no fuese así procederá a informar del cambio de estado.

MÓDULO DE CÓDIGO: **SEGMENTS.BAS**

- Contiene el conjunto de procedimientos y funciones necesarias para el tratamiento de segmentos en dos dimensiones.
- *PTOCORTESEGMENTOS:* Procedimiento que calcula los valores de intersección de dos segmentos.
- *PTOINTSEGQSECRUZANUNPTO:* Procedimiento que devuelve el punto de intersección para dos segmentos.

MÓDULO DE CÓDIGO: **UTILITIES.BAS**

- Contiene aquellas funciones y procedimientos de utilidad general para el tratamiento de información en el desarrollo de aplicaciones.
- *VISUALIZARFRM:* Procedimiento que visualiza un formulario a través del método *show* con o sin centrado.
- *EXISTEFICHERO:* Comprueba si un fichero está en la ubicación indicada. Si el medio no estuviese disponible (ej. el diskette no ha sido insertado) será advertido.
- *FRMPOSICIONAR_:* Posiciona un formulario ya visible de forma centrada, aleatoriamente o con respecto a un formulario primario si es una aplicación MDI.

4.2.6 Formularios

La elaboración de *SCD* trae consigo un total de 39 formularios (ventanas). Dado que se trata de una aplicación de interfaz de múltiples documentos (MDI), hay un formulario primario (ventana principal) y tres son considerados secundarios (contenidos dentro del primario). De estos secundarios, dos son utilizados con múltiples instancias de los mismos: son los formularios de variables de entrada y salida (tantas instancias del formulario de entrada como número de variables y con respecto a las variables de salida igual). El resto de formularios son los utilizados por la aplicación para visualizar mensajes, opciones... y demás acciones necesarias para el desarrollo de la aplicación.