PREPROCESADOR DE FUZZY INFERENCE SYSTEM (FIS) Y MOTOR DE INFERENCIA DIFUSA PARA LA PLATAFORMA DE DESARROLLO EMBEBIDA FREESCALE TWR-K70F120M

CRISTHIAN ALONSO ZÁRATE VÁSQUEZ

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2016

PREPROCESADOR DE FUZZY INFERENCE SYSTEM (FIS) Y MOTOR DE INFERENCIA DIFUSA PARA LA PLATAFORMA DE DESARROLLO EMBEBIDA FREESCALE TWR-K70F120M

CRISTHIAN ALONSO ZÁRATE VÁSQUEZ

Trabajo de grado para optar al título de
Ingeniero de Sistemas

Director
Luis Carlos Gómez Flórez
M.Sc. en Informática

Codirector
Jorge Enrique Meneses Flórez
M.Sc. en Ingeniería Mecánica

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2016

AGRADECIMIENTOS

Le expreso mi total agradecimiento:

Al profesor M.Sc. Luis Carlos Gómez Flórez, director del proyecto por su orientación y gran apoyo para llevar a cabo este trabajo.

Al profesor M.Sc. Jorge Enrique Meneses Flórez por el constante acompañamiento y asesoría brindada durante el desarrollo del proyecto.

A los profesores de la escuela de Ingeniería de sistemas por motivar mi realización como persona íntegra además de formarme en el ámbito académico.

A mis compañeros de carrera que me compartieron ideas en los momentos en que lo necesité.

A mis compañeros del laboratorio de Automatización Industrial por el aliento y compañerismo que de ellos recibí.

Y a mi familia por confiar en mis capacidades y proveerme sustento en los momentos de necesidad.

CONTENIDO

LISTA DE FIGURAS

LISTA DE ANEXOS

RESUMEN

TÍTULO: PREPROCESADOR DE FUZZY INFERENCE SYSTEM (FIS) Y MOTOR DE INFERENCIA DIFUSA PARA LA PLATAFORMA DE DESARROLLO EMBEBIDA FREESCALE TWR-K70F120M*

AUTOR: ZÁRATE VÁSQUEZ, Cristhian Alonso**

PALABRAS CLAVE: Lógica difusa, Modelos difusos, Fuzzy Inference System, Motores de inferencia difusa, Sistemas embebidos.

DESCRIPCIÓN:

Este documento detalla el desarrollo de una solución software que satisface la necesidad presente en el proyecto de investigación 8556 “Desarrollo de un prototipo de pozo inteligente para CEC” de Campo Escuela Colorado, el cual busca reducir costos de operación y gastos de capital, aumentar los niveles de producción y reservas y detectar de manera precisa y oportuna las fallas que se puedan presentar en un pozo, mediante el uso de elementos hardware y software en los sistemas de levantamiento artificial de tipo bombeo mecánico (concepto de Pozo y Campo Inteligentes).

El proyecto de investigación 8556 desarrolló en sus dos primeras fases el “Controlador de Pozo Inteligente” CPI, solución de hardware y software que permite recolectar y presentar la información de operación de un sistema de bombeo mecánico a través de sensores que se conectan de manera inalámbrica a un dispositivo embebido ubicado en la cercanía de la cabeza de pozo.

El proyecto denominado “Preprocesador de Fuzzy Inference System (FIS) y Motor de Inferencia Difusa para la Plataforma de Desarrollo Embebida Freescale TWR-K70F120M” propuso e implementó la aplicación software “Preprocesador de Fuzzy Inference System para Plataformas Embebidas” FISPES, la cual es capaz de transformar un modelo difuso, contenido en un archivo de texto, en un motor de inferencia que se ejecuta sobre el dispositivo embebido presente en el CPI.

* Trabajo de grado.

** Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director: M.Sc. Luis Carlos Gómez Flórez

ABSTRACT

TITLE: FUZZY INFERENCE SYSTEM (FIS) PREPROCESSOR AND FUZZY INFERENCE ENGINE FOR THE FREESCALE TWR-K70F120M EMBEDDED PLATFORM*

AUTHOR: ZARATE VASQUEZ, Cristhian Alonso**

KEYWORDS: Fuzzy logic, fuzzy models, Fuzzy Inference System, fuzzy inference engine, embedded systems.

DESCRIPTION:

This document details the development of a software solution that satisfies the needs present in the research project 8556 “Desarrollo de un prototipo de pozo inteligente para Campo Escuela Colorado” (Development of a Smart Well prototype for *Campo Escuela Colorado*), which seeks to reduce operational costs and capital expense, increase production and reserve levels, and to accurately detect failures that may occur in an oil well, by using software and hardware elements in mechanical pump type artificial lift systems (Smart Well and Smart Field concepts).

The research project 8556 in its first two stages, developed the “Controlador de Pozo Inteligente” (Smart Well Controller), abbreviated CPI, a hardware and software solution that collects and presents information about operation of a mechanical pump system through sensors that transmit wirelessly to an embedded system located in the vicinity of the machinery.

The bachelor thesis “Fuzzy Inference System (FIS) Preprocessor and Fuzzy Inference Engine for the Freescale TWR-K70F120M Embedded Platform” proposed and implemented the software application “Fuzzy Inference System Preprocessor for Embedded Systems”, FISPES, which is able transform a fuzzy model, contained in a plain text file, into a fuzzy inference engine that can be run on the embedded system located in the CPI.

* Bachelor Thesis.

** Faculty of Physicomechanical Engineering. School of Software Engineering and Computer Science. Director: M.Sc. Luis Carlos Gómez Flórez

INTRODUCCIÓN

Este documento contiene los resultados de la culminación del proyecto de grado "Preprocesador de Fuzzy Inference System y motor de inferencia para la plataforma de desarrollo embebida TWR-K70F120M" en el cual se propuso el diseño y la implementación de una herramienta software que responde a las necesidades planteadas por el proyecto de investigación 8556 "Desarrollo de un prototipo de pozo inteligente para CEC".

En el primer capítulo se presenta el alcance del proyecto, se plantea el problema a resolver, se justifica la realización del proyecto, se proponen los objetivos desarrollados y se describe la metodología que se siguió para llevar a término el proyecto.

En el segundo capítulo se plantea la relación de este proyecto con el proyecto de investigación 8556 y se presenta la solución software a las necesidades del proyecto, el *Preprocesador FIS para plataformas Embebidas FISPES* y cada uno de sus módulos.

En los capítulos 3, 4 y 5 se describe a profundidad la solución desarrollada:

En el tercer capítulo se describe el módulo "Herramienta de manejo de archivos de entrada", el modelo difuso que recibe la entrada del preprocesador y los elementos que lo componen: El contexto gramatical y la gramática.

En el cuarto capítulo se describe el módulo "Modelo de datos", las entidades que lo conforman y el submódulo normalizador de sistema difuso.

En el quinto capítulo se describe el módulo "Herramienta de manejo de archivos de salida", los algoritmos que componen el módulo y el motor de inferencia difusa que conforma la salida del preprocesador.

En el sexto capítulo se describen las características del primer y segundo prototipo obtenidos y el resultado final del desarrollo de la solución.

En el séptimo capítulo se describen las pruebas realizadas al preprocesador FISPES y al motor de inferencia para Campo Escuela Colorado CEC y se presentan los resultados de cada prueba.

En el octavo capítulo se enumeran las conclusiones del proyecto y se presentan recomendaciones para trabajos futuros que se basen en este proyecto.

1. ALCANCES DEL PROYECTO

1.1 IDENTIFICACIÓN DEL PROBLEMA

Los ingenieros de producción están constantemente buscando formas de optimizar la producción de los pozos de petróleo con el mínimo desplazamiento a los sitios. Pozos inteligentes y Campos inteligentes (Intelligent wells - Smart Fields), son conceptos modernos que conducen al aumento en la producción de un campo petrolero. Así mismo, son términos que abarcan todas aquellas soluciones centradas en las tecnologías de información y de automatización, que permiten a las empresas aumentar la productividad de los recursos disponibles.

Campo Escuela Colorado (CEC) es un terreno destinado a la explotación petrolera, cedido en convenio por Ecopetrol a la Universidad Industrial de Santander en el año 2006, en donde la universidad lleva a cabo proyectos de investigación y extensión. Los pozos petroleros presentes en Campo Escuela Colorado se encuentran en la etapa de madurez y presentan características que dificultan el proceso de extracción y pueden ocasionar fallas en la maquinaria de bombeo. La presencia de personal que supervise la actividad no es suficiente para garantizar la operación continua debido a la cantidad de factores que intervienen en el mal funcionamiento y que dificultan la detección temprana de fallas. Al aplicar el concepto de Pozo y Campo Inteligente se busca reducir costos de operación y gastos de capital, aumentar los niveles de producción y reservas y detectar de manera precisa y oportuna las fallas que se puedan presentar en un pozo.

En el marco de Campo Escuela Colorado (CEC) se ejecutó la fase 1 del proyecto de investigación 8556 “Desarrollo de un prototipo de pozo inteligente para CEC”, a través del cual se diseñó y construyó una solución propia e innovadora de hardware y software, que ha permitido establecer las bases para el concepto de POZO Y CAMPO INTELIGENTE. En la fase desarrollada se obtienen en tiempo real los dinagramas de fondo de un sistema de bombeo mecánico, utilizando sensores ubicados en la unidad de bombeo y se realiza la transmisión inalámbrica de estos datos a un DISPOSITIVO EMBEBIDO, ubicado en la cercanía.

La fase 2 del proyecto debe identificar con precisión y rapidez los problemas (fallas) en un pozo, incorporando un sistema automático (software) que reconozca los patrones de falla del sistema de bombeo mecánico, a partir de los dinagramas de fondo, anticipando los problemas a través de su pronta identificación.

El trabajo de grado “Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico”, permitió concluir que, utilizando el software NEFCLASS-Q, se puede

obtener un modelo difuso (o modelo fuzzy), que diagnostique, a partir de los dinagramas de fondo, las fallas presentes en un sistema de bombeo mecánico.

¿Cuál es el problema?
El Controlador de Pozo Inteligente (CPI), desarrollado en el proyecto de investigación 8556 utiliza como hardware el sistema embebido TWR-K70F120M de Freescale, basado en el microcontrolador MK70FN1M0VMJ12 ARM® Cortex®-M4 (ver Anexo B). Al utilizar NEFCLASS-Q, se obtiene un modelo difuso, que no puede ser llevado directamente al sistema embebido.

1.2 JUSTIFICACIÓN

El Controlador de Pozo Inteligente CPI desarrollado en el proyecto de investigación 8556, automatiza el proceso de obtención de información a modo de dinagramas de fondo, pero carece de la capacidad de generar conocimiento y evitar la presencia de personal de supervisión.

Es necesario el desarrollo de la solución que supla las necesidades de generación de conocimiento en el vacío ocasionado por la capa de software faltante.

1.3 OBJETIVOS

1.3.1 Objetivo general. Desarrollar un preprocesador de Fuzzy Inference System (FIS) y motor de inferencia basado en lógica difusa que opere sobre la plataforma de desarrollo embebida Freescale TWR-K70F120M.

1.3.2 Objetivos específicos

- Diseñar e implementar un modelo de datos para el manejo de información de sistemas difusos Fuzzy Inference System (FIS) en base a las entidades que establece la teoría de lógica difusa y las herramientas de manejo de archivos que den soporte a la tarea de preprocesamiento.
- Implementar los segmentos de código que conforman el motor de inferencia difuso resultante de la operación del preprocesador, de acuerdo con las características ofrecidas por FIS.
- Documentar el preprocesador bajo el estándar UML utilizando como mínimo los diagramas de clases, de estados y de secuencia.

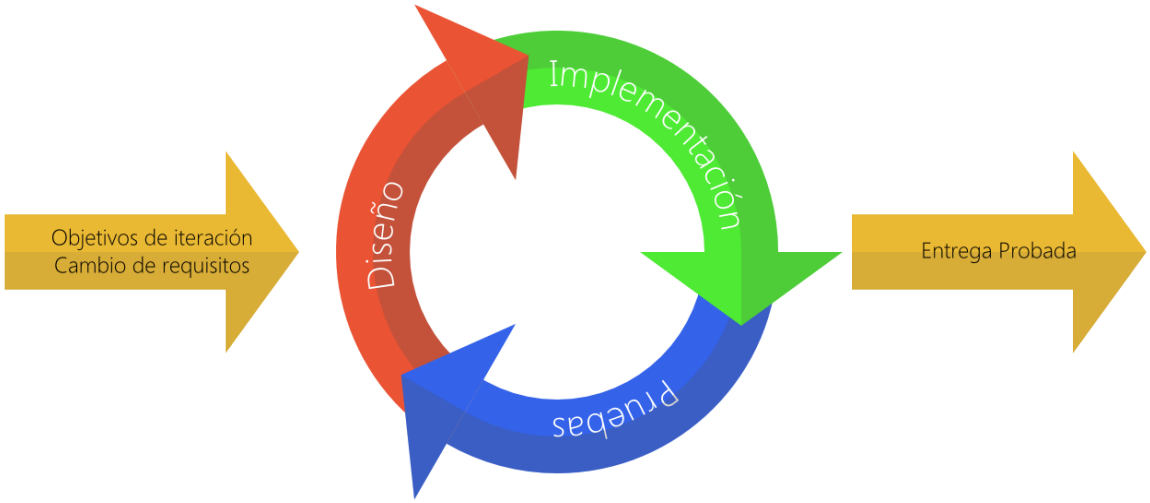
- Realizar pruebas de integración y de sistema que verifiquen el buen comportamiento del preprocesador.
- Validar el motor de inferencia generado por el preprocesador a partir del modelo difuso de interés para Campo Escuela Colorado, usando los datos de entrada y resultados esperados generados por el mismo.

1.4 METODOLOGÍA

Para la realización de este trabajo de grado se tuvo como guía las actividades del ciclo de vida del software: análisis, diseño, implementación, pruebas, documentación, evaluación y mantenimiento. Durante la etapa de análisis, las necesidades y pormenores del proyecto fueron definidos, sin embargo, una nueva iteración en la aplicación de las actividades del desarrollo garantizó que los requerimientos se ajustaran a las necesidades reales del cliente.

La metodología de desarrollo que opera en estas condiciones es el modelo iterativo (ver Figura 1). Este método, a pesar de que puede presentar desventajas (en comparación con otros modelos como el de cascada) tales como un mayor grado de dificultad para administrarse, menor claridad en las metas (o hitos) en el proceso de desarrollo y que ninguna etapa resulte completamente finalizada, ofrece beneficios como el manejo más adecuado del cambio y la calidad, captura y mitigación de errores temprana, entregas más tempranas y fomento de la comunicación con el cliente.

Figura 1. Modelo de desarrollo iterativo

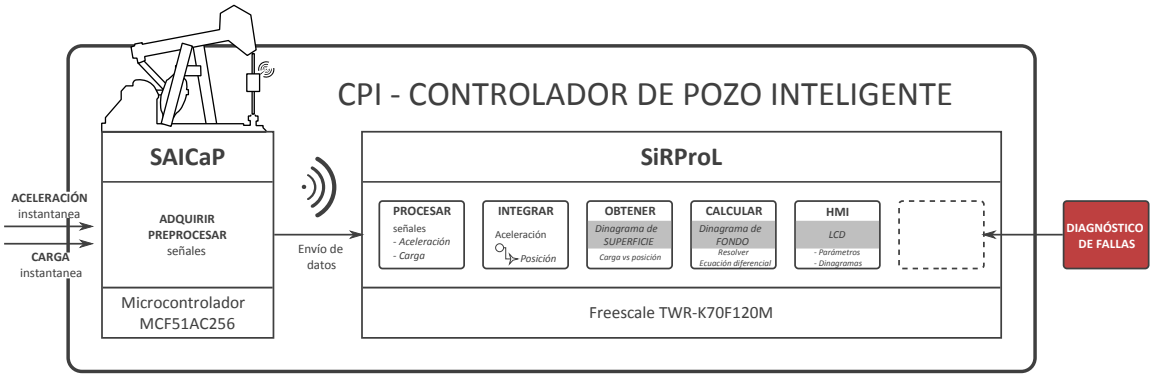


2. DESCRIPCIÓN DEL PROYECTO DESARROLLADO

2.1 INTRODUCCIÓN

En el proyecto de investigación 8556, “Desarrollo de un prototipo de pozo inteligente para CEC”, se desarrolló el Controlador de Pozo Inteligente CPI, el cual adolece de una capa de software que diagnostique de manera automática las fallas presentes en un sistema de bombeo mecánico. El proyecto aquí presentado proporciona dicha capa (ver Figura 2).

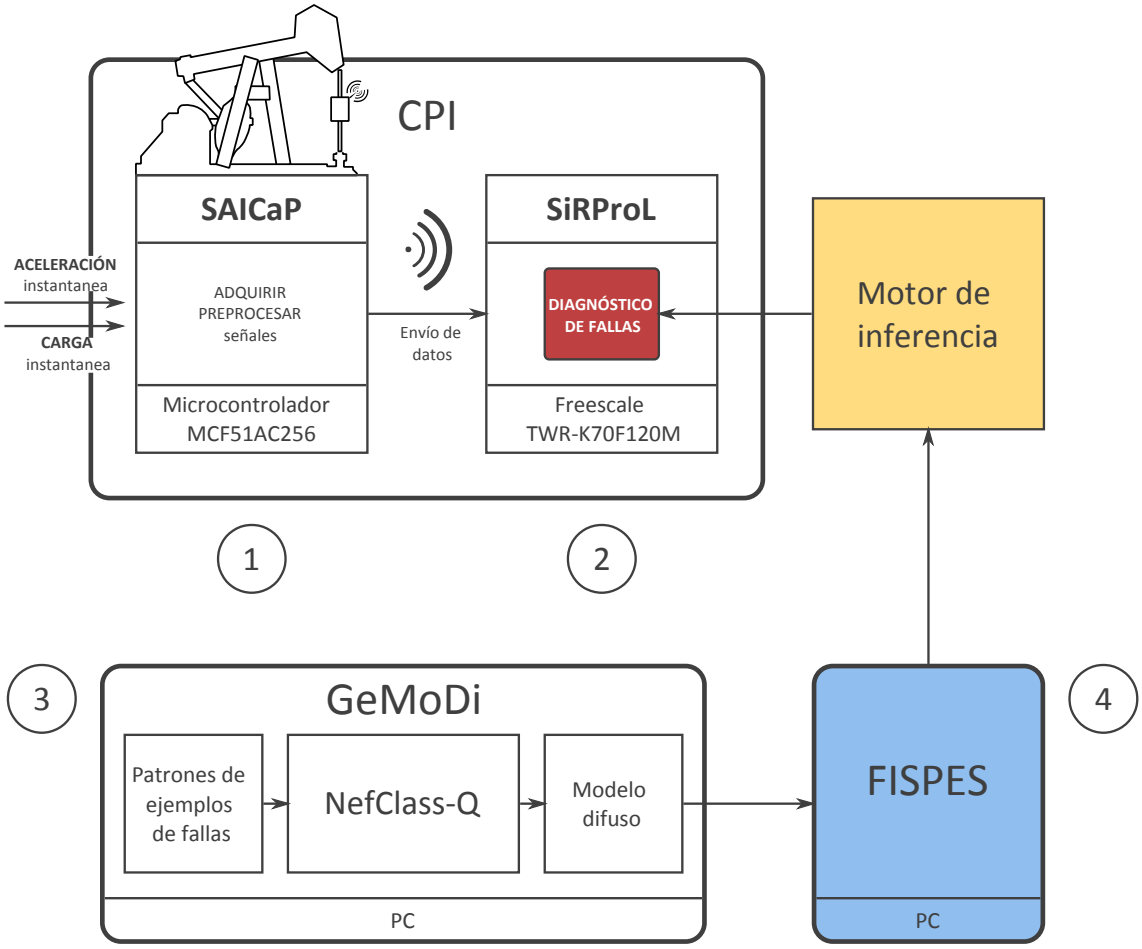
Figura 2. Capa de software faltante en el CPI



Fuente: Meneses F., Jorge E. Acevedo P., Alfredo R. Ramón S., Jorge H. Desarrollo de un prototipo de pozo inteligente para CEC. Universidad Industrial de Santander.

2.2 RELACIÓN ENTRE EL PROYECTO DESARROLLADO Y EL CPI

Figura 3. Solución a la capa de software faltante



El SAICaP ① (Sistema de Adquisición de datos Inalámbrico (Wireless) en Cabeza de Pozo), ubicado en cabeza de pozo sobre la barra pulida debe obtener en tiempo real y de manera permanente las señales instantáneas de aceleración y carga presentes en la barra de pulida de la unidad de levantamiento artificial. Una vez obtenidas y procesadas las señales mediante hardware y software, éstas son transmitidas vía inalámbrica (ZigBee) a otro sistema remoto (SiRProL) que se encarga de obtener los dinagramas de fondo.

El SiRProL ② (Sistema de Recepción y Procesamiento Local en pozo) recibe vía inalámbrica (ZigBee) los datos de carga y aceleración de la barra pulida del pozo

adquiridos por el sistema SAICaP. SiRProL procesa los datos para obtener en primera instancia el dinagrama de cabeza de pozo y, mediante procesamiento matemático - computacional, obtiene el dinagrama de fondo de pozo.

El GeMoDi ③ (Generador de Modelo difuso para pozos) utiliza la herramienta software NEFCLASS-Q para generar un modelo difuso a partir de patrones de ejemplos de fallas recolectados de expertos en análisis de dinagramas y de resultados de la operación de extracción en los pozos de Campo Escuela Colorado.

Para transformar el modelo difuso en código ejecutable en el sistema embebido se hace necesario interpretar la información contenida en el modelo y replantearla como instrucciones.

El proyecto “PREPROCESADOR DE FUZZY INFERENCE SYSTEM (FIS) Y MOTOR DE INFERENCIA DIFUSA PARA LA PLATAFORMA DE DESARROLLO EMBEBIDA FREESCALE TWR-K70F120M”, descrito en este documento, cuenta con la herramienta software FISPES ④ (Preprocesador de FIS para Plataformas Embebidas) que produce un motor de inferencia difusa para plataformas embebidas a partir de un modelo difuso proveído.

FISPES ingresa al proyecto de investigación 8556 como la solución a la capa de software faltante en el Controlador de Pozo Inteligente. A continuación se realiza la presentación de la solución.

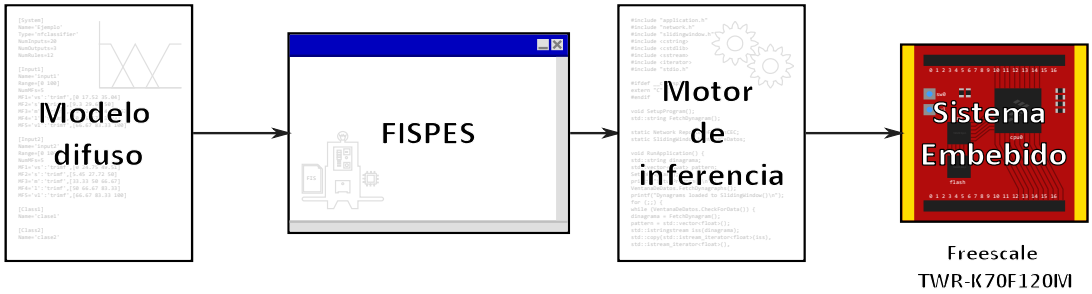
2.3 PREPROCESADOR DE FIS PARA PLATAFORMAS EMBEBIDAS (FISPES)

La solución software desarrollada en este proyecto, FISPES, se planteó como una aplicación de escritorio, compilable con los sistemas operativos Windows y Linux. Se acordó desarrollar la aplicación en lenguaje C#, que cuenta con herramientas para ejecutarse en sistemas operativos tipo Windows y Unix-like, cuenta además con una sintaxis bastante sencilla, semejante a la sintaxis de Java y posee una Interfaz de programación de aplicaciones (API) extensa, lo que permitió que el proceso de desarrollo fuera ágil y consistente.

El preprocesador FISPES extrae la información de modelos difusos contenidos en archivos de texto plano y le realiza un tratamiento (interpretación) para plasmarla en un motor de inferencia, el cual se almacena en un archivo de código y puede ser ejecutado en la plataforma embebida TWR-K70F120M.

El entorno de operación de FISPES se ilustra en la Figura 4.

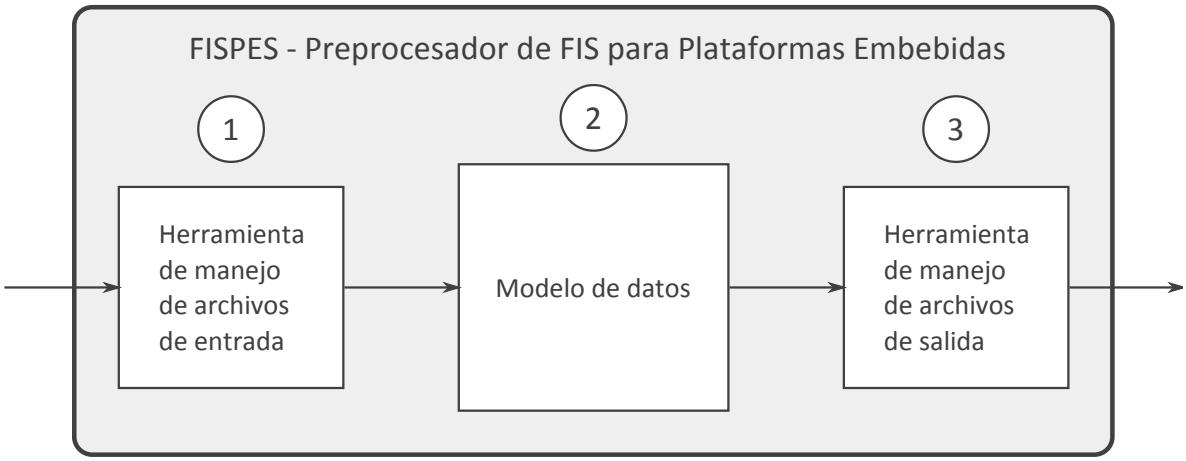
Figura 4. Función de FISPES



El diseño del preprocesador FISPES refleja los componentes principales de un sistema de procesamiento: entrada, unidad de procesamiento y salida.

FISPES está compuesto por tres subsistemas o módulos: la herramienta de manejo de archivos de entrada, el modelo de datos y la herramienta de manejo de archivos de salida (ver Figura 5).

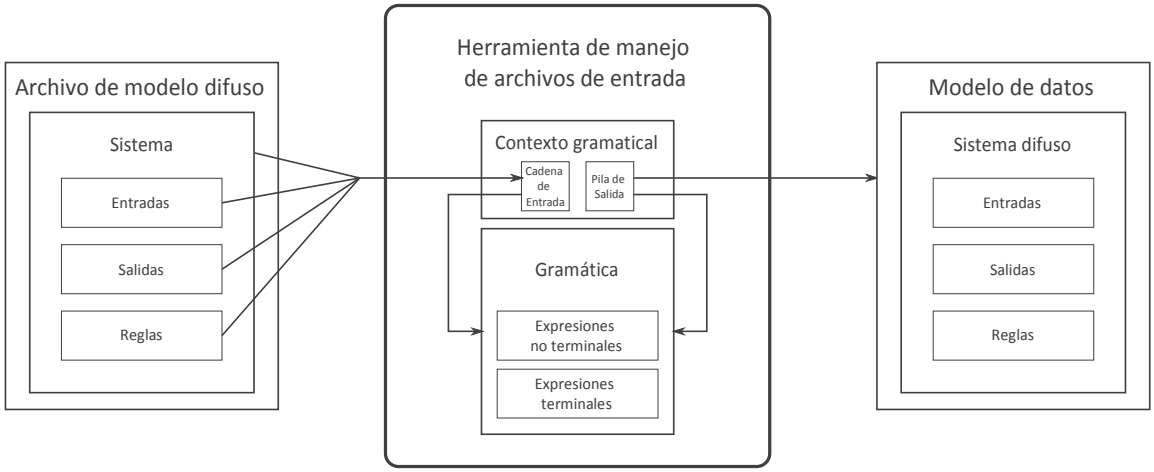
Figura 5. Componentes de FISPES



A continuación se describe en detalle cada uno de los módulos del preprocesador.

2.3.1 Herramienta de manejo de archivos de entrada. La herramienta de manejo de archivos de entrada (ver Figura 6) se encarga de la lectura del archivo que contiene la definición del modelo difuso a interpretar, tiene como base el patrón de diseño intérprete, el cual propone el uso de una gramática para capturar datos desde una fuente y pasarlos a un cliente interno o externo a la aplicación. La gramática se define usando las reglas de producción del lenguaje a interpretar.

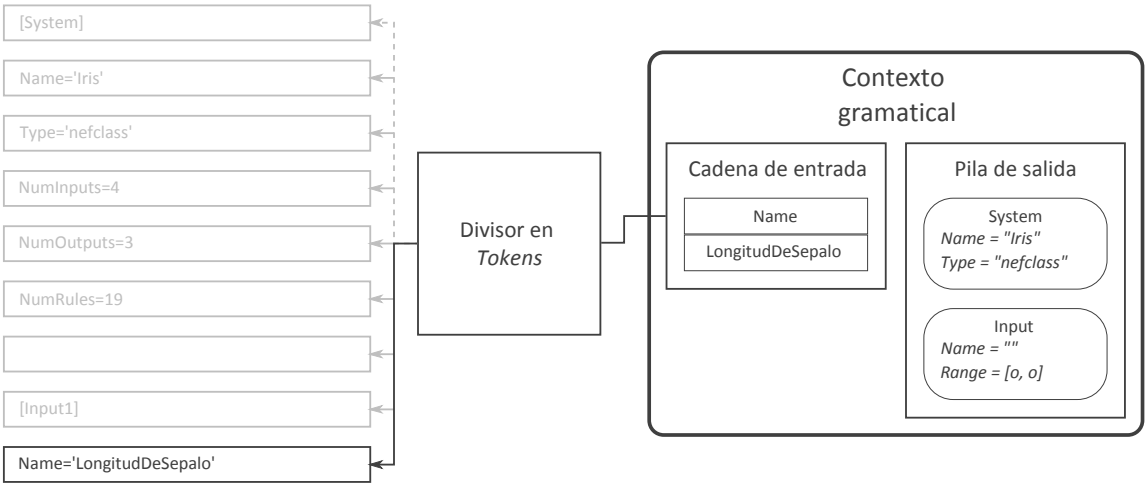
Figura 6. Entorno de la herramienta de manejo de archivos de entrada



Para el caso de este proyecto se creó una gramática capaz de manipular archivos de modelo difuso escritos en los lenguajes *Fuzzy Inference System* (FIS) y *Fuzzy Programming Language* (FPL). Se tomó el lenguaje FIS como base, debido a que la mayoría de las reglas de producción contienen un sólo símbolo terminal. Luego se extendió la herramienta para agregar compatibilidad con el lenguaje FPL, en el cual el nivel de abstracción de las entidades y su complejidad gramatical es mayor.

A continuación se describe el proceso de lectura del archivo de modelo difuso.

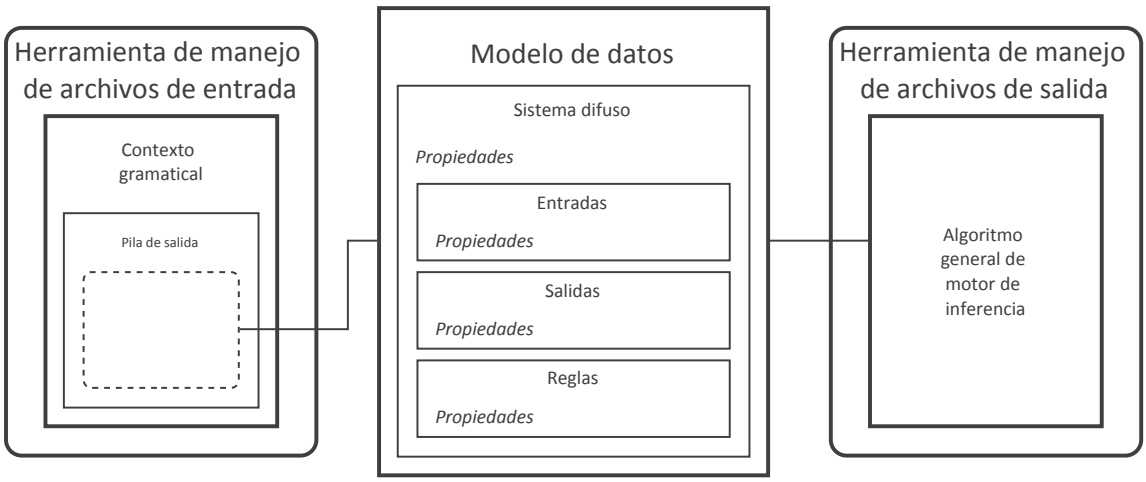
Figura 7. Operación de lectura de archivo de modelo difuso



Durante la ejecución del preprocesador, la expresión regular correspondiente al lenguaje del modelo difuso ingresado, divide las líneas de texto capturadas del archivo en *tokens*, los cuales son comparados contra las palabras clave de cada una de las expresiones de la gramática que definen el lenguaje y se crean entidades o se asignan propiedades en el modelo de datos.

2.3.2 Modelo de datos. El modelo de datos es la estructura que almacena temporalmente los datos capturados por la herramienta de manejo de archivos de entrada, los manipula y los dispone a la herramienta de manejo de archivos de salida para su escritura en el archivo de código (ver Figura 8).

Figura 8. Entorno del modelo de datos



Las clases del modelo de datos y sus propiedades representan las entidades y los métodos presentes en los diferentes tipos de sistemas difusos que se pueden interpretar, en este caso, tipo Mamdani, Sugeno, Tsukamoto y de tipo clasificador neurodifuso.

Los datos presentes en el modelo de datos deben ser alterados para que cumplan con los requisitos del sistema difuso del motor de inferencia a generar. El submódulo normalizador de sistema difuso (ver Figura 9), realiza los siguientes cambios: corregir el nombre de entidades anónimas o con caracteres inválidos en su nombre, corregir el nombre de métodos y tipos de función de membresía para ajustarlos al estándar del motor de inferencia y corregir la estructura de las reglas de acuerdo con el lenguaje del modelo y tipo de sistema difuso.

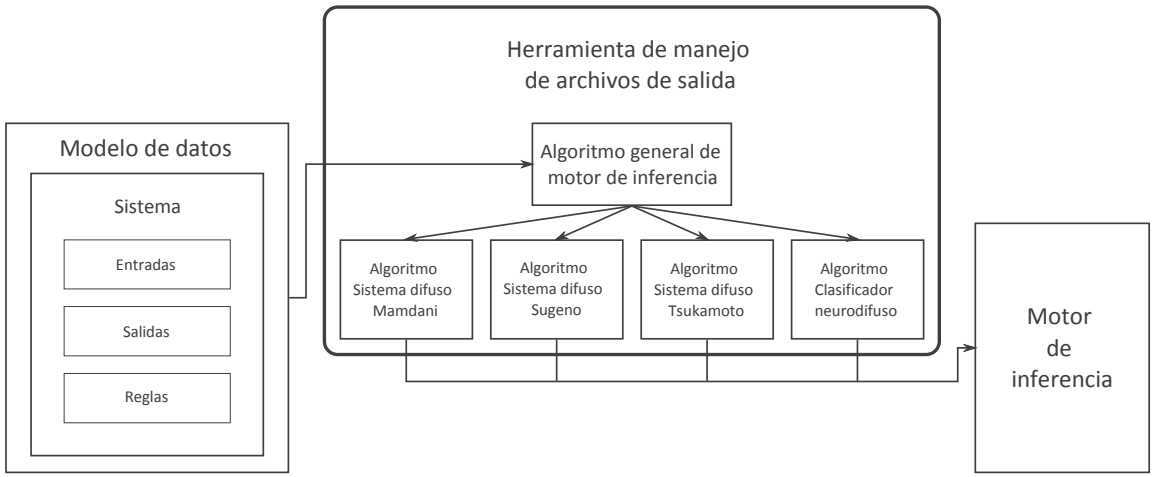
Figura 9. Proceso de corrección del sistema difuso

La normalización del sistema difuso adquiere relevancia cuando se tienen en cuenta para el preprocesamiento modelos heterogéneos con alto grado de variabilidad en sus parámetros.

El proceso de normalización entrega un modelo difuso a la herramienta de manejo de archivos de salida, apto para su registro en el archivo de motor de inferencia difusa.

2.3.3 Herramienta de manejo de archivos de salida. La herramienta de manejo de archivos de salida escribe en un archivo el código correspondiente al sistema difuso contenido en el modelo de datos. Este archivo resultante contiene el motor de inferencia que será cargado al dispositivo embebido.

Figura 10. Entorno de la herramienta de manejo de archivos de salida



La herramienta se diseñó con base en el patrón de diseño estrategia, el cual plantea el uso de un algoritmo generalizado compuesto por diferentes algoritmos concretos que especifican la funcionalidad final de acuerdo con los parámetros que reciba el algoritmo general.

En este proyecto, el algoritmo general indica los posibles procedimientos que puede realizar un algoritmo específico a modo de métodos abstractos, los algoritmos específicos definen las rutinas de los sistemas tipo Mamdani, Sugeno, Tsukamoto y clasificadores neurodifusos, indicando qué métodos se usan y el orden con que se escriben en el archivo de motor de inferencia.

La aplicación genera el código de motores de inferencia con base en las librerías Fuzzylite, para los sistemas difusos tipo Mamdani, Sugeno y Tsukamoto y la librería NEFCLASS-Q que permite la creación de clasificadores neurodifusos.

El código se presenta en lenguaje C++ el cual es ampliamente usado por dispositivos embebidos, incluyendo la plataforma TWR-K70F120M y tiene un alto nivel de compatibilidad con el lenguaje C.

El sistema leído del archivo de modelo difuso no sufre alteraciones significativas durante su paso por los módulos del preprocesador.

El sistema difuso se traslada desde la herramienta de manejo de archivos de entrada hasta el modelo de datos, donde el módulo normalizador de sistema difuso ajusta los valores de las propiedades del sistema. Posteriormente, el sistema pasa a la herramienta de manejo de archivos de salida, donde se junta a los componentes que definen los métodos de inferencia para conformar el archivo de motor de inferencia.

En los siguientes capítulos se describe en detalle la estructura y función de cada uno de los módulos del preprocesador FISPES.

3. HERRAMIENTA DE MANEJO DE ARCHIVOS DE ENTRADA

La herramienta de manejo de archivos de entrada es el módulo del procesador FISPES que se encarga de recibir como entrada de la operación del preprocesador, el archivo que contiene la definición del modelo difuso (ver 3.1.1 Modelo difuso). El patrón de diseño intérprete, define los componentes de la herramienta: El contexto gramatical y la gramática del lenguaje.

A continuación se describe la estructura y características de los archivos de modelo difuso, la estructura y funcionamiento del contexto gramatical y los componentes y proceso de construcción de la gramática.

3.1 ARCHIVO DE MODELO DIFUSO

La herramienta de manejo de archivos de entrada del preprocesador, toma como entrada un archivo de texto plano que contiene: la descripción de un modelo difuso, sus entradas, sus salidas y sus reglas. Es de interés para el desarrollo de este proyecto, preprocesar el archivo de modelo difuso planteado en el trabajo de grado “Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico”. El modelo se denomina en este proyecto como “Modelo difuso para CEC”.

3.1.1 Modelo difuso. Los modelos difusos soportados por el preprocesador FISPES son:

- Mamdani, Sugeno y Tsukamoto (modelos difusos tradicionales).
- Clasificador neurodifuso (modelo neurodifuso o NEFCLASS)

El *modelo difuso para CEC* que se utilizó en el desarrollo de este proyecto es un modelo de tipo clasificador neurodifuso.

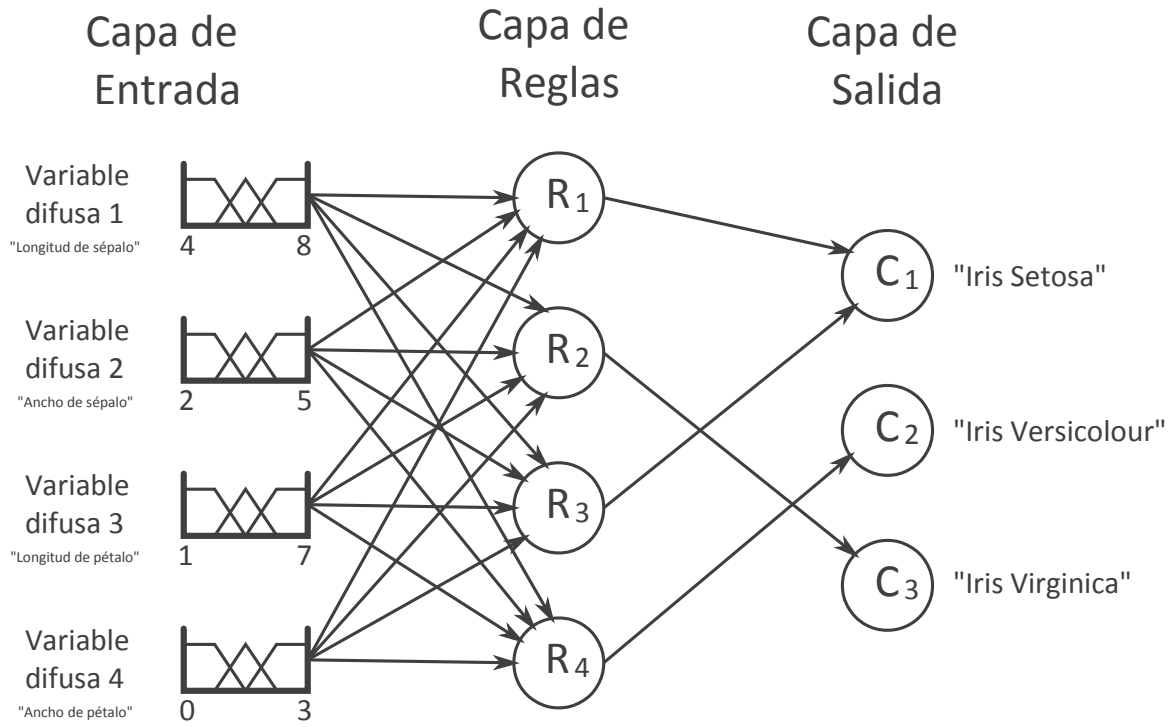
A continuación se describe, a modo de ilustración, el modelo neurodifuso del archivo de modelo llamado “Iris”, que sirve como ejemplo debido a su claridad, sencillez y a su reducido número de elementos. *Iris* se obtuvo como resultado del entrenamiento de una red neurodifusa usando como base de ejemplos la *base de datos de plantas Iris* y el software Nefclass-Q para el entrenamiento y validación de la red.

La base de datos de plantas Iris es un conjunto de datos, presentados por Ronald Fisher en 1936, que contiene las medidas de longitud de sépalo, ancho de sépalo,

longitud de pétalo y ancho de pétalo para 150 especímenes de plantas del género *Iris*. Consiste en 50 muestras de 3 especies de Iris relacionadas: Iris Setosa, Iris Virginica e Iris Versicolor.

La Figura 11 muestra el modelo neurodifuso generado por la red neurodifusa.

Figura 11. Modelo de clasificador neurodifuso (Modelo NEFCLASS)



El modelo NEFCLASS consiste en una red neuronal con variables difusas y capas de neuronas que representan las capas de entrada, de reglas y de salida en los modelos difusos tradicionales.

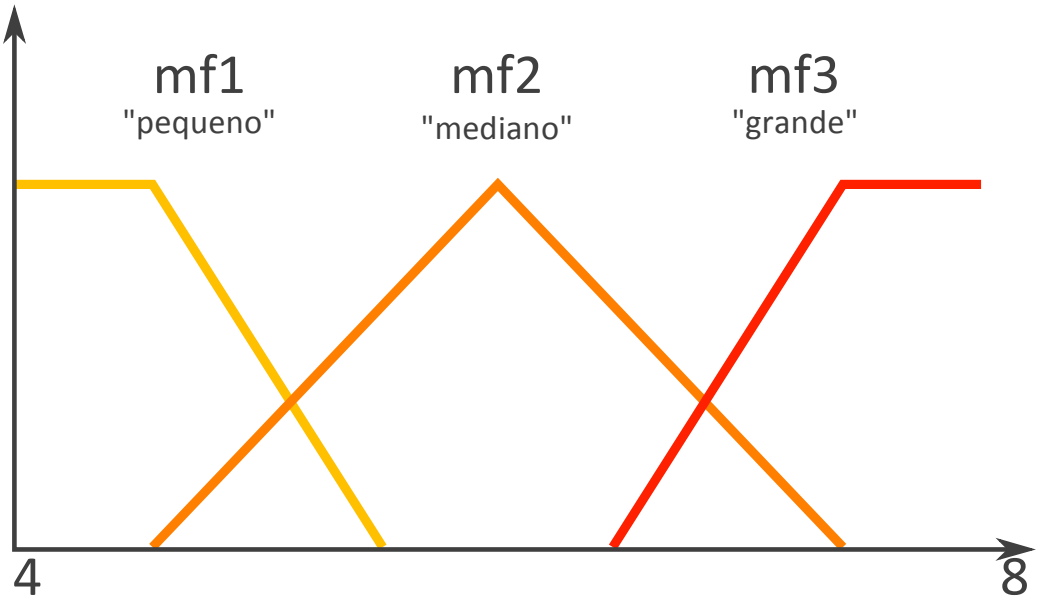
El modelo recibe como entrada una serie de datos llamada “patrón”. La ***capa de entrada*** cuenta con variables que se encargan de fusificar la entrada nítida del sistema y se conectan con la capa de reglas.

La ***capa de reglas*** conecta la capa de entrada con la de salida y permite que un patrón habilite un trayecto en la red neuronal hacia una de las clases en la capa de salida mediante la activación de algunas de sus neuronas de regla.

La ***capa de salida*** contiene las neuronas de clases y se encarga de entregar la salida, producto de la operación del sistema neurodifuso, a modo de una única clase seleccionada.

Las variables de entrada del modelo están compuestas por conjuntos difusos (o etiquetas lingüísticas) dispuestos en un rango numérico. Los conjuntos están delimitados por funciones de membresía, funciones de dos dimensiones cuyos valores en el eje Y oscilan entre 0 y 1 (ver Figura 12).

Figura 12. Conjunto de la *Variable difusa 1* del modelo NEFCLASS



Las reglas del modelo están conformadas por antecedente, consecuente, operadores y peso:

- El antecedente contiene cláusulas que constan del nombre de una variable y uno de sus conjuntos e indica las condiciones que se deben cumplir para que se active una regla.
- El consecuente indica la acción que se efectúa cuando se activa la regla. Para los sistemas neurodifusos el consecuente indica la clase que selecciona la regla.

- Los operadores son palabras que unen las cláusulas del antecedente y representan el método con el que se procesará cada clausula, “And” (Norma-T) y “Or” (Norma-S).
- El peso es un multiplicador de la activación de la regla.

En la Figura 13 se observan algunas de las reglas del modelo neurodifuso IRIS.

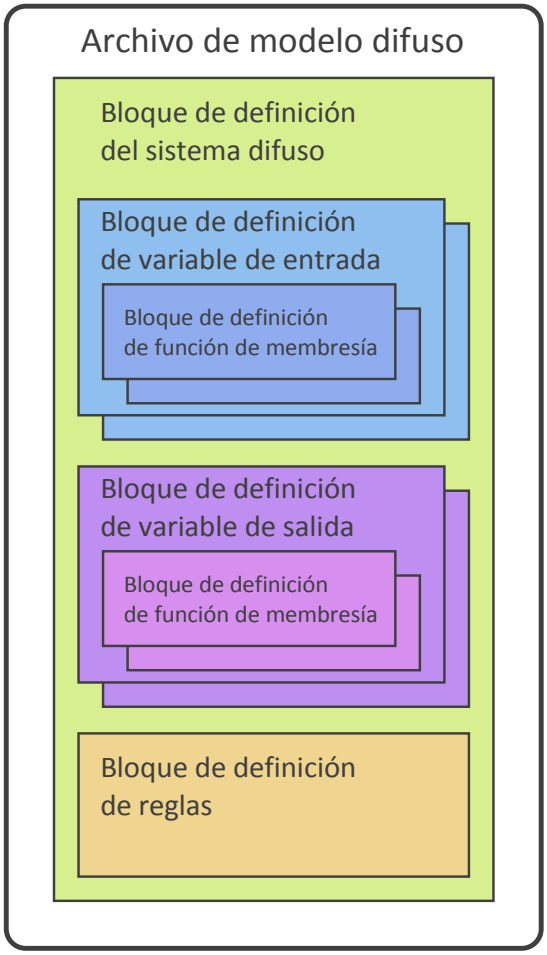
Figura 13. Reglas del modelo NEFCLASS

```
SI (longitudDeSepalo ES pequeno) Y (anchoDeSepalo ES mediano) Y (longitudDelPetalo ES pequeno) Y
(anchoDelPetalo ES pequeno) ENTONCES Clase ES irisSetosa

SI (longitudDeSepalo ES grande) Y (anchoDeSepalo ES mediano) Y (longitudDelPetalo ES mediano) Y
(anchoDelPetalo ES mediano) ENTONCES Clase ES irisVersicolor

SI (longitudDeSepalo ES mediano) Y (anchoDeSepalo ES mediano) Y (longitudDelPetalo ES grande) Y
(anchoDelPetalo ES grande) ENTONCES Clase ES irisVirginica
```

El archivo de modelo difuso puede estar escrito en diferentes lenguajes, tales como Fuzzy Inference System, Fuzzy Programming Language, Fuzzy Control Language, FuzzyLite Language, entre otros, lo que significa que la sintaxis y semántica puede variar entre archivos para un mismo modelo. El Preprocesador FISPES soporta por defecto los lenguajes FIS y FPL.

A continuación se realiza la descripción del archivo de modelo neurodifuso escrito en lenguaje FIS. En la Figura 14 se aprecian los elementos que conforman un archivo de modelo difuso.

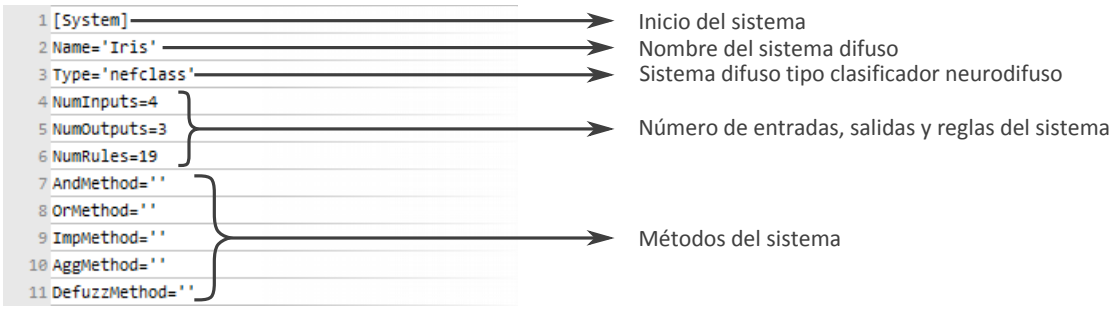
Figura 14. Componentes del archivo de modelo difuso



Se describen a continuación cada uno de los bloques que componen el archivo de modelo difuso.

3.1.2 Bloque de definición del sistema difuso

Figura 15. Bloque de definición del sistema difuso



Línea 1: Indica el inicio de la descripción del modelo difuso en la Figura 11.

Línea 2: Indica el nombre que se le asigna a la entidad sistema, el nombre puede contener letras y números.

Línea 3: Define el tipo de sistema difuso, por ejemplo, mamdani, sugeno, tsukamoto o nefclass.

Líneas 4 a 6: Indica la cantidad de variables difusas de entrada, de salida y de reglas en el modelo. En la Figura 11 representa las variables en la capa de entrada (parte izquierda de la imagen), las neuronas en la capa de salida (parte derecha de la imagen) y las neuronas en la capa de reglas (parte central de la imagen). Estos valores señalan cuantos bloques de entradas y salidas y cuantas reglas deben aparecer en el archivo.

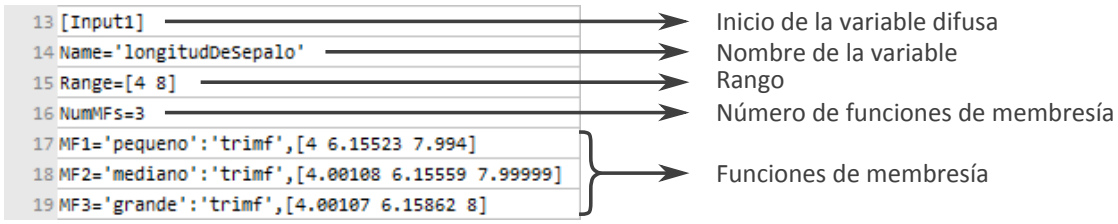
Líneas 7 a 11: Indican los métodos que usa el sistema difuso para las operaciones de inferencia: los operadores del antecedente en las reglas (“min”, “max”), los métodos de implicación y agregación para el procesamiento de las reglas (“min”, “max”) y el método de defusificación para obtener la salida nítida del motor de inferencia (“centroid”, “mom”, etc.).
Los clasificadores neurodifusos usan métodos predeterminados, por lo que no es necesario incluir estas líneas en el archivo de modelo difuso.

El *bloque de definición del sistema difuso* contiene la información del sistema que representa el modelo difuso. Aquí se definen las propiedades de nombre y tipo de sistema difuso, cantidad de variables, cantidad de reglas y los métodos que usa el sistema. En este bloque se definen también las entradas, salidas y reglas.

Las líneas siguientes a la primera línea especifican propiedades o entidades pertinentes al modelo difuso. Antes de esta línea es posible ubicar comentarios o metainformación del modelo difuso.

3.1.3 Bloque de definición de variable de entrada. El *bloque de definición de variable de entrada* describe una entrada del sistema difuso; las propiedades (nombre, tipo, rango) y las funciones de membresía asociadas a esta entrada.

Figura 16. Bloque de definición de la Variable difusa 1



Línea 13: Define la creación de la *Variable difusa 1* en el sistema difuso (ver Figura 11).

Línea 14: Define el nombre de la *Variable difusa 1*.

Línea 15: Asigna los valores de rango de la variable. En la Figura 11 el rango se encuentra bajo la figura de la *Variable difusa 1*.
Los valores se organizan entre corchetes cuadrados y separados por espacios para simbolizar un arreglo de datos.

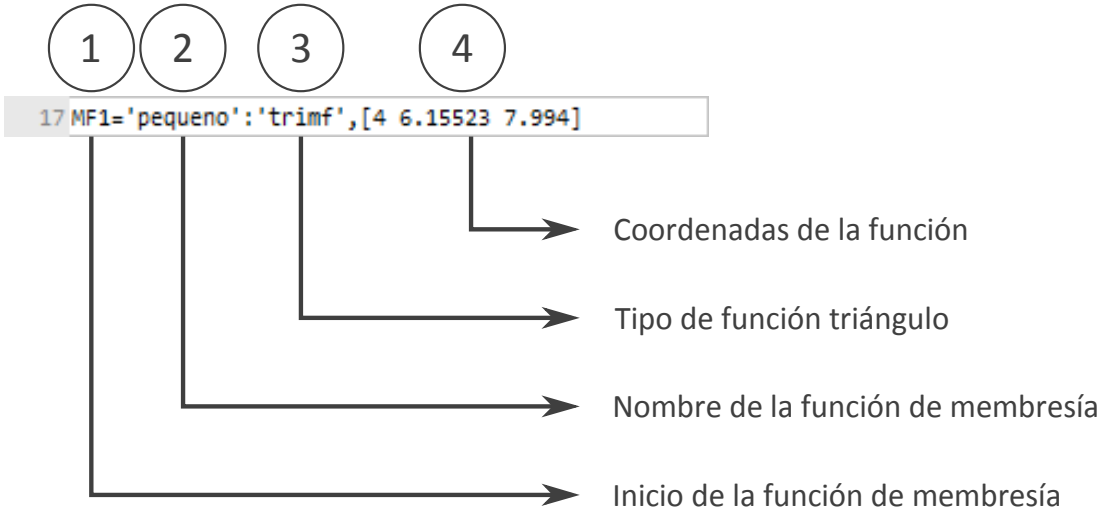
Línea 16: Indica que la *Variable difusa 1* cuenta con 3 funciones de membresía (ver Figura 12).

Líneas 17 a 19: Denotan la creación de las funciones de membresía de la *Variable difusa 1*, las etiquetas lingüísticas “pequeno”, “mediano” y “grande” de izquierda a derecha en la Figura 12. Estas líneas son bloques de definición de función de membresía para los conjuntos de la *Variable difusa 1*.

A continuación se describen los componentes del bloque para el conjunto mf1 “pequeno”.

3.1.4 Bloque de definición de función de membresía. El *bloque de definición de función de membresía* contiene las propiedades de la función de membresía representada: nombre, tipo y parámetros o coordenadas (ver Figura 17).

Figura 17. Bloque de definición de función de membresía para el conjunto mf1 "pequeno"



- El segmento ① indica el inicio de la descripción del conjunto difuso *pequeno* de la Figura 12.
- El segmento ② Define el nombre del conjunto *mf1*, "pequeno".
- El segmento ③ Define el tipo de función de membresía triangular del conjunto *mf1*. Los conjuntos en los modelos neurodifusos pueden ser de dos tipos: "trapmf", función trapezoidal, si la función se encuentra en alguno de los dos costados del universo de discurso (shouldered) y "trimf", función triangular, si la función se ubica entre las funciones de los costados (middle).
- El segmento ④ Indica las coordenadas en el eje X para cada uno de los vértices de la función.

3.1.5 Bloque de definición de variable de salida. El *bloque de definición de variable de salida* sigue la misma estructura del bloque de definición de entrada en los sistemas de lógica difusa tradicional, en los cuales las variables de salida del sistema son universos de discurso que tienen asociadas funciones de membresía. En los clasificadores neurodifusos se recurre al uso de variables de salida de tipo "clase" en vez de universos de discurso (ver Figura 18).

Figura 18. Bloque de definición de neurona de salida C_1 del modelo neurodifuso

Línea 45: Define la creación de la neurona de clase *C_1*, "Iris Setosa" en el sistema difuso (parte derecha de la Figura 11).

Línea 46: Define el nombre de la neurona de clase *C_1*.

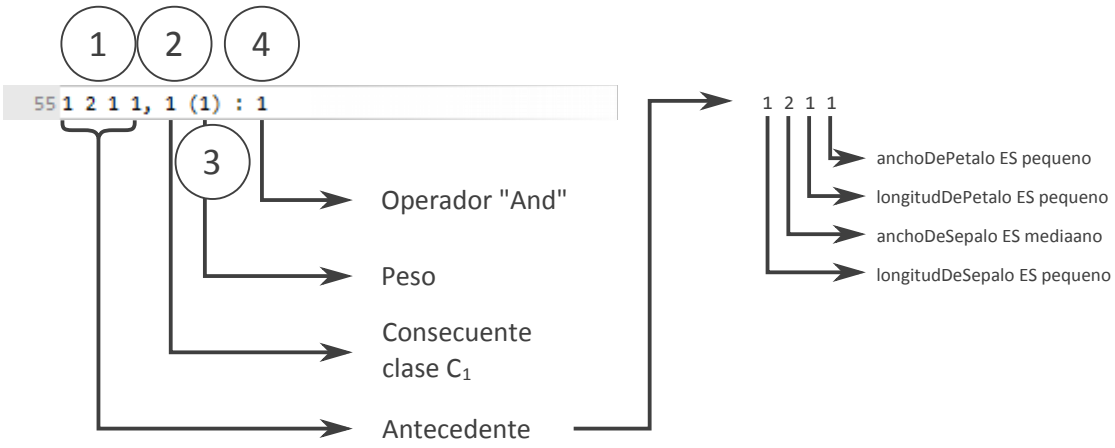
3.1.6 Bloque de definición de reglas. El ***bloque de definición de reglas*** contiene todas las reglas pertenecientes al sistema difuso modelado.

Figura 19. Bloque de definición de reglas

Línea 54: Indica el inicio de la descripción del bloque de reglas en el modelo difuso (Capa de reglas en la Figura 11).

Líneas 55 a 57: Indica la creación de las neuronas de regla R_1, R_2 y R_3 en el sistema difuso (parte central de la Figura 11). Representan las reglas en la Figura 13.
La definición de una regla está conformada por 4 segmentos, los cuales se describen a continuación.

Figura 20. Estructura de la regla R_1



- El segmento ① es el antecedente de la regla. En la Figura 13 representa las cláusulas "longitudDeSepalo ES pequeno", "anchoDeSepalo ES mediano", "longitudDelPetalo ES pequeno" y "anchoDelPetalo ES pequeno" de la primera regla. Los valores indican qué conjunto se selecciona por cada variable de entrada.
- El segmento ② es el consecuente de la regla R_1. Representa la cláusula "Class IS irisSetosa" en la primera regla de la Figura 13.
- El segmento ③ representa el peso de la regla R_1.
- El segmento ④ es el operador u operadores, Representa la palabra "AND" que enlaza las cláusulas del antecedente de la primera regla R_1 en la Figura 13.

La información sobre el sistema difuso contenida en el archivo de modelo es recolectada por los elementos de la herramienta de manejo de archivos de entrada: el contexto gramatical y la gramática.

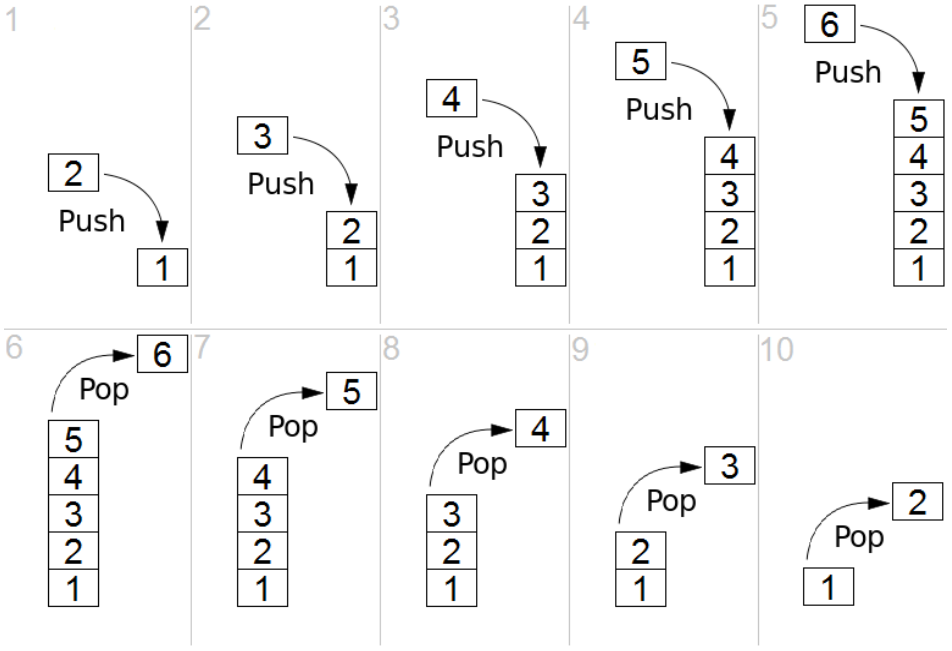
3.2 CONTEXTO GRAMATICAL

El contexto gramatical es uno de los elementos definidos en el patrón de diseño intérprete, encargado de almacenar temporalmente la información generada durante la actividad de lectura del archivo de modelo difuso.

Está conformado por una clase que contiene dos atributos: la cadena de entrada y la pila de salida.

La cadena de entrada está implementada como un arreglo o vector de cadenas de texto (string) que almacena los *tokens* generados en la ejecución del procesador al leer una línea del archivo de modelo difuso. La pila de salida está constituida por un arreglo de tipo “Stack” o pila, el cual permite la inserción y el retiro de elementos a modo LIFO “Last In, First Out” (ver Figura 21).

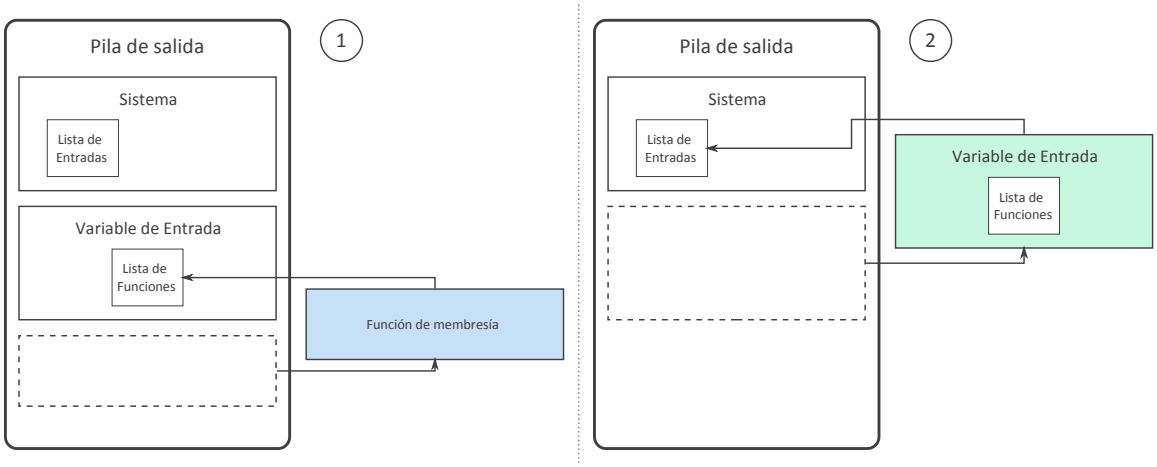
Figura 21. Estructura de datos tipo pila



Autor: Usuario Maxtremus en Wikipedia en inglés, consultado en Julio de 2016.

La pila recibe las entidades creadas al interpretar la cadena de entrada con la gramática, las cuales se organizan en la pila de forma tal, que al momento de ser creadas ocupan un nuevo espacio en la pila, permitiendo la lectura de las consecuentes líneas del modelo que contienen sus propiedades. Una vez asignados todos los valores a la entidad, esta se expulsa de la pila y se asigna a una lista contenedora de la entidad que se encuentre ubicada en la parte superior de la pila. De este modo se organizan las entidades en la jerarquía del sistema difuso (ver Figura 22).

Figura 22. Asignación de una variable al sistema difuso en la pila de salida



El modelo difuso contenido en la pila de salida al terminar la operación de lectura del archivo difuso es enviado al modelo de datos del procesador para su tratamiento y posterior escritura en el archivo de motor de inferencia difusa.

3.3 GRAMÁTICA

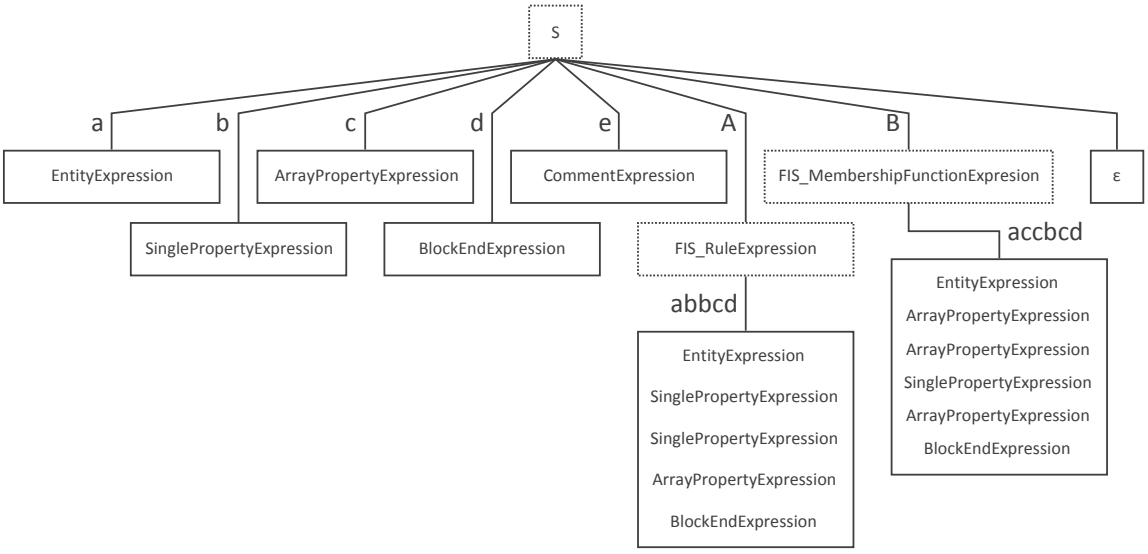
La gramática o árbol de sintaxis está conformada por elementos que representan las expresiones del lenguaje que se dispone a interpretar. Hace parte de los elementos propuestos por el patrón de diseño intérprete. En el preprocesador FISPES está conformada por un conjunto de objetos que se encargan de reconocer el lenguaje del modelo difuso, sea FIS o FPL. Los objetos de la gramática representan las expresiones terminales y no terminales que componen el lenguaje.

La gramática se crea al inicio del preprocesamiento del archivo de modelo a partir de clases de expresión de la *herramienta de manejo de archivos de entrada*,

seleccionadas de acuerdo a la información contenida en un archivo de configuración de lenguaje (ver Anexo F).

A continuación se describe la gramática que usa el preprocesador FISPES para interpretar los archivos de modelo difuso escritos en lenguaje FIS.

Figura 23. Gramática para el lenguaje FIS



La definición formal para la gramática del lenguaje FIS es ***G*** = *(N, Σ, P, S)*, tales que:

- **N** = {S, A, B}, es el conjunto de los símbolos no terminales.
- **Σ** = {a, b, c, d, e}, es el conjunto de símbolos terminales.
- **P**, representa las reglas de producción, consiste en:
  - S → a | b | c | d | e | A | B | ε, donde ε representa una cadena vacía.
  - A → abbcd.
  - B → accbcd.

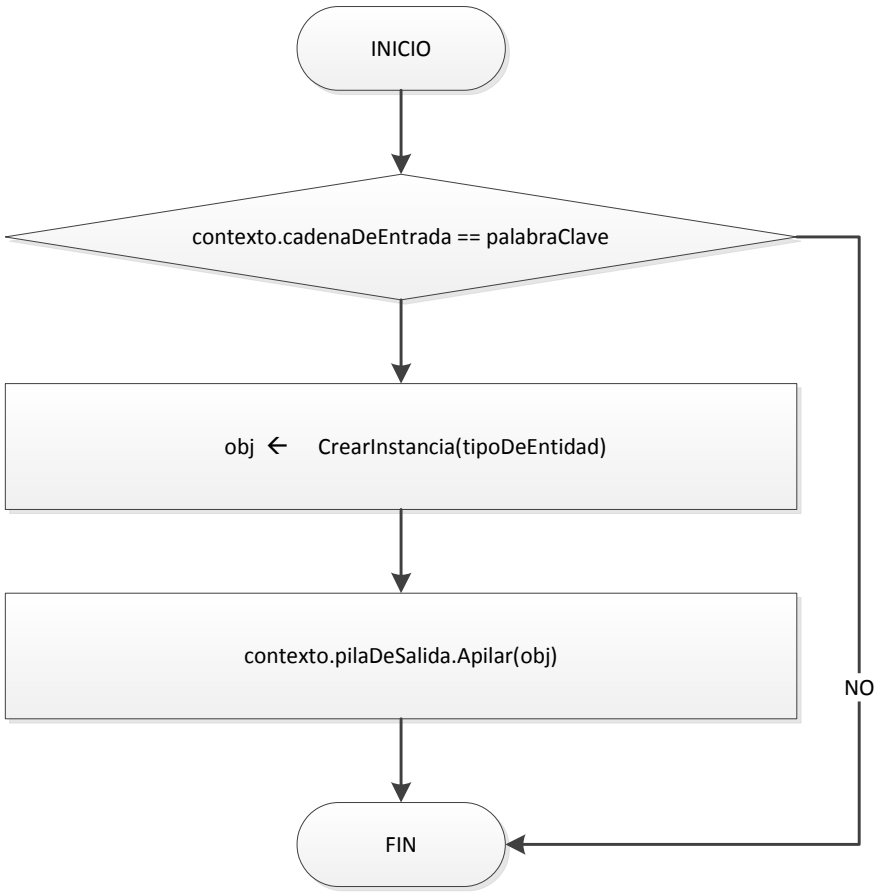
  P representa las aristas en Figura 23.
- **S** ∈ N, es el símbolo inicial (Parte superior en Figura 23).

A continuación se describen los dos tipos de expresiones del lenguaje.

3.3.1 Expresiones terminales. Las *expresiones terminales* contienen las instrucciones que debe ejecutar el preprocesador cuando se acepta la cadena de entrada del contexto gramatical.

Se muestra a continuación el procedimiento que sigue el preprocesador para crear una entidad al interpretar el contexto en la expresión terminal “EntityExpression”.

Figura 24. Interpretación del contexto en la expresión terminal de entidad “EntityExpresison”



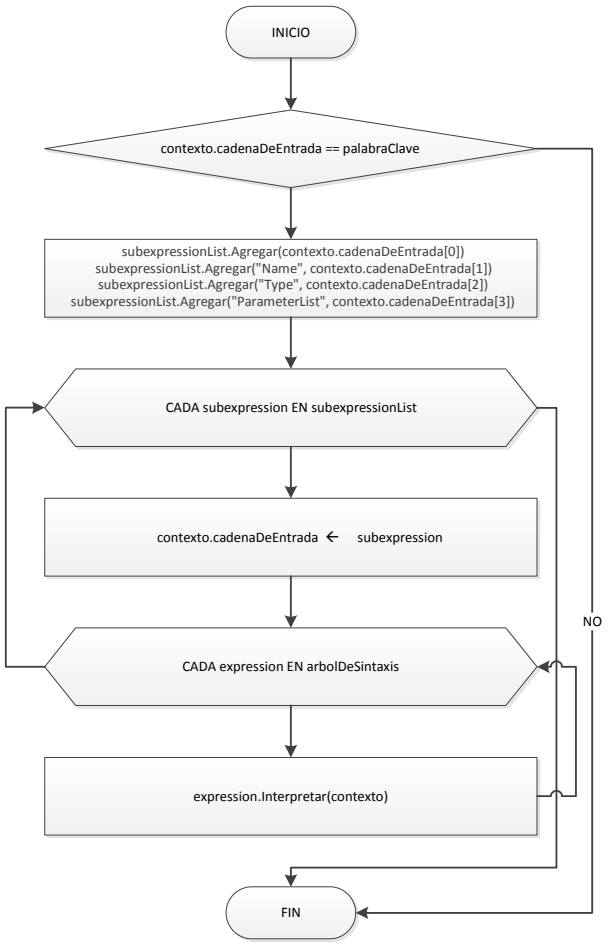
La cadena de entrada del contexto gramatical se compara con la palabra clave de la expresión *EntityExpresison* para determinar si la cadena es aceptada como una expresión de creación de entidad. Se crea el objeto (entidad) del tipo indicado por

la cadena de entrada y se envía a la parte superior de la pila de salida del contexto.

3.3.2 Expresiones no terminales. Las expresiones no terminales poseen como atributo un nuevo árbol de sintaxis, donde se almacenan las expresiones en las que deriva esta expresión no terminal según las reglas de producción del lenguaje.

En la Figura 25 se muestra el procedimiento que realiza el preprocesador para crear una entidad de tipo función de membresía al interpretar el contexto en la expresión no terminal “FIS_MembershipFunctionExpression”.

Figura 25. Interpretación de la cadena de entrada en la expresión no terminal “FIS_MembershipFunctionExpression”



La cadena de entrada del contexto gramatical se compara con la palabra clave de la expresión *FIS_MembershipFunctionExpression* para determinar si la cadena es aceptada como una expresión de función de membresía. En el árbol de sintaxis de *FIS_MembershipFunctionExpression* se añaden los *tokens* de la cadena de entrada como subexpresiones. Se toma cada una de las subexpresiones como una nueva línea en la cadena de entrada del contexto y se itera sobre cada una de las expresiones en la gramática para interpretarla.

En operación conjunta, el contexto gramatical y la gramática extraen la información contenida en el archivo de modelo y disponen el sistema difuso creado al modelo de datos del preprocesador.

4. MODELO DE DATOS

El modelo de datos es el módulo del preprocesador FISPES que se encarga del almacenamiento temporal la información capturada por la herramienta de manejo de archivos de entrada. Cuenta con un submódulo, el normalizador de sistema difuso, que posee la habilidad de manipular los datos, permitiendo la correcta escritura de los sistemas independientemente de sus características en el modelo difuso.

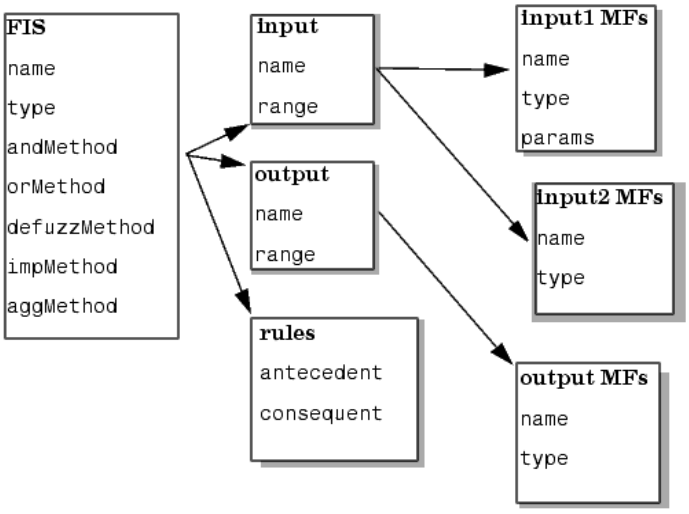
El modelo de datos dispone el sistema difuso normalizado a la herramienta de manejo de archivos de salida para su escritura en el archivo de código.

Se describen a continuación las entidades difusas que componen el modelo de datos y el funcionamiento del submódulo normalizador de sistemas difusos.

4.1 ENTIDADES DEL MODELO DE DATOS

El modelo de datos está conformado por clases que representan las entidades de los modelos difusos. Estas clases cuentan con atributos equivalentes a las propiedades de las entidades (ver Figura 26) y métodos para gestionar la asignación de valores a los atributos de tipo lista contenedora.

Figura 26. Entidades del modelo difuso definido por el lenguaje FIS



Fuente: The MathWorks, Inc., Matlab R2016a Documentation.

La jerarquía de clases del modelo de datos está compuesta por los siguientes elementos:

La clase abstracta “FLEntity” es la clase superior de la jerarquía del modelo (ver Anexo C) y permite que sus subclases hereden las propiedades “Name”, el nombre de la entidad y “Type”, el tipo específico para esa entidad.

La clase “FLSystem” se encarga de contener los parámetros del sistema difuso y sus componentes (entradas, reglas y salidas o clases). Sus atributos contemplan el almacenamiento de los siguientes valores:

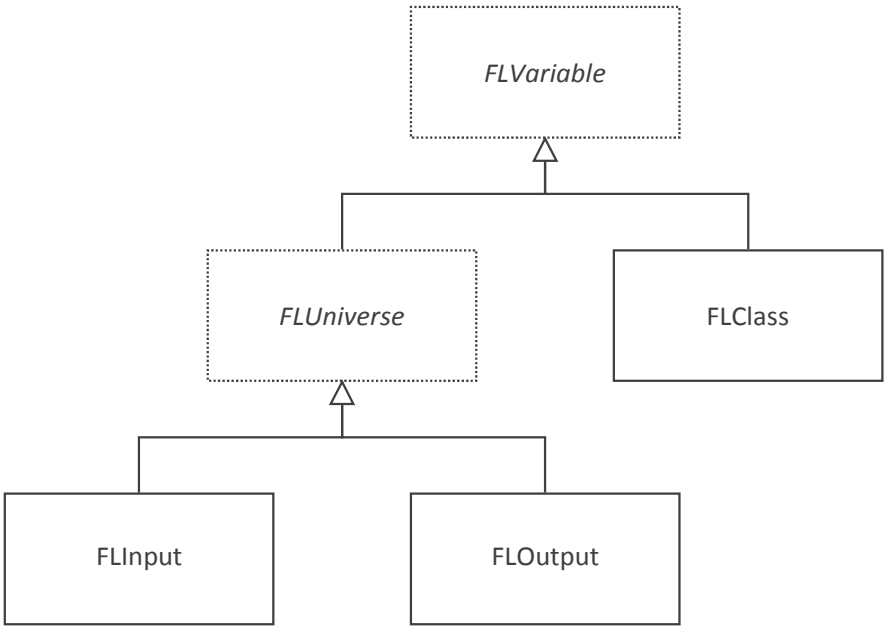
- “AndMethod”, método a usar para el operador “And” durante la evaluación de las reglas.
- “OrMethod”, método para la operación “Or” en las reglas.
- “ImplicationMethod”, método de la implicación (Norma-T).
- “AggregationMethod”, el método de agregación (Norma-S).
- “DefuzzificationMethod”, método de defusificación.
- “InputList”, lista de entradas.
- “OutputList”, lista de salidas.
- “UniverseList”, lista de entradas/salidas cuando hay entidades de conexión (FLConnection) en el modelo difuso.
- “RuleBase”, lista de reglas.
- “ConnectionList”, lista de conexiones en el sistema difuso que indica cuales entidades en “UniverseList” son entradas y cuales son salidas.

Sus métodos permiten manejar los datos que se agregan o recogen de las listas contenedoras: “Add” agrega entradas, salidas y reglas a sus respectivas listas contenedoras dependiendo de su tipo de entidad. “TryGetUniverse” y “TryGetClass” devuelven universos y clases respectivamente al ser buscadas y encontradas por su nombre en su respectiva lista contenedora.

La clase “FLVariable” y sus subclases contienen los datos de las variables del modelo difuso. La clase abstracta “FLVariable” permite crear una variable difusa sin definir el tipo de variable que ha de representar en el sistema, es decir, no especifica si es una variable de entrada, de salida o una clase, esto permite realizar búsquedas en las listas contenedoras que contienen diferentes entidades dependiendo del tipo de modelo difuso, por ejemplo: la lista de salidas puede contener variables de tipo “FLOutput” o ”FLClass” si se preprocesan modelos difusos o clasificadores neurodifusos respectivamente.

La jerarquía de clases de variables define los atributos que tienen las entidades de tipo variable (ver Figura 27).

Figura 27. Jerarquía de clases para las entidades de tipo variable



La clase abstracta “FLUniverse” contiene los atributos que almacenan los valores de rango “Range” o “Minimum” y “Maximum” y la lista de funciones de membresía. Además, define las operaciones para agregar “Add” y recoger “TryGetMembershipFunction” las funciones de membresía ubicadas en la lista.

Las clases “FLInput” y ”FLOutput” tienen los atributos heredados de la clase “FLUniverse”.

La clase “FLClass” posee únicamente los atributos heredados de la clase “FLVariable”.

Las clases “FLInput”, “FLOutput” y “FLClass” tienen su tipo de entidad definido y son a estas a las que se refiere el algoritmo de escritura en la herramienta de manejo de archivos de salida.

La clase “FLMembershipFunction” tiene los atributos que almacenan los datos de funciones de membresía de variables en el modelo difuso. Sus atributos cumplen las siguientes funciones:

- “ParameterList” almacena los datos de los parámetros que construyen la función de membresía, por ejemplo, si la función es una curva de Gauss, en la lista se almacenan los parámetros “σ” sigma y “c” de la función.
- “XYPointList” almacena las coordenadas de la función de membresía. Si la función está conformada por segmentos de líneas rectas, se almacenan las coordenadas de cada vértice de la función.

Los parámetros de la función se almacenan en cada lista dependiendo del tipo de función de membresía especificado en el modelo difuso.

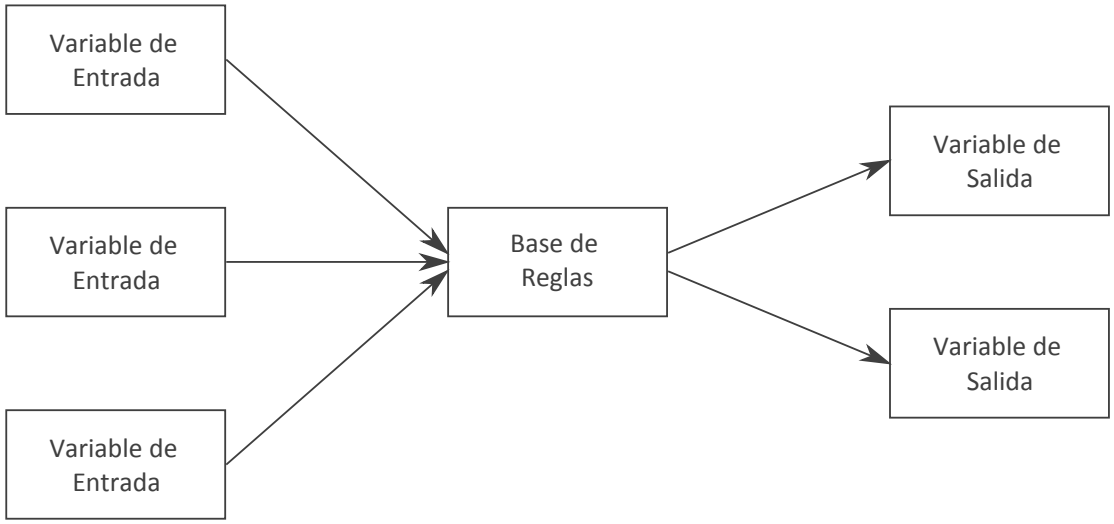
La clase “FLRule” contiene los atributos que alojan los datos de reglas definidas en el modelo difuso. Posee los siguientes atributos:

- “Weight”, el peso que se aplica a la regla.
- “AntecedentList”, lista que contiene los nombres o índices de los conjuntos difusos mencionados en las cláusulas del antecedente de la regla. El orden en que se encuentran ordenados en la lista determina a qué variable pertenece el conjunto.
- “ConsequentList”, lista que almacena los nombres o índices de los conjuntos presentes en las cláusulas del consecuente de la regla, o si el sistema difuso es de tipo clasificador neurodifuso, el índice o nombre de la clase indicada en el consecuente.
- “OperatorList”, lista que contiene los operadores que conectan las cláusulas del antecedente.

La clase “FLConnection” se utiliza cuando se preprocesan modelos difusos escritos en lenguaje FPL. Estos contienen entidades de tipo conexión que definen el tipo de variable, entrada o salida/clase, para cada variable del modelo mediante la dirección de la conexión.

Las entidades de tipo variable desde donde se realiza la conexión son de tipo entrada y desde estas se realiza la conexión hacia la base de reglas. Las variables hacia donde se efectúa la conexión desde la base de reglas son de tipo salida (ver Figura 28).

Figura 28. Conexiones en modelos difusos en lenguaje FPL



La clase posee los atributos que permiten almacenar los datos que indican desde qué variable se realiza la conexión “FromEntity” y hacia qué variable se realiza “ToEntity”.

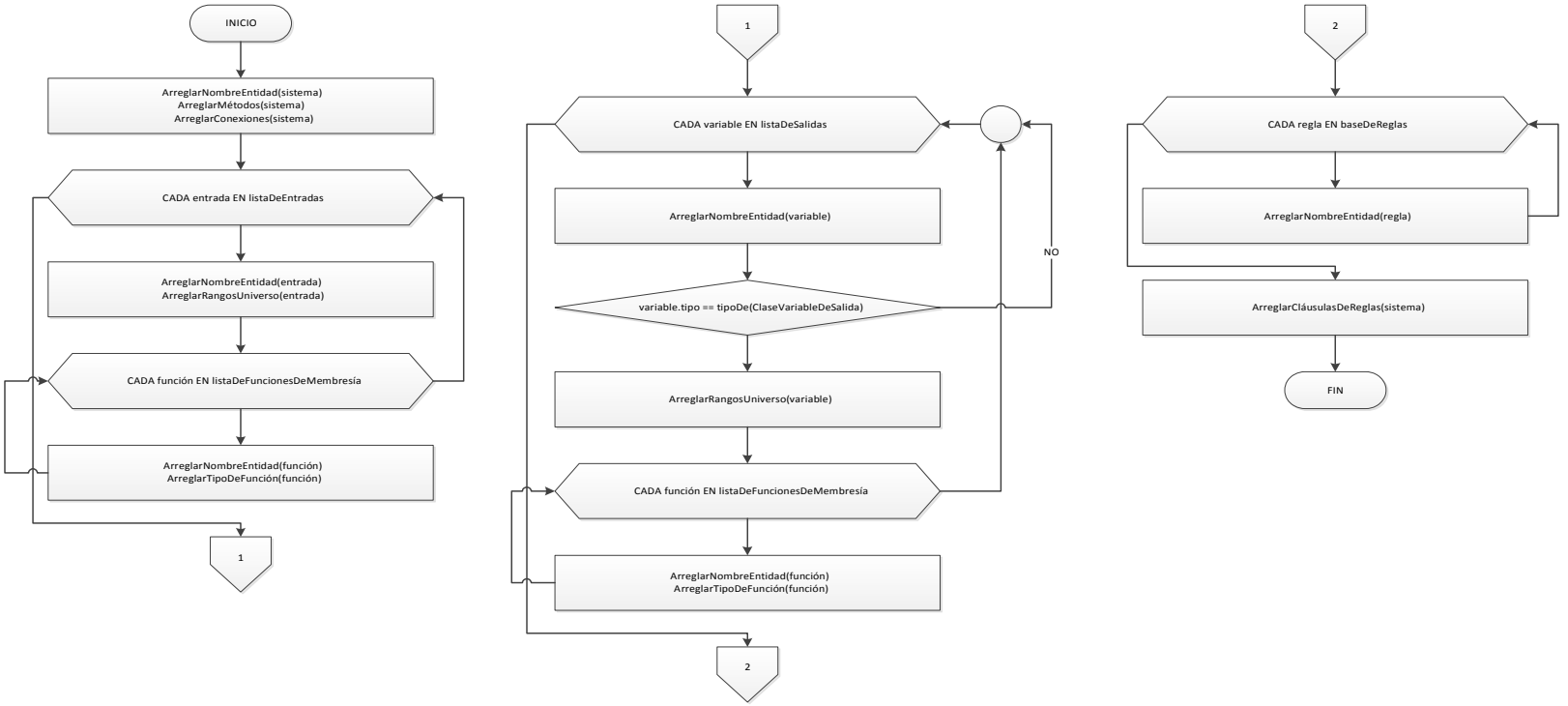
Cuando se ha consolidado el sistema difuso en el modelo de datos al finalizar la operación de lectura del archivo de modelo difuso, el preprocesador FISPES procede a realizar la normalización del sistema difuso.

4.2 SUBMÓDULO NORMALIZADOR DE SISTEMA DIFUSO

El modelo de datos cuenta con un submódulo denominado “normalizador de sistema difuso” que consiste en una clase, *Normalizer*, que realiza correcciones al sistema difuso contenido en el *modelo de datos* antes de enviarse a la *herramienta de manejo de archivos de salida*.

La normalización del sistema difuso es una parte importante del proceso de preprocesamiento debido a las diferencias en las diversas características de los modelos difusos pueden generar inconsistencias durante la escritura del archivo de modelo difuso por parte de la herramienta de manejo de archivos de salida. A continuación se describen las funciones que desempeña el submódulo normalizador para lograr la coherencia en los modelos difusos.

Figura 29. Operaciones del submódulo normalizador de sistema difuso



Los métodos de la clase *Normalizer* y sus funciones son los siguientes:

- "fixEntityName(FLEntity)", corrige el nombre de entidades difusas anónimas o que contengan caracteres inválidos.
- "fixUniverseRange(FLUniverse)", Asigna los valores máximo y mínimo de un universo, que se encuentran almacenados en los atributos "Maximum"y "Minimum" correspondientemente, a su propiedad "Range", esta operación se realiza porque en la escritura del motor de inferencia, la herramienta de manejo de archivos de salida sólo toma los valores contenidos en este último atributo.
- "fixMethods(FLSystem)", corrige los nombres de los métodos del sistema difuso para hacerlos compatibles con los nombres de métodos en las librerías.
- "fixMembershipFunctionTypes(FLMembershipFunction)", corrige los nombres de los tipos de función de membresía del sistema difuso para hacerlos compatibles con los nombres de los tipos de función de membresía en las librerías.
- "fixRuleClauses(FLSystem)", corrige la estructura de las reglas del sistema difuso de acuerdo con el lenguaje en el que se describió el modelo difuso y el tipo de sistema:
    - Para los sistemas neurodifusos con FIS disminuye en 1 el índice de todos los conjuntos en la lista de antecedentes, de este modo se evita la ambigüedad sobre el índice del primer elemento en la lista.
    - Para los sistemas neurodifusos con FPL recupera los nombres de las entradas, sus funciones de membresía y las clases para agregar sus índices a la lista de antecedentes y consecuentes.
    - Para los sistemas difusos tradicionales con FIS disminuye en 1 el índice de los conjuntos en el antecedente, recupera los nombres de las entradas, sus funciones de membresía y las clases para agregar sus índices a la lista de antecedentes y consecuentes, recupera nombres de operaciones a partir del valor numérico de la operación en la regla.
- "fixConnections(FLSystem)", Realiza la asignación de universos a las listas contenedoras de entidades de entrada y salida dependiendo de las conexiones especificadas en el modelo difuso.

Terminada la operación de normalización, el sistema difuso es transferido a la herramienta de manejo de archivos de salida, la cual genera el motor de inferencia difusa correspondiente al sistema y las librerías que brindan soporte a las operaciones del motor.

5. HERRAMIENTA DE MANEJO DE ARCHIVOS DE SALIDA

La herramienta de manejo de archivos de salida es el módulo del preprocesador FISPES que se encarga de la escritura del sistema difuso, recibido del modelo de datos, en el archivo de código del motor de inferencia difusa junto a las librerías que prestan soporte al respectivo sistema difuso.

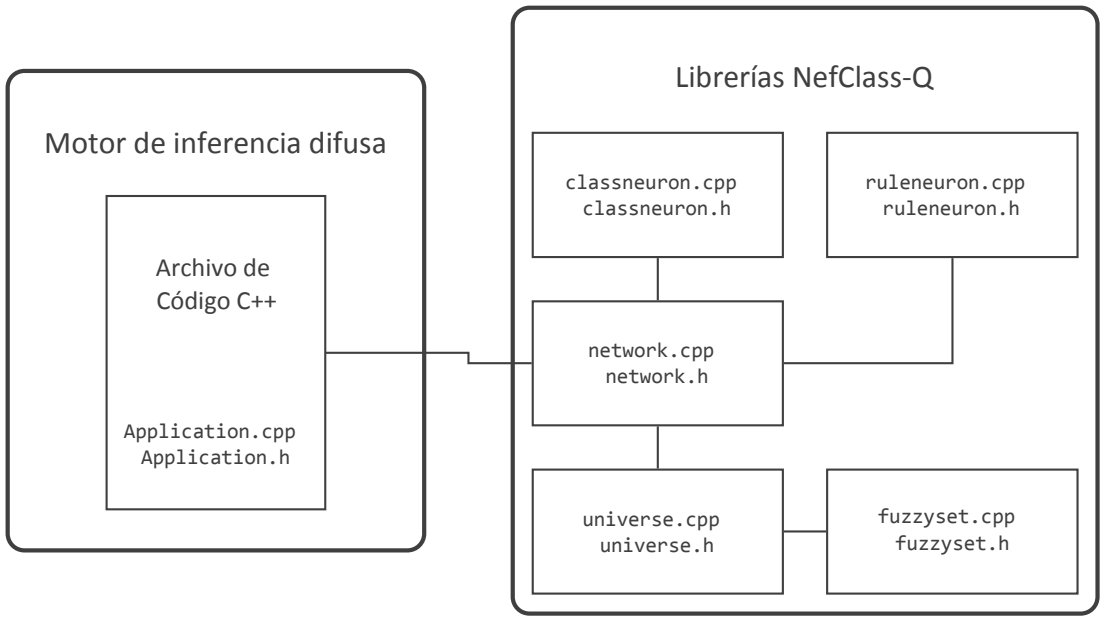
Se describen a continuación los elementos que componen la salida del preprocesador FISPES: el archivo de motor de inferencia difusa y las librerías acompañantes. Así mismo se describen los elementos que componen el módulo: el algoritmo general y los algoritmos específicos.

5.1 MOTOR DE INFERENCIA DIFUSA

La *herramienta de manejo de archivos de salida* genera un archivo de código C++ que contiene el motor de inferencia y una copia de las librerías que soportan el sistema difuso. Por defecto el procesador FISPES ofrece las librerías FuzzyLite y Nefclass-Q, para sistemas difusos tradicionales y clasificadores neurodifusos respectivamente.

Se muestran a continuación los elementos que constituyen las salidas del preprocesador.

Figura 30. Archivo de motor de inferencia y Librerías



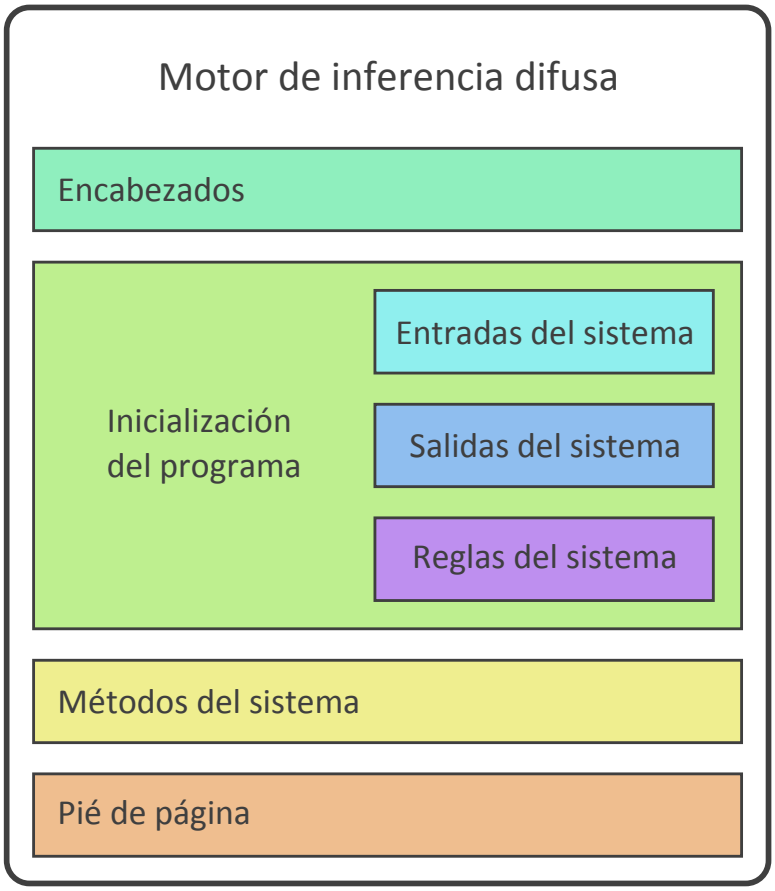
Tanto el motor de inferencia difusa como las librerías de soporte consisten en conjuntos de archivos de código y encabezados, caracterizados por sus extensiones de archivo “cpp” y “h” respectivamente. Los archivos de código contienen instrucciones y métodos para realizar las operaciones de inferencia y los archivos de encabezado presentan la estructura del sistema difuso.

Las librerías Nefclass-Q son un grupo de 5 clases que representan las entidades de los modelos neurodifusos y definen sus operaciones en el proceso de inferencia. Cada componente de la librería representa un elemento del modelo difuso (ver 3.1.1 Modelo difuso):

- Las clases *universe* y *fuzzyset* representan los elementos de la capa de entrada del modelo: los universos de discurso (parte izquierda en Figura 11) y sus conjuntos difusos o etiquetas lingüísticas (Figura 12).
- La clase *ruleneuron* representa la capa de reglas del modelo (parte central en Figura 11).
- La clase *classneuron*, representa la capa de salida que contiene las variables de salida, o clases, del modelo difuso (parte derecha en Figura 11).
- La clase *network* representa el modelo difuso en su totalidad y refiere a todos los demás elementos de la librería para conformar el sistema neurodifuso.

El archivo de código del motor de inferencia está compuesto por los bloques que se describen a continuación.

Figura 31. Estructura del motor de inferencia



- En el *bloque de encabezados* se incluyen las librerías estándar (C++ Standard Library) que prestan soporte a los tipos de archivo y operaciones propias del lenguaje C++.
- El bloque de inicialización del programa crea los objetos de las entradas, salidas y reglas del sistema difuso.
- El bloque de los métodos del sistema contiene las instrucciones para aplicar cada método del sistema difuso. Contiene además el método de captura de los datos de entrada del motor de inferencia.

- El pie de página contiene las instrucciones que se ejecutan antes de terminar la operación del motor de inferencia. Si en el algoritmo de motor de inferencia se definió la operación continua, las instrucciones de este segmento no se ejecutan.

Los bloques del motor de inferencia difusa son creados y ensamblados en el archivo de motor de inferencia por los algoritmos de sistema difuso a través de la ejecución de cada uno de sus métodos (pasos del algoritmo). La organización de los bloques en el archivo de motor de inferencia no es estricta y varía entre los motores de inferencia generados con las dos librerías predeterminadas del preprocesador FISPES.

A continuación se describen los algoritmos de la herramienta de manejo de archivos de salida que generan el motor de inferencia.

5.2 ALGORITMO GENERAL DE MOTOR DE INFERENCIA

El algoritmo general presenta los procesos requeridos para implementar un algoritmo específico. Está definido en la clase abstracta “GeneralAlgorithm” de la cual heredan sus métodos las subclases (ver Anexo C).

La clase cuenta con los siguientes métodos, de los cuales algunos se implementan en el algoritmo general y otros están marcados como abstractos y deben ser sobrescritos en los algoritmos específicos:

- “WriteFiles”, método abstracto. Establece el orden en que se ejecutan los métodos del algoritmo específico. Éste método es público y es por el cual se accede para realizar la escritura del motor de inferencia.
- “WriteLibraries”, método concreto. Crea una copia de los archivos de la librería seleccionada por el tipo de sistema difuso, por defecto FuzzyLite o Nefclass-Q.
- “WriteHeaders”, método abstracto. Escribe el *bloque de encabezados*, el cual define cuales de las librerías estándar (C++ Standard Library) se incluyen en el motor de inferencia.
- Para registrar el sistema difuso se definen los métodos abstractos: “WriteInputs”, “WriteOutputs”, “WriteRules” y “WriteMethods”, los cuales escriben los elementos del *bloque de inicialización del programa*.
- “WriteFooter”, método abstracto. Escribe el *pie de página* que cierra el bloque final del archivo de código del motor de inferencia.

Los algoritmos específicos implementan los métodos abstractos utilizando, en su mayor parte, instrucciones de escritura o impresión en el archivo y estructuras de control de flujo del código (condicional y bucles).

Se describen a continuación los algoritmos específicos para la escritura de sistemas difusos tradicionales y clasificadores neurodifusos en el archivo de motor de inferencia.

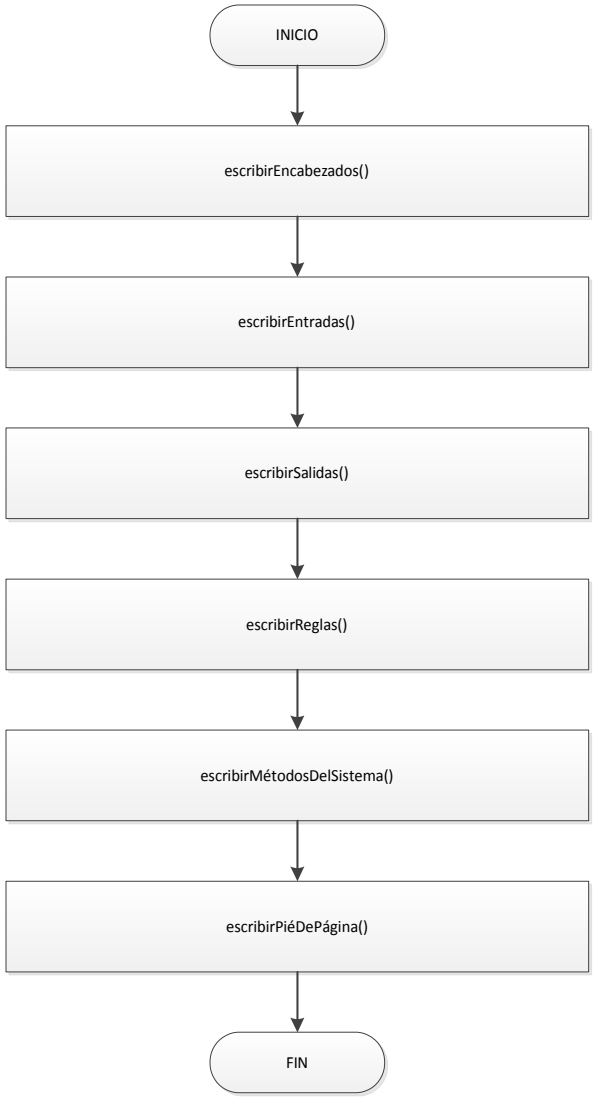
5.3 ALGORITMOS DE SISTEMA DIFUSO DE TIPO MAMDANI, SUGENO Y TSUKAMOTO

Los algoritmos específicos de los sistemas difusos tradicionales crean y organizan los bloques del motor de inferencia difuso en un archivo de motor de inferencia y adjuntan una copia de las librerías FuzzyLite para ser entregados como el producto final de la operación del preprocesador FISPES.

Los algoritmos para los tres tipos de sistema difuso tradicionales comparten la misma estructura. Los algoritmos están definidos en las clases “MamdaniAlgorithm”, “SugenoAlgorithm” y “TsukamotoAlgorithm”.

A continuación se describe la estructura del algoritmo.

Figura 32. Estructura del algoritmo de sistema difuso tipo mamdani



Los métodos de los algoritmos específicos de sistemas difusos tradicionales sobrescriben los métodos abstractos del algoritmo general.

- escribirEncabezados, o “WriteHeaders”, incluye las librerías FuzzyLite en el *bloque de encabezados* del archivo de código del motor de inferencia, abre el bloque de función principal del programa (main) y crea una instancia del sistema difuso.
- escribirEntradas, escribirSalidas y escribirReglas insertan el modelo difuso en el *bloque de inicialización del programa*. En el anexo F se ilustra el

funcionamiento de los métodos de escritura de los elementos del sistema difuso.

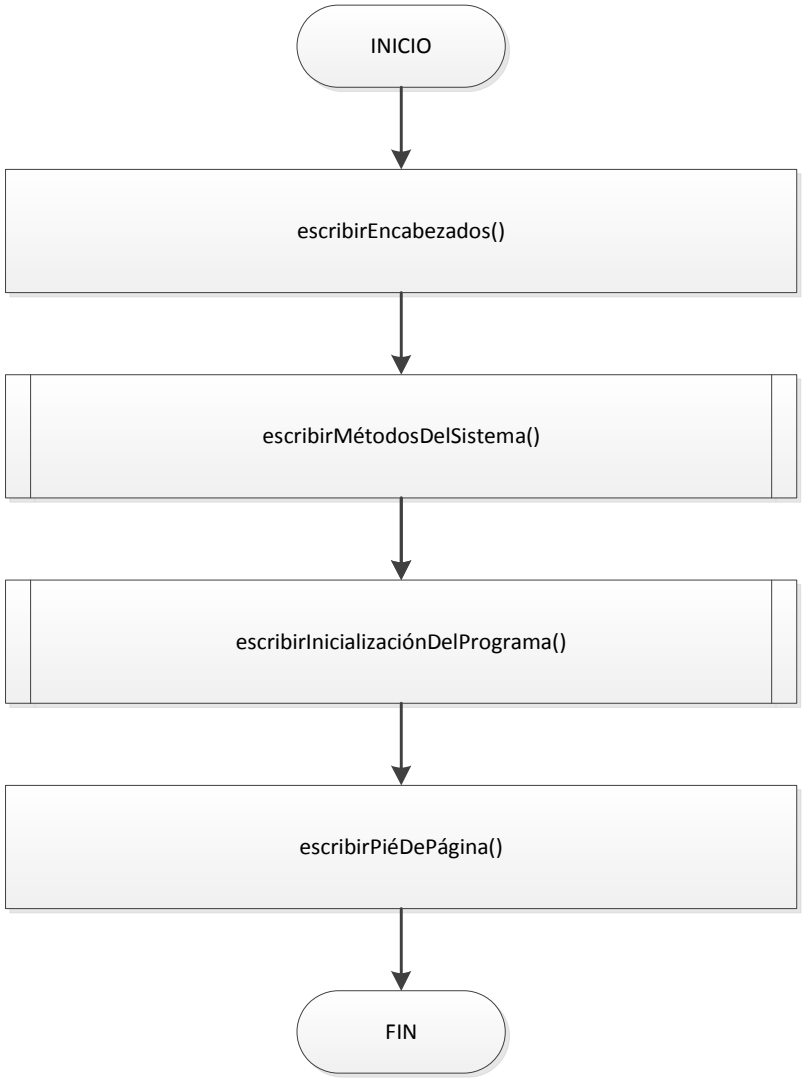
- escribirMétodosDelSistema, o “WriteMethods”, registra las operaciones de implicación, agregación y defusificación del motor de inferencia.
- escribirPieDePágina, o “WriteFooter”, cierra el bloque *main*.

5.4 ALGORITMO DE CLASIFICADOR NEURODIFUSO

El algoritmo específico de clasificador neurodifuso crea y organiza los bloques de motor de inferencia difusa para plasmar el sistema neurodifuso ingresado en el preprocesador, en el archivo de motor de inferencia. El algoritmo adjunta una copia de las librerías Nefclass-Q al archivo de motor de inferencia para ser entregados como la salida de la ejecución del preprocesador.

A continuación se describe la estructura del algoritmo de clasificador neurodifuso.

Figura 33. Estructura del algoritmo de clasificador neurodifuso



Los métodos del algoritmo específico de clasificador neurodifuso implementan los métodos abstractos del algoritmo general. El orden en que el algoritmo ejecuta los métodos difiere del orden de los métodos en el algoritmo de sistemas difusos tradicionales debido a las diferencias en las operaciones que ofrecen ambas librerías.

El método ***escribirEncabezados*** incluye las librerías de Nefclass-Q en el archivo de motor de inferencia difusa, escribe las líneas de prototipado de las funciones presentes en el archivo, inicia el bloque principal del programa (*main*) y ubica el

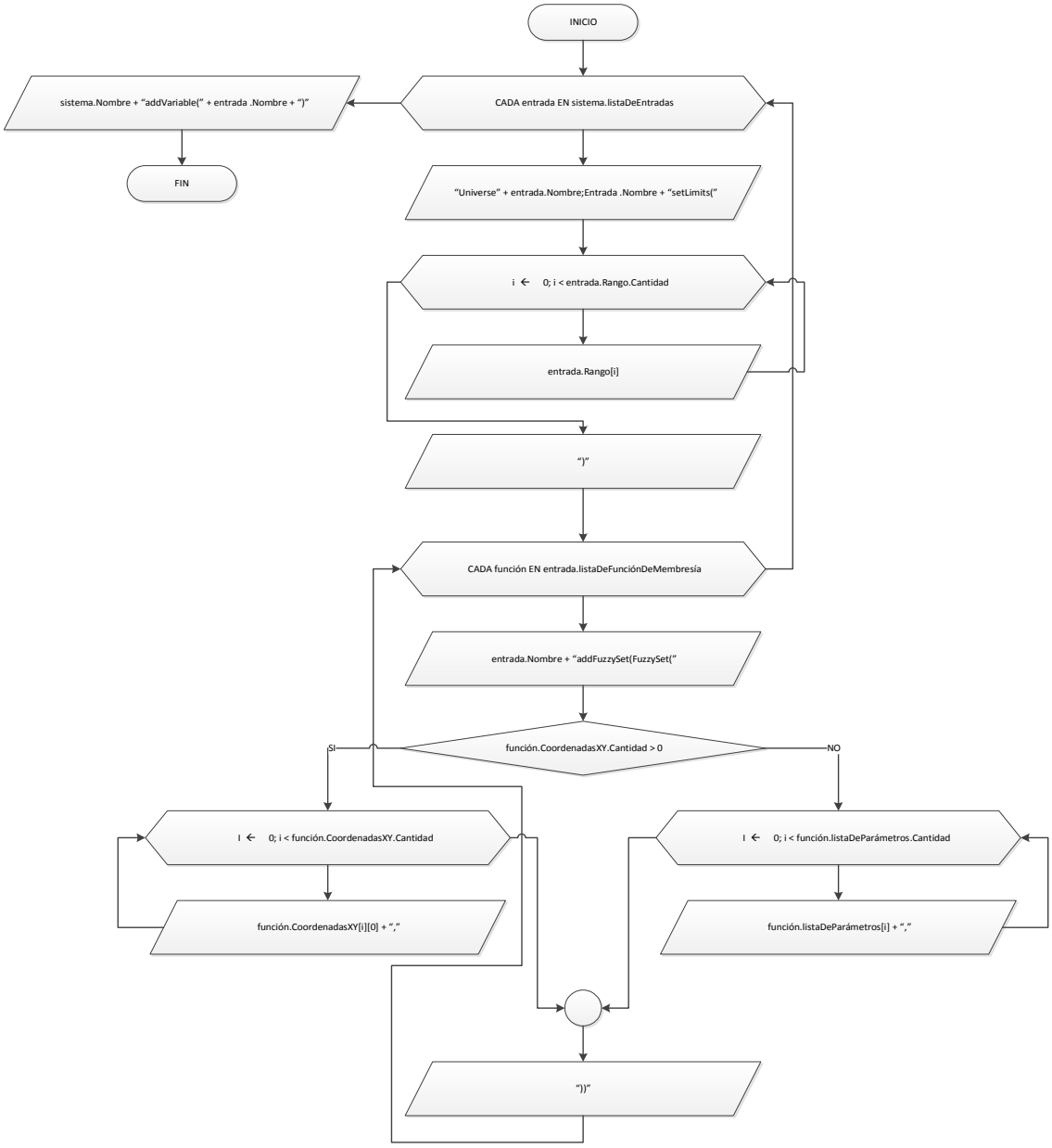
llamado a los métodos que inicializan el sistema difuso y cambian el estado del motor de inferencia a operación continua.

El método ***escribirMétodosDelSistema*** define el método de captura de los datos a partir de los cuales se realiza la inferencia, o clasificación y el método que realiza la clasificación.

El método ***escribirInicializaciónDelPrograma*** escribe el *bloque de inicialización del programa* y está dividido en los métodos escribirEntradas, escribirSalidas y escribirReglas que cumplen la función de integrar el sistema neurodifuso en el motor de inferencia.

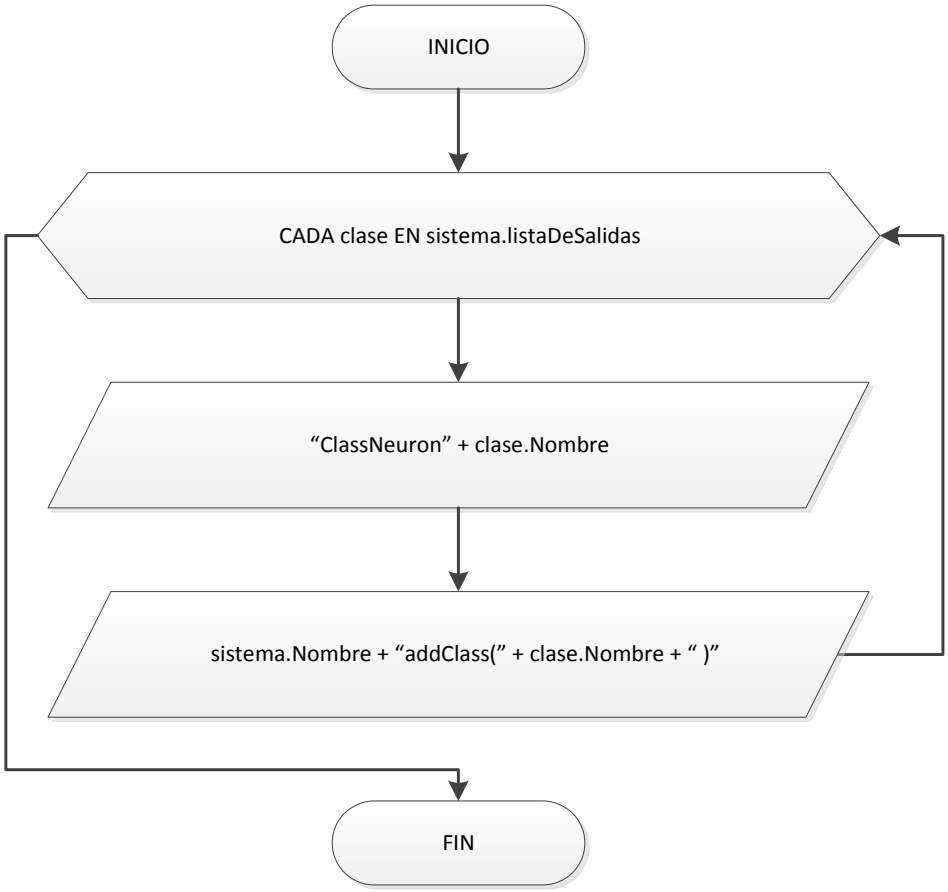
A continuación se describe el procedimiento de cada división del método escribirInicializaciónDelPrograma.

Figura 34. Diagrama de flujo de la función escribirEntradas



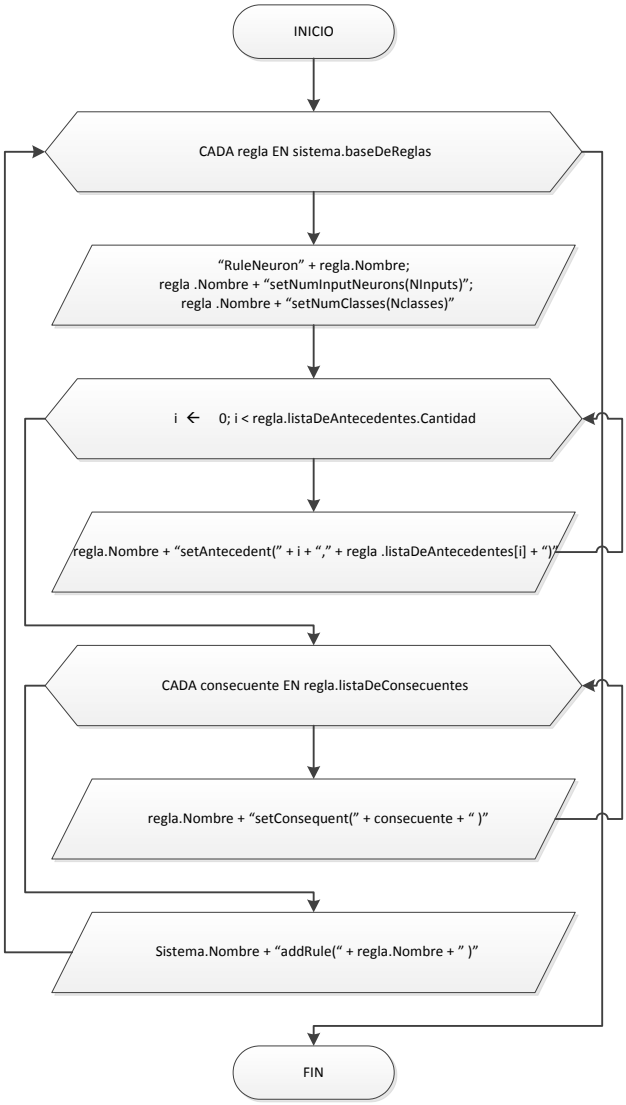
El método ***escribirEntradas*** del algoritmo específico de clasificador neurodifuso, para cada una de las entradas del sistema imprime: la línea de creación del objeto entrada o universo, la asignación de sus límites o rango, la asignación de conjuntos difusos o funciones de membresía y la línea de agregación de la entrada al sistema difuso.

Figura 35. Diagrama de flujo de la función escribirSalidas



El método ***escribirSalidas*** del algoritmo específico imprime por cada una de las salidas del sistema su línea de creación de la neurona de clase y de asignación al sistema.

Figura 36. Diagrama de flujo de la función escribirReglas



El método ***escribirReglas*** del algoritmo específico imprime por cada una de las reglas: la línea de creación de la neurona de regla, de asignación de número de entradas y clases (requerido por la librería Nefclass-Q), la asignación de antecedentes y consecuentes y la línea de asignación de la regla al sistema.

El método ***escribirPieDePágina*** define el bloque de pie de página. El algoritmo no define actualmente ninguna instrucción adicional en este bloque debido a que el motor de inferencia opera de forma continua.

6. PRODUCTO SOFTWARE

El Preprocesador FIS para Plataformas Embebidas FISPES es una aplicación software que extrae la información (sistema difuso) de un modelo difuso contenido en un archivo de texto plano, le realiza un tratamiento la integra en un motor de inferencia, el cual se almacena en un archivo de código y puede ser ejecutado en la plataforma embebida TWR-K70F120M.
El sistema difuso presente en el archivo de modelo se traslada desde la herramienta de manejo de archivos de entrada hasta el modelo de datos, donde el módulo normalizador de sistema difuso ajusta los valores de las propiedades del sistema. El sistema pasa a la herramienta de manejo de archivos de salida, donde se junta con los componentes que definen los métodos de inferencia para conformar el archivo de motor de inferencia.

El diseño del preprocesador se encuentra documentado en los diagramas UML presentes en el anexo C.

Durante el desarrollo de la solución se obtuvieron dos prototipos producto de las iteraciones sobre el ciclo de desarrollo y a partir del segundo prototipo se consolidó el producto de este proyecto.

A continuación se describen los prototipos obtenidos y la solución software FISPES.

6.1 PRIMER PROTOTIPO

El primer prototipo de la solución software permitió realizar el preprocesamiento de archivos de modelo difuso escritos en lenguajes FIS y FPL.
La aplicación tomaba el archivo de modelo de una ubicación predeterminada y guardaba el motor de inferencia junto con sus librerías en otra ubicación.

El prototipo careció de un componente que normalizara las propiedades del sistema difuso antes de su escritura en el archivo de motor de inferencia y presentaba un alto grado de acoplamiento en la definición de las expresiones que conforman la gramática.

El prototipo contó con una interfaz de usuario estilo línea de comandos que no permitía ajustar los parámetros de la operación de preprocesamiento como, por ejemplo, la selección de la ubicación del archivo de modelo difuso y la ruta de guardado del motor de inferencia basada en interfaz de usuario gráfica y la visualización previa del contenido del archivo de modelo difuso.

Se le realizaron pruebas utilizando los modelos de ejemplo “Iris” y “Temperaturas” (ver Anexo F).

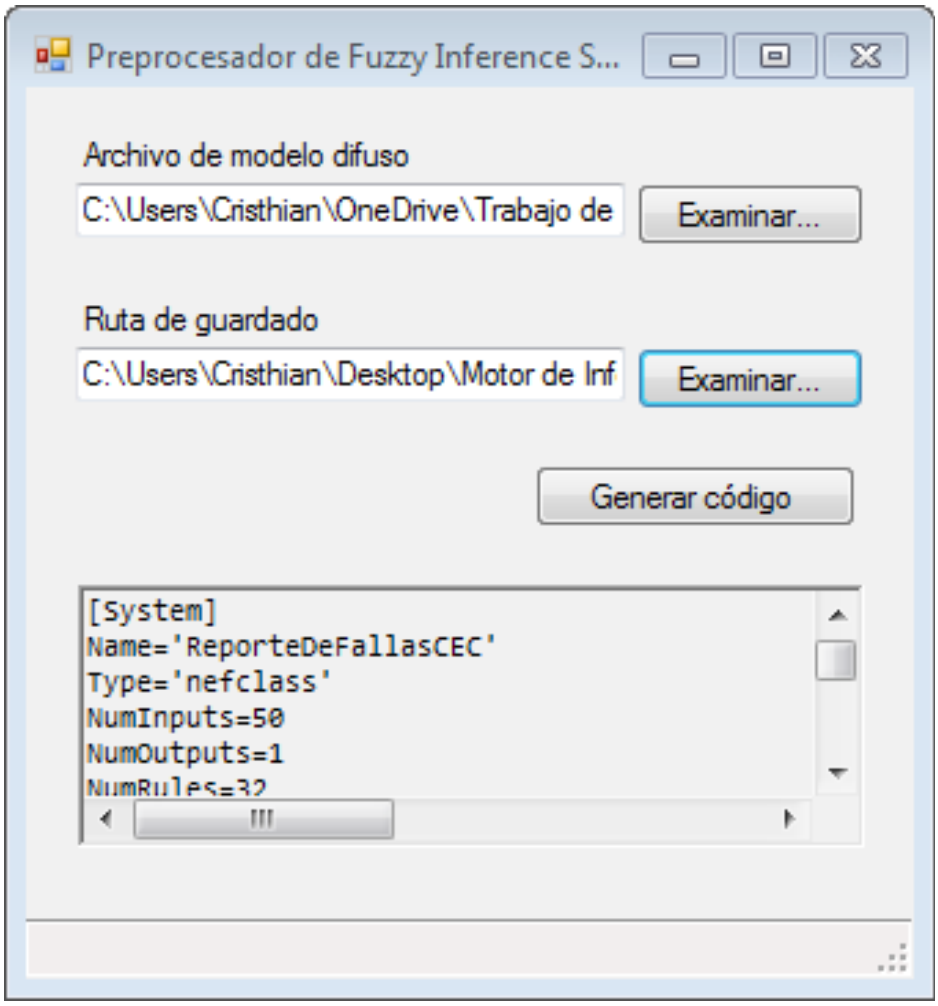
Finalizada la primera etapa del ciclo de desarrollo que arrojó como resultado el primer prototipo, se identificaron nuevos requerimientos (ver Anexo D), dando inicio a la segunda fase del proyecto, etapa en la que se obtuvo el segundo prototipo.

6.2 SEGUNDO PROTOTIPO

Producto de una iteración adicional sobre el ciclo de desarrollo se obtuvo el segundo prototipo de la solución, el cual corrigió las falencias que se presentaron en el primer prototipo y agregó características que satisfacen los requerimientos que surgieron en esta etapa adicional.

El desarrollo de este prototipo tuvo como prioridad la disminución en el acoplamiento entre los módulos y sus componentes. Logró disminuir la cohesión en las operaciones de creación de entidades y sus propiedades en el sistema difuso, disminuyó el acoplamiento en la lectura de los elementos XML del archivo de configuración de lenguaje de sistema difuso y expandió el cubrimiento de los algoritmos de escritura de la herramienta de manejo de archivos de salida para soportar un mayor número de tipos de función de membresía y métodos de sistema.
Contó con una interfaz de usuario gráfica para sistemas operativos Windows (ver Figura 37).

Figura 37. Interfaz de usuario gráfica del prototipo 2



A partir de este segundo prototipo se consolidó el producto final.

6.3 CARACTERÍSTICAS DEL PRODUCTO FINAL

El preprocesador FISPES cuenta con la capacidad para interpretar lenguajes adicionales mediante archivos de configuración XML. Para definir un nuevo lenguaje de modelo difuso es necesario describir las reglas de producción del lenguaje a interpretar y definir una expresión regular que separe los valores y palabras clave de los operadores y caracteres de separación (ver Anexo F).

Las cadenas de texto de la interfaz de usuario gráfica (GUI) se encuentran desacopladas de los formularios, lo que permite la traducción a otros idiomas sin

necesidad de alterar los elementos gráficos de la aplicación. Actualmente la aplicación cuenta con cadenas de texto en inglés (ver Figura 38) y español (ver Figura 39) que se muestran al usuario dependiendo del idioma configurado en su sistema.

Figura 38. Interfaz de usuario gráfica de FISPES, configuración de idioma inglés

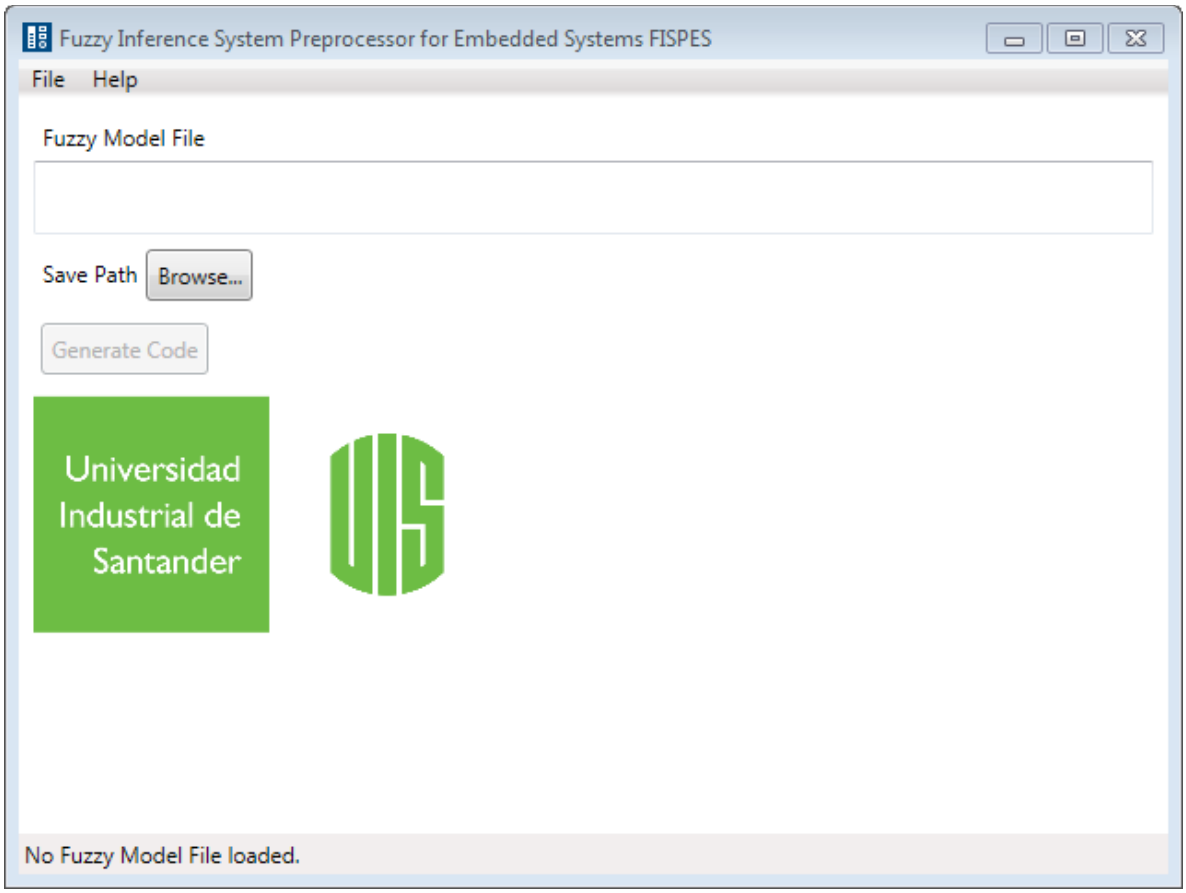
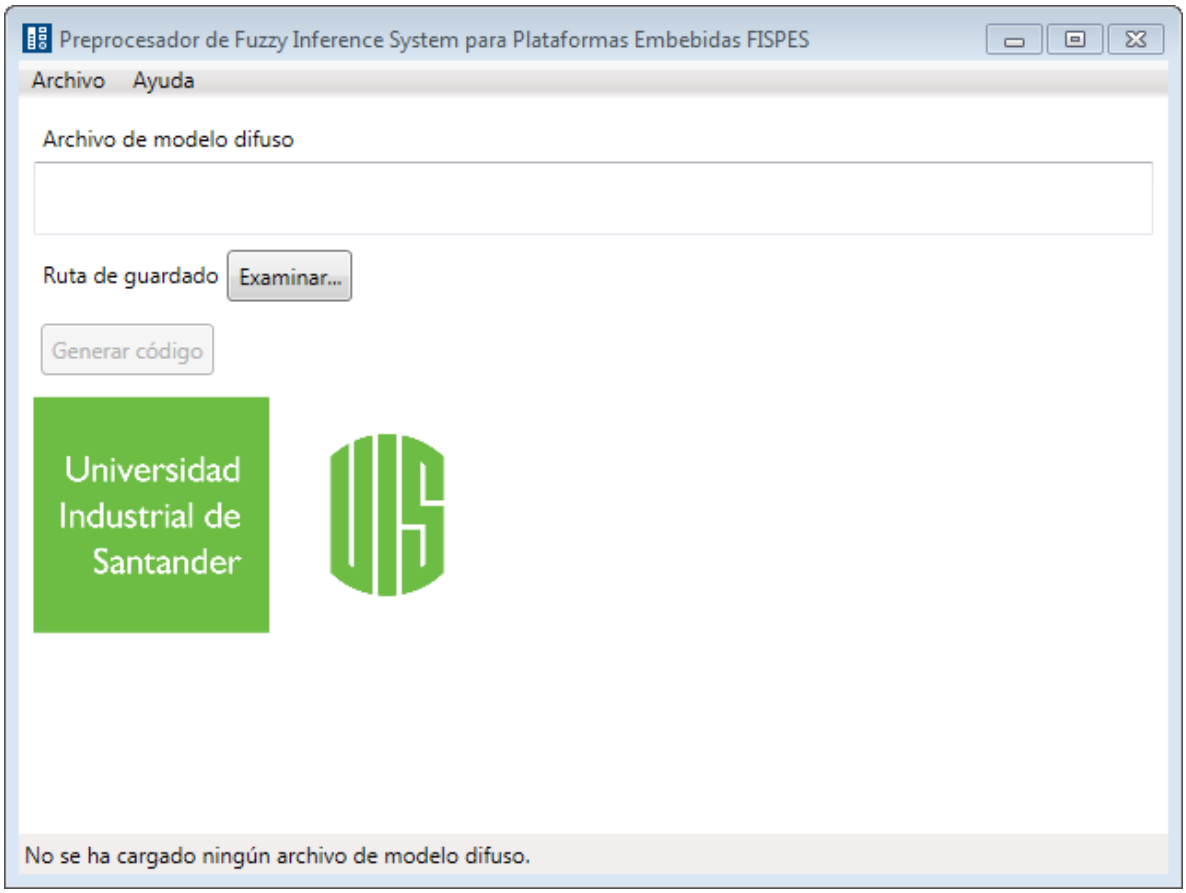


Figura 39. Interfaz de usuario gráfica de FISPES, idioma español



El código fuente del preprocesador fue documentado utilizando las herramientas ofrecidas por el Entorno Integrado de Desarrollo (IDE), Visual Studio 2015 Community Edition, para generar comentarios con etiquetas XML que sucesivamente fueron procesadas con el programa Doxygen para crear el manual de la aplicación en formatos PDF y web HTML.

El producto final presenta un alto grado de desacoplamiento y alta modularidad, lo que permite que se realicen cambios sobre el código fuente sin afectar el funcionamiento de las partes no relacionadas. Adicionalmente, se realizó la adecuación y optimización de las librerías NEFCLASS-Q para ajustarse a los requerimientos de tamaño ocupado en memoria de la plataforma embebida del proyecto, esto significó el recorte de variables y funciones no pertinentes a la ejecución del motor de inferencia, reducción de la precisión de los tipos de datos, mejora del manejo de memoria, simplificación de bucles, reducción de librerías incluidas y disminución de la cantidad de líneas de código.

7. PRUEBAS Y RESULTADOS

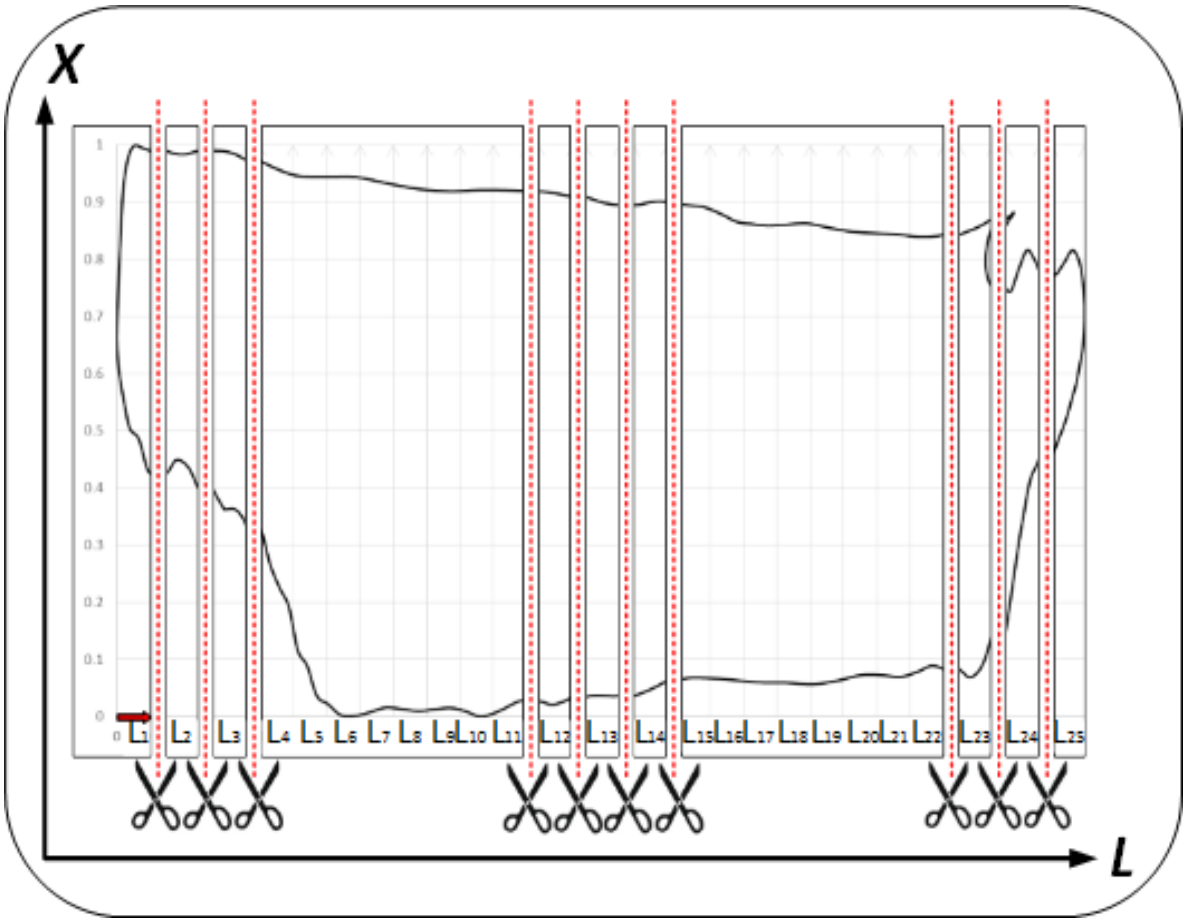
Las pruebas de la solución software se realizaron con el fin de verificar que la ejecución de la aplicación generara motores de inferencia correspondientes al modelo difuso ingresado en el archivo de modelo.

Las *pruebas de integración y de sistema* buscan detectar el funcionamiento no deseado entre los componentes de un sistema y el sistema software como tal. Se realizaron con ayuda de la utilidad *Unit Testing Framework* incluida en el IDE de Visual Studio Community 2015 y el framework *DeepEqual* desarrollado por James Foster que permite hacer comparaciones a profundidad para verificar la igualdad en contenido de dos objetos (Test-specific Equality).

La *validación del motor de inferencia* generado por el preprocesador permite establecer si su funcionamiento está de acuerdo con la clasificación de fallas de la maquinaria de bombeo realizada por expertos. Para realizar la validación, se recurrió a patrones (dinagramas de fondo) de fallas ingresados durante la operación del motor de inferencia.

Los patrones de fallas se ingresaron al motor de inferencia a modo de arreglo de datos, en el cual cada dato representa la amplitud $\boldsymbol{X_i}$ del dinagrama en determinada longitud $\boldsymbol{L_j}$. El dinagrama se seccionó en 25 segmentos los cuales incluyen 2 puntos de amplitud del dinagrama, Un punto para el trayecto de bajada de la bomba, o carrera descendente y otro para el movimiento de subida, o carrera ascendente, (ver Figura 40).

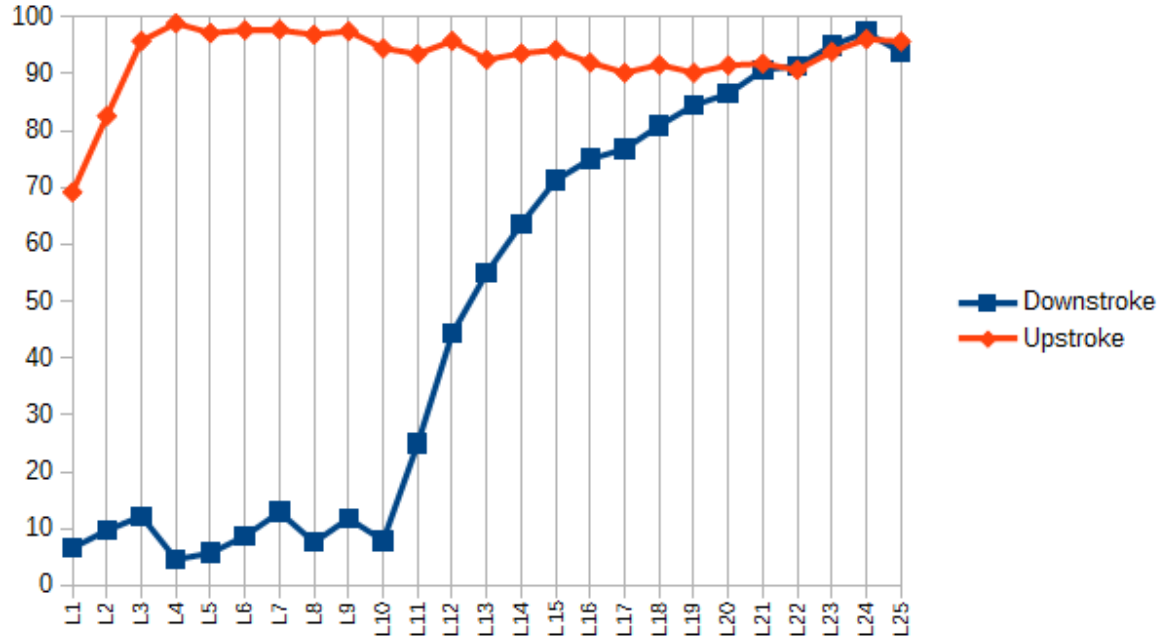
Figura 40. División de los dinagramas en 25 segmentos de longitud



Fuente: MENESES, Edxon. GARAVITO, Fredy, Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico.

Los patrones de fallas se organizaron tomando primero las 25 amplitudes correspondientes a la carrera descendente X_1 a X_{25} y luego las amplitudes de la carrera descendente X_{26} a X_{50} (ver Figura 41).

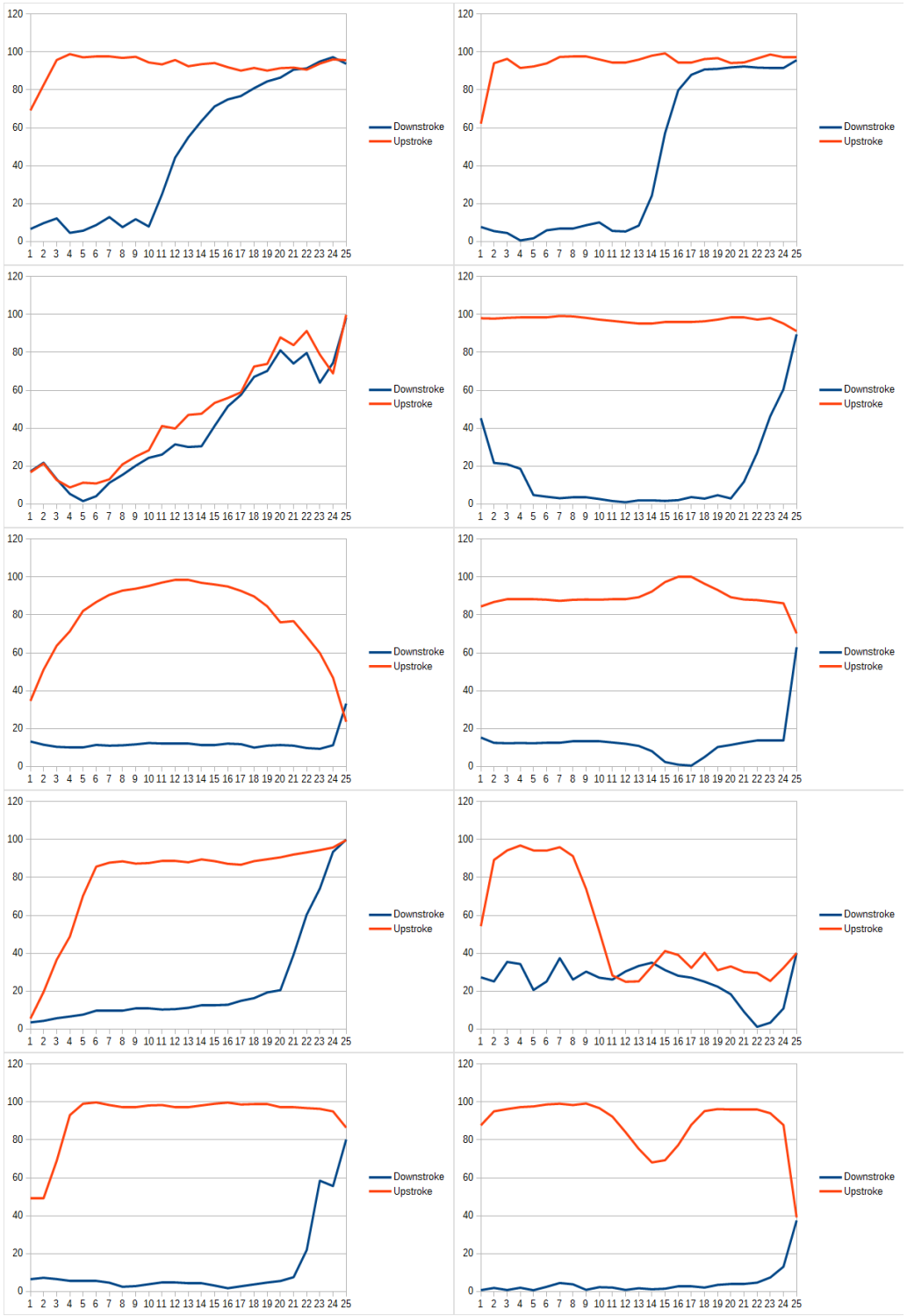
Figura 41. Amplitudes del dinagrama



A continuación se muestran 10 patrones de fallas ingresados al motor de inferencia para su clasificación, cada patrón en la Figura 42 representa una de las 10 fallas que se pueden presentar en la maquinaria de bombeo (de izquierda a derecha, de arriba hacia abajo):

- Interferencia de gas.
- Golpe de fluido.
- Rotura de varillas.
- Fuga en la válvula fija
- Fuga en la válvula viajera o en pistón.
- Barril de la bomba doblado o pegándose.
- Buen llenado, tubería anclada.
- Agujero en el barril de la bomba.
- Ancla de tubería en mal funcionamiento
- Barril de la bomba gastado o partido.

Figura 42. Primeros 10 dinagramas de prueba



Se redactaron los planes de pruebas que describen los objetivos de las pruebas las actividades a seguir para verificar el cumplimiento de estos (ver Anexo E).

7.1 PRUEBAS AL PREPROCESADOR DE FIS PARA PLATAFORMAS EMBEBIDAS

7.1.1 Pruebas de integración. Las *pruebas de integración* determinaron la eficacia de cada uno de los módulos del preprocesador FISPES para transmitir, de manera consistente, el sistema difuso a la siguiente etapa de la operación de preprocesamiento. El plan de pruebas de integración se describe en el anexo E.

Se definieron los siguientes métodos de pruebas de integración:

- IntegrationTestMethodIrisFuzzyModelFileToIFMT: crea un objeto *InputFilesManagementTool* (herramienta de manejo de archivos de entrada) con la ruta del archivo de modelo neurodifuso *Iris* y un objeto de sistema difuso *FLSystem* equivalente al sistema difuso Iris (fase *Arrange*), llama el método ReadFuzzyModelFile de la herramienta de manejo de archivos (fase *Act*) y verifica la igualdad profunda (Test-Specific Equality) entre el sistema difuso creado en la fase Arrange (esperado) y el sistema difuso creado por la herramienta de manejo de archivos de entrada (actual).
- IntegrationTestMethodIrisFuzzySystemToOFMT: crea un objeto *DataModel* (modelo de datos) con el sistema neurodifuso *Iris* normalizado y un objeto *OutputFilesManagementTool* (herramienta de manejo de archivos de salida) con el sistema Iris (fase *Arrange*), llama el método WriteFiles de la herramienta de manejo de archivos y almacena el contenido del motor de inferencia generado en una cadena de texto (fase *Act*). Finalmente compara la cadena de texto generada con otra cadena previamente preparada con el motor de inferencia esperado.
- IntegrationTestMethodTemperaturesFuzzyModelFileToIFMT: crea un objeto *InputFilesManagementTool* con la ruta del archivo de modelo difuso tipo Mamdani *Temperatures* y un objeto de sistema difuso *FLSystem* equivalente al sistema del modelo de temperaturas (fase *Arrange*), llama el método ReadFuzzyModelFile de la herramienta de manejo de archivos (fase *Act*) y verifica la igualdad profunda entre el sistema difuso creado en la fase Arrange y el sistema difuso creado por la herramienta de manejo de archivos de entrada.
- IntegrationTestMethodTemperaturesFuzzySystemToOFMT: crea un objeto *DataModel* (modelo de datos) con el sistema difuso tipo Mamdani *Temperatures* normalizado y un objeto *OutputFilesManagementTool* (herramienta de manejo de archivos de salida) con el sistema temperaturas

(fase *Arrange*), llama el método WriteFiles de la herramienta de manejo de archivos y almacena el contenido del motor de inferencia generado en una cadena de texto (fase *Act*) y compara la cadena de texto generada con otra cadena previamente poblada con el motor de inferencia preparado para el sistema temperaturas.

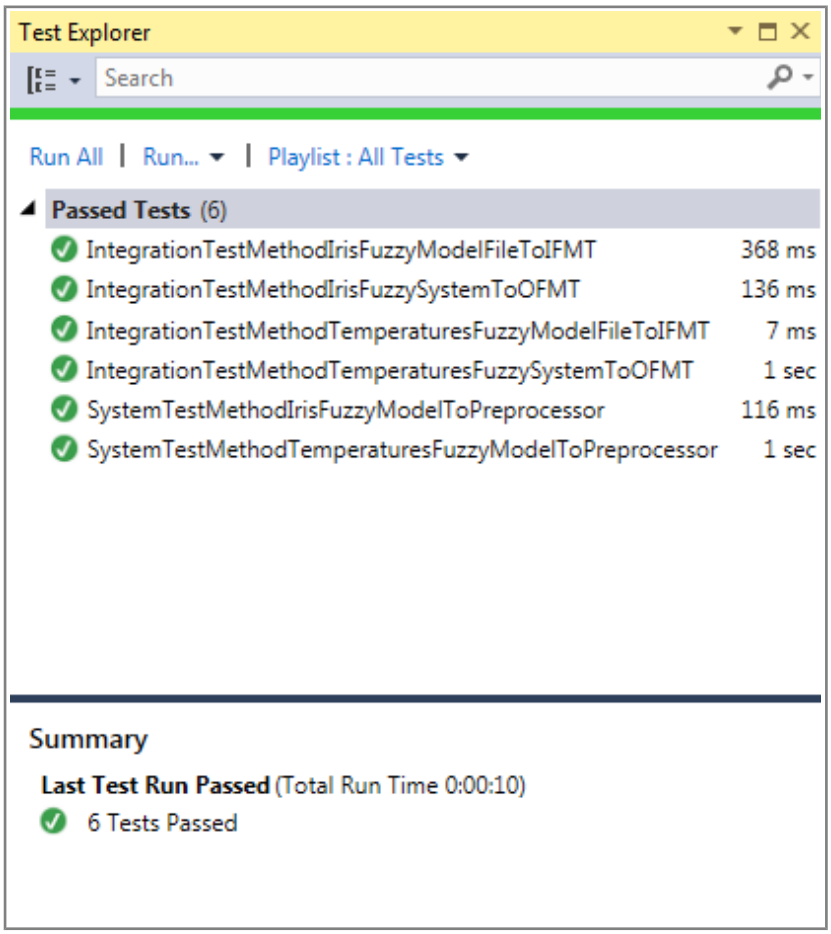
7.1.2 Pruebas de sistema. Las *pruebas de sistema* establecieron la eficacia del preprocesador, como un sólo elemento, para generar un motor de inferencia correspondiente al modelo difuso ingresado en el archivo de modelo. El plan de pruebas de sistema se describe en el anexo E.

Se definieron los siguientes métodos de pruebas de integración:

- SystemTestMethodIrisFuzzyModelToPreprocessor: se crea un objeto de tipo *Preprocessor* (preprocesador) con la ruta del archivo de modelo neurodifuso *Iris* y la ruta de guardado de los archivos como parámetros (fase *Arrange*), llama el método *Run* del preprocesador y almacena el motor de inferencia generado en una cadena de texto (fase *Act*). Por último verifica la igualdad entre el motor de inferencia almacenado en la cadena de texto y una cadena preparada con el motor de inferencia para el sistema *Iris*.
- SystemTestMethodTemperaturesFuzzyModelToPreprocessor: se crea un objeto de tipo *Preprocessor* (preprocesador) con la ruta del archivo de modelo difuso tipo Mamdani *Temperatures* y la ruta de guardado de los archivos como parámetros (fase *Arrange*), llama el método *Run* del preprocesador y almacena el motor de inferencia generado en una cadena de texto (fase *Act*). Finalmente verifica la igualdad entre el motor de inferencia almacenado en la cadena de texto con la cadena preparada con el motor de inferencia del sistema temperaturas.

El preprocesador pasó todas las pruebas de integración y de sistema (ver Figura 43).

Figura 43. Resultados de las pruebas de integración y de sistema



7.2 VALIDACIÓN DEL MOTOR DE INFERENCIA PARA CEC

El motor de inferencia difusa para CEC fue generado a partir del preprocesamiento del archivo de modelo difuso planteado en el proyecto “Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico”. Para verificar que el motor de inferencia difusa correspondiera al modelo difuso ingresado al preprocesador, se comparó el resultado de la inferencia con la clasificación de una serie de patrones de fallas realizada por expertos de Campo Escuela Colorado.

La prueba consistió en cargar el motor de inferencia en la plataforma de desarrollo embebida TWR-K70F120M, ingresar como entradas del motor de inferencia los patrones de fallas y comparar las fallas clasificadas por el motor de inferencia con las fallas reportadas por los expertos.

Se ingresaron 60 patrones de fallas en dos grupos de 30 dinagramas, organizando cada grupo en una ventana flotante de la cual el motor de inferencia retiró y clasificó uno a uno los dinagramas. La totalidad (100%) de los dinagramas fueron clasificados correctamente. El elevado porcentaje de acierto se debe al entrenamiento de la red neurodifusa que definió el modelo para CEC con suficientes patrones de ejemplo y épocas de entrenamiento.

8. CONCLUSIONES Y RECOMENDACIONES

1. Se desarrolló una solución software denominada preprocesador de Fuzzy Inference System (FIS), que generó el motor de inferencia basado en lógica difusa y que opera sobre la plataforma de desarrollo embebida Freescale TWR-K70F120M.

2. Se diseñó e implementó un módulo del preprocesador, el modelo de datos, para el manejo de información de sistemas difusos Fuzzy Inference System (FIS) con base en las entidades que establece la teoría de lógica difusa y los módulos, las herramientas de manejo de archivos de entrada y salida, que dan soporte a la tarea de preprocesamiento.

3. Se implementaron los segmentos de código, los algoritmos específicos de la *herramienta de manejo de archivos de salida*, que conforman el motor de inferencia difuso resultante de la operación del preprocesador, de acuerdo con las características ofrecidas por FIS.

4. Se documentó el preprocesador bajo el estándar UML utilizando los diagramas de clases, estados de máquina y de secuencia.

5. Se realizaron las pruebas de integración y de sistema, las cuales verificaron que el preprocesador se comporta de la manera deseada, generando motores de inferencia correspondientes al modelo difuso ingresado en el archivo de modelo.

6. Se validó el motor de inferencia generado por el preprocesador a partir del modelo difuso de interés para Campo Escuela Colorado, usando como datos de entrada, patrones de fallas clasificados por expertos de CEC y recibiendo un reporte de fallas con un nivel de precisión bastante acertado.

7. Se cumplieron a cabalidad los objetivos propuestos en el proyecto de grado creando la solución software que logra llevar el archivo de modelo difuso para CEC al sistema embebido presente en el proyecto de investigación 8556, aportando en su objetivo de materializar un sistema de detección y reporte de fallas automático e INTELIGENTE para Campo Escuela Colorado.

8. Se demostró la versatilidad de la solución software en el tratamiento de modelos difusos de diferentes campos del conocimiento, como lo fueron la clasificación de especies botánicas y el diagnóstico de fallas en maquinaria de bombeo.

9. El desarrollo de la solución software FISPES representa un paso adelante en el cumplimiento de la misión de la Universidad Industrial de Santander.

10. La ejecución de este proyecto representó para el autor/estudiante un gran aprendizaje en las áreas de inteligencia artificial, ingeniería del software y desarrollo sobre dispositivos embebidos y la introducción a la temática de informática industrial y desarrollo enfocado a *Internet of the Things* (IoT).

11. Se invita a la comunidad estudiantil de la escuela de Ingeniería de Sistemas de la Universidad Industrial de Santander a abordar temas de investigación sobre desarrollo en dispositivos embebidos, los cuales conforman un potencial campo de acción a desarrollarse en años futuros.

12. Se recomienda a los desarrolladores interesados en expandir la aplicación para operar modelos difusos que requieran el manejo de números racionales o enteros, de mayor o menor precisión, en plataformas con espacio en memoria diferente al de la *board* utilizada en este proyecto, crear una opción que permita seleccionar los tipos de datos adecuados para el rendimiento o precisión del motor de inferencia.

BIBLIOGRAFÍA

CABALLERO, Fabio; SALAMANCA, Julian. Herramienta De Aplicación De Software Para Clasificación De Patrones De Datos Implementando Una Arquitectura Neuro-Fuzzy. Bucaramanga: Universidad Industrial de Santander, 2011.

FCSL. Software Development Productivity. Iterative Development - delivering projects on time. En: fcsl - Software Development Productivity. [En línea]. Recuperado en julio de 2014.Disponible en: http://www.fcsltech.com/iterativedevelopment.html

Fisher, Ronald A. The Use of Multiple Measurements in Taxonomic Problems. Blackwell Publishing Ltd. [En línea]. Lóndres: University College London. 1936. [Consultado en julio de 2016]. Disponible en: http://onlinelibrary.wiley.com/doi/10.1111/j.1469-1809.1936.tb02137.x/pdf

MAHESHWARI, Shikha; JAIN, Dinesh Ch. A Comparative Analysis of Different types of Models in Software Development Life Cycle. International Journal of Advanced Research in Computer Science and Software Engineering. [En línea]. 2012. [Consultado en octubre de 2014]. ISSN: 2277 128X. Disponible en: http://www.ijarcsse.com/docs/papers/May2012/Volum2_issue5/V2I500405.pdf

MENESES, Edxon; GARAVITO, Fredy. Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico. Bucaramanga: Universidad Industrial de Santander, 2014.

MENESES, Jorge; ACEVEDO, Alfredo; RAMÓN, Jorge. Desarrollo de un prototipo de pozo inteligente para CEC. Bucaramanga: Universidad Industrial de Santander, 2014.

Object Oriented Design. Strategy Pattern. En: oodesign.com. [En línea]. Recuperado en noviembre de 2015. Disponible en: http://www.oodesign.com/strategy-pattern.html

Object Oriented Design. Interpreter Pattern. En: oodesign.com. [En línea]. Recuperado en noviembre de 2015. Disponible en: http://www.oodesign.com/interpreter-pattern.html

PARI, M. Nandri; KABIR, A.H.; MOTAHHARI, S. Mahdia; BEHROUZ, Turaj. Smart well Benefits, Types of Sensors, Challenges, Economic Consideration, and Application in Fractured Reservoir. Society of Petroleum Engineers. [En línea]. 2009. [Consultado en julio de 2016]. ISBN: 978-1-61399-021-6. Disponible en: DOI: http://dx.doi.org/10.2118/126093-MS

ANEXOS

ANEXO A. ESTADO DEL ARTE

Cerca del 70% de la producción mundial de petroleo y gas proviene de campos maduros. Un campo puede ser considerado maduro si ha producido más del 50% de sus recursos probados y probables, ha producido por más de 25 años o la producción ha alcanzado su pico y ha empezado a disminuir.
La cantidad de petróleo en campos maduros es aproximadamente igual a la cantidad de reservas identificadas de crudo de petroleo convencional acreditado a medio oriente.

El petroleo extraído de los pozos maduros es altamente viscoso y no fluye fácilmente comparado al de los pozos de producción bajo condiciones de reserva normales.
La industria se ha concentrado en realizar investigaciones para desarrollar métodos que faciliten la recuperación del petróleo en este tipo de pozos, la extracción de crudos pesados presenta varios desafíos a la maquinaria, debido a las fuerzas experimentadas durante la extracción del crudo pesado se pueden presentar fallas en los componentes del sistema de bombeo mecánico.

Una de las aplicaciones de los pozos inteligentes (*Smart Wells*) ha sido en campos maduros, donde el uso de válvulas y sensores, monitoreados de forma remota mejora la eficiencia de los equipos de bombeo, permitiendo detectar fallas en tiempo real y corregirlas, entre otros beneficios.

Para mejorar la extracción en campos petrolíferos se han creado sistemas automatizados para obtener datos, analizar y tomar decisiones sobre los pozos, estos sistemas se denominan campos inteligentes (*Smart Field*), que gracias al uso de tecnologías permiten monitorear y detectar fallas remotamente sin necesidad de intervención humana.

SMART FIELD

Diferentes compañías petroleras a nivel mundial han incorporado las tecnologías de Smart Well y Smart Field a su portafolio de servicios: Schlumberger Limited, Halliburton Company, Baker Hughes Incorporated, Weatherford International, entre otras.

Las compañías petroleras BP plc (anteriormente conocida como British Petroleum), Petróleo Brasilero S.A. - Petrobras, Statoil ASA, entre otras, ya han integrado las tecnologías Smart Field a sus campos.

En la Universidad Industrial de Santander se ha puesto en marcha el proyecto denominado “Desarrollo de un prototipo de pozo inteligente para CEC” el cual busca dotar a la industria colombiana de las tecnologías de pozo y campo inteligente propias, evitando la dependencia económica, de funcionamiento y mantenimiento que generan las herramientas actualmente presentes en CEC que han sido adquiridas de empresas extranjeras y son de carácter privativo.

SISTEMAS NEURODIFUSOS

La inteligencia artificial se ha aplicado con éxito en problemas que no pueden ser abordados desde un modelo matemático o que son muy complejos para ser computarizados, las técnicas de *soft computing* aprovechan el área del razonamiento cognitivo para reducir la complejidad de un modelo, es así como en 1965 el ingeniero y matemático Lotfi Zadeh desarrolló una lógica que se ajustara más a la realidad y que permitiera definir conjuntos difusos que pueden aprovechar el conocimiento de expertos para la implementación de sistemas de control más fáciles de mantener.

Paralelamente, la tecnología de redes neuronales se desarrollaba para aprender automáticamente patrones a partir de datos (aprendizaje supervisado). Actualmente, estas redes se usan en infinidad de aplicaciones de reconocimiento de imágenes, procesamiento de texto y clasificación en general. Sin embargo, tanto la lógica difusa como las redes neuronales presentan ventajas y desventajas para sistemas de control. Gracias a la experimentación con sistemas híbridos artificiales para mejorar el rendimiento de estos modelos se obtuvieron los sistemas neurodifusos que mediante redes neuronales permiten aprender reglas y conjuntos difusos lo cual representa una ventaja en términos de tiempo e interpretabilidad. Desde entonces, se han desarrollado paquetes de software neurodifuso como NEFCLASS y el *toolbox* de Fuzzy-logic de Matlab que son ampliamente usados para modelar sistemas de control industrial.

El campo de la automatización de procesos y sistemas de control industrial ha sido uno de los más beneficiados con el desarrollo de sistemas neurodifusos. En especial en áreas como la medicina o servicios financieros, donde por razones de seguridad se necesitan modelos transparentes, confiables que puedan ser entendidos por el usuario.

En la Universidad Industrial de Santander el estudio de sistemas neurodifusos por parte de estudiantes de ingeniería mecánica, ha dado como resultado el desarrollo

de una aplicación con arquitectura que permite el entrenamiento y aprendizaje para un problema dado por el usuario, entre sus características se encuentran la posibilidad de definir parámetros iniciales del sistema como las tasas de aprendizaje, base de conocimiento inicial, número de conjuntos difusos por variable, número de épocas de entrenamiento y máximo número de reglas.

La aplicación de técnicas de inteligencia artificial en pozos petroleros es algo común particularmente para detectar problemas mecánicos en sistemas de bombeo usando diagramas de dinamómetros y datos del tipo de petroleo extraído. El trabajo de grado “Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico” realizado por estudiantes de la escuela de ingeniería mecánica, sirve al proyecto de pozo y campo inteligente como la herramienta para realizar la clasificación de dinagramas de fondo según las fallas que se presenten en sistema de bombeo mecánico. Producto de este trabajo es el conjunto de reglas que permite a un sistema difuso, como resultado de su ejecución, clasificar los dinagramas que ingresan al sistema en tipos de fallas en la maquinaria.

La propuesta de desarrollar un preprocesador de Fuzzy Inference System (FIS) que genere programas de tipo motor de inferencia difusa, busca responder a la problemática de cómo determinar si se están presentando fallas en pozos de extracción petrolera, utilizando un dispositivo embebido para tratar los datos obtenidos de la actividad de bombeo consignados en dinagramas de fondo, ayudándose en conocimiento experto adquirido de una fuente externa (sistema neurodifuso). La ejecución de este trabajo de grado busca el avance en una de las etapas del desarrollo del proyecto de pozo y campo inteligente y pretende abrir el camino al desarrollo de trabajos consecuentes que se encarguen de dar autonomía al sistema de bombeo para notificar situaciones de funcionamiento erróneo y corregir en la medida que sea posible sus propias fallas.

ANEXO B. MARCO TEÓRICO

DISPOSITIVO EMBEBIDO

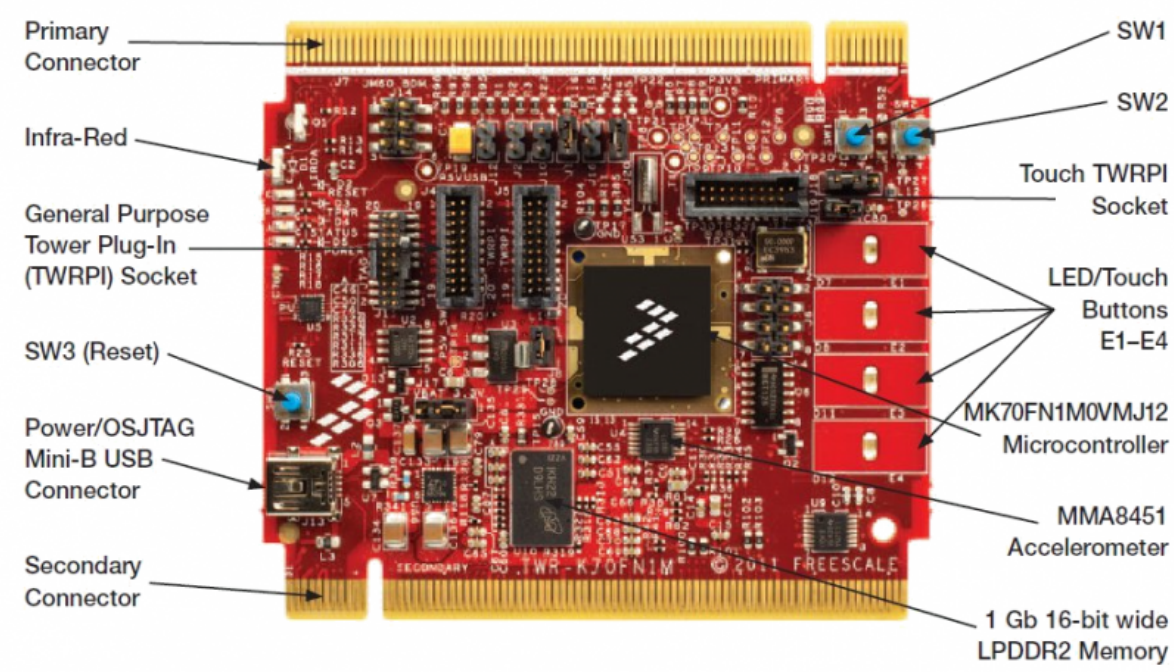
Un dispositivo embebido es un sistema de computación diseñado para realizar funciones específicas en tiempo real. Los sistemas embebidos se pueden programar directamente en el lenguaje ensamblador del microcontrolador o microprocesador incorporado sobre el mismo, o utilizando los compiladores específicos, pueden utilizarse lenguajes como C o C++; en algunos casos, cuando el tiempo de respuesta de la aplicación no es un factor crítico, también pueden usarse lenguajes como Java.

Plataforma de desarrollo embebida TWR-K70F120M. El dispositivo TWR-K70F120M es una placa de desarrollo parte de la plataforma Freescale Tower System, que permite el prototipado rápido y la reutilización de herramientas gracias al hardware configurable.

Características del dispositivo:

- Microcontrolador (MCU) MK70FN1M0VMJ12 (ARM® Cortex®-M4 @ 120 MHz, 1 MB flash, capacidad para LCD gráfico, Ethernet, USB On The Go, soporte para encriptación, controlador flash y DDR NAND, 256 MBGA).
- Memoria RAM Micron MT47H64M16HR-25 1 Gb 16-bit wide LPDDR2.
- Memoria NAND flash Micron MT29F2G16ABAEAWP 2 Gb.
- Circuito MC9S08JM60 open source JTAG (OSJTAG).
- Interfaz con el módulo periférico TWR-LCD-RGB (acepta datos RGB directamente del controlador K70 MCU LCD).
- Cuatro LEDs controlados por el usuario.
- Cuatro touch pads capacitivos y dos pulsadores mecánicos.
- Socket TWRPI de propósito general.
- Socket TWRPI-TOUCH-STR (sensible al tacto).

Figura 1. Dispositivo embebido TWR-K70F120M

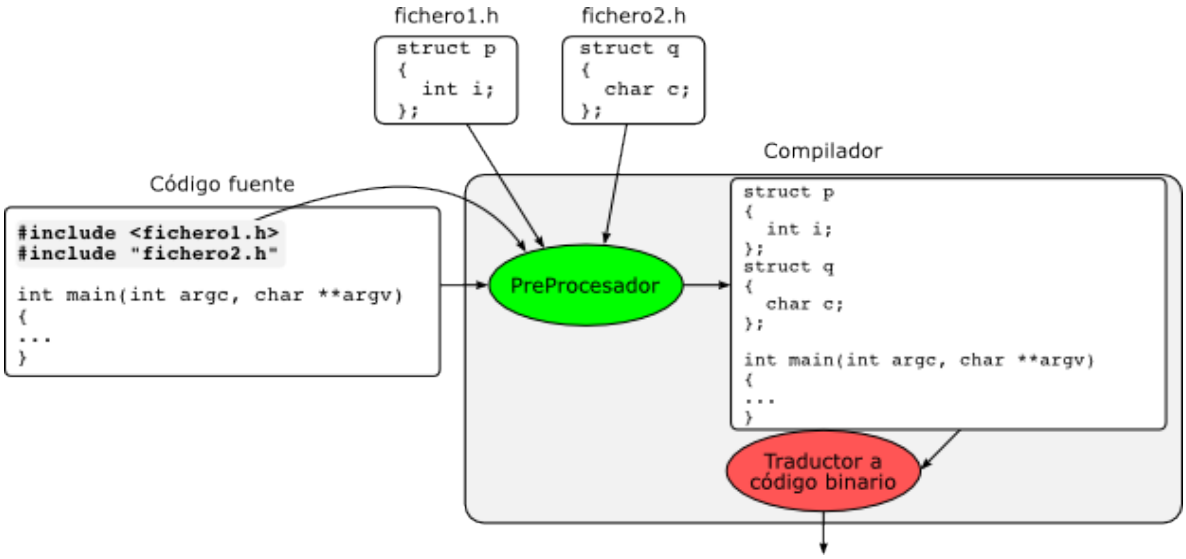


Fuente: Freescale Semiconductor Inc.

PREPROCESADOR

Es un programa que forma parte del compilador, manipula el texto de un archivo fuente antes de que comience la traducción real. El preprocesador divide el archivo fuente en *tokens* con el fin de buscar llamadas a macros. Aunque el compilador invoca normalmente el preprocesador en el primer paso, también se puede invocar el preprocesador por separado para procesar el texto sin compilación.

Figura 2. Esquema de un preprocesador de lenguaje C



Fuente: Universidad Carlos III de Madrid, Departamento de Ingeniería Telemática.

Algunas funciones de un preprocesador son:

- Procesar macros: Un macro es una abreviatura de estructuras más grandes.
- Incluir archivos: El ejemplo más común es cuando una llamada a una librería #include en el lenguaje C es reemplazada y se inserta el archivo en su lugar.
- Preprocesadores racionales: Enriquecen el lenguaje con recursos como estructuras de control y datos.
- Extensiones a lenguajes.
- Eliminación partes del código fuente.

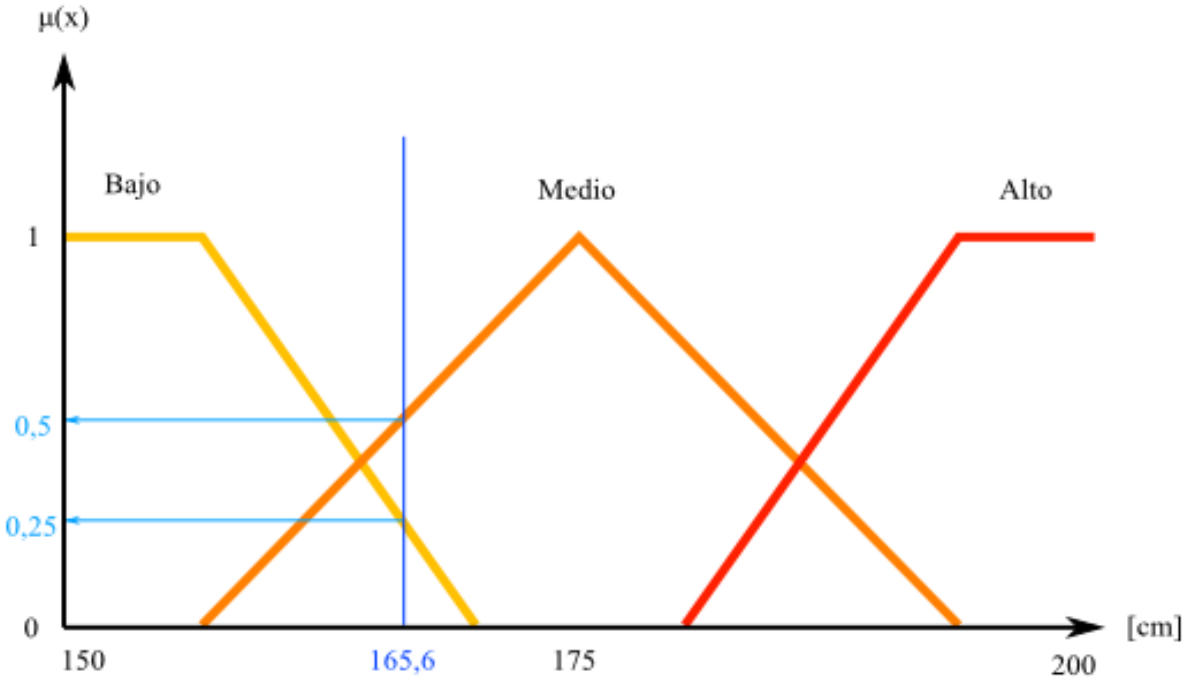
LÓGICA DIFUSA Y MOTOR DE INFERENCIA

La lógica difusa es una rama de la inteligencia artificial y una forma de la lógica plurivalente. Trata con el razonamiento aproximado que difiere del razonamiento exacto y está basada en la teoría de conjuntos difusos, propuesta por Lotfi A. Zadeh en 1965, que maneja clases de objetos con límites difusos en las cuales la membresía es medible en grados.

La lógica difusa se puede aplicar en procesos complejos donde no existe un modelo matemático preciso. Cuando se necesite usar el conocimiento de un experto que utiliza conceptos ambiguos o imprecisos, hay partes del sistema que son desconocidas o no se pueden medir de forma confiable y cuando el ajuste de una variable puede producir el desajuste de otras.

La teoría de conjuntos difusos es una generalización de la teoría clásica de conjuntos, ésta hace posible la representación de conceptos que no tienen límites precisos como “alto” o “pesado”, esto se logra al permitir que los elementos pertenezcan a una variable en cierto grado, donde el grado de membresía de una variable es un número real entre 0 y 1 (ver Figura 3 de los anexos).

Figura 3. Representación de la variable lingüística “Altura” en conjuntos difusos



Formalmente un conjunto difuso F sobre un universo de discurso o rango X, está definido por su función de pertenencia μF el cual relaciona elementos x a un número entre cero y uno.

$$\mu F(x) : X \rightarrow [0, 1]$$

La arquitectura de un sistema experto puede resumirse en tres componentes principales: base de conocimientos, motor de inferencia y memoria de trabajo.

La base de conocimiento contiene hechos conocidos y reglas utilizables por el sistema experto.

La memoria de trabajo almacena las explicaciones de las conclusiones intermedias, con la cadena de reglas recorrida para llegar a una conclusión, que el sistema experto utiliza durante el proceso de resolución.

El motor de inferencia representa la inteligencia del sistema experto. Genera los resultados a partir de la base de conocimientos y la memoria de trabajo.

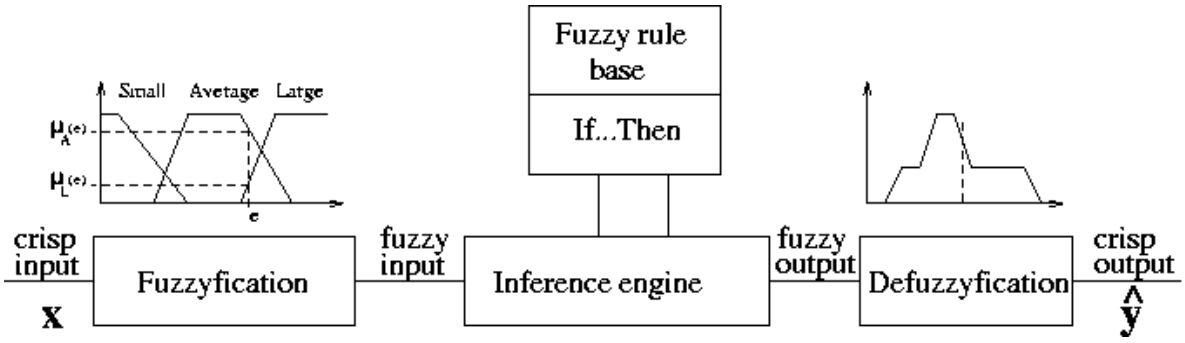
Figura 4. Componentes de un sistema experto

Fuente: Amitrano, Sergio. Di Chiazza, Rodolfo. Lorenzo, Jorge. Sistema experto basado en lógica difusa para la Central Nuclear Embalse. Universidad de Buenos Aires, Facultad de Ciencias Exactas y Naturales.

CONTROLADORES DIFUSOS

Uno de los campos dónde es más frecuente ver la aplicación de lógica difusa es en controladores difusos, estos permiten describir un conjunto de reglas que utilizaría un experto para controlar un proceso y a partir de estas reglas generar acciones de control. El control difuso se aplica en gran variedad de problemas.

Figura 5. Estructura de un modelo difuso



Fuente: L'Institut national de la recherche agronomique, département de Mathématiques et Informatique Appliquées.

En la fuzzificación los valores reales se convierten en valores difusos y se asignan grados de pertenencia a cada una de las variables de entrada con relación a los conjuntos difusos previamente definidos utilizando las funciones de pertenencia asociadas a los conjuntos difusos. En la base de conocimiento se definen las reglas lingüísticas de control que realizará la toma de decisiones y decidirán la forma en la que debe actuar el sistema.

La inferencia relaciona los conjuntos difusos de entrada y salida para representar las reglas que definirán el sistema. En la inferencia se utiliza la información de la base de conocimiento para generar reglas o condiciones de forma:

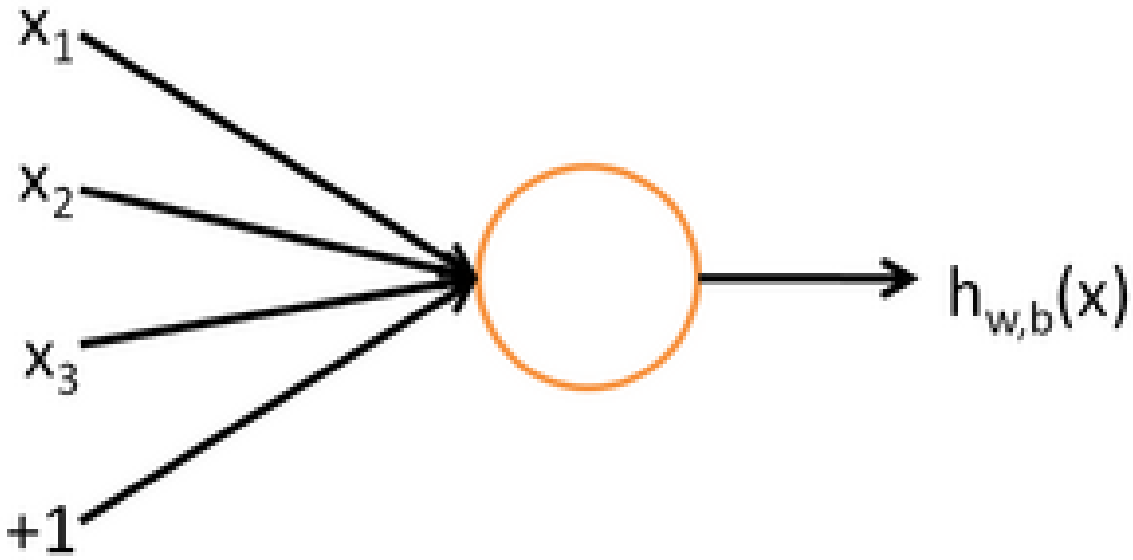
Si caso1 y caso2 se cumplen, entonces realizar acción A

Finalmente en la defuzzificación se adecuan los valores difusos generados en la inferencia en valores del mundo real, que posteriormente se utilizarán en el proceso de control. En la defuzzificación se utilizan métodos matemáticos simples como el método del centroide, promedio ponderado y método de membresía del medio del máximo.

REDES NEURONALES

Las redes neuronales artificiales son un paradigma de aprendizaje automático inspirado en la forma en que operan las neuronas del cerebro de los animales. Se componen de un conjunto masivamente paralelo de unidades de proceso muy simples (neuronas) que se conectan entre sí.

Figura 6. Unidad de cómputo de una red neuronal artificial



Fuente: Universidad de Stanford.

Una neurona es una unidad computacional que toma una entrada x_1,x_2,x_3 (y un +1 término de bias) y produce una salida:

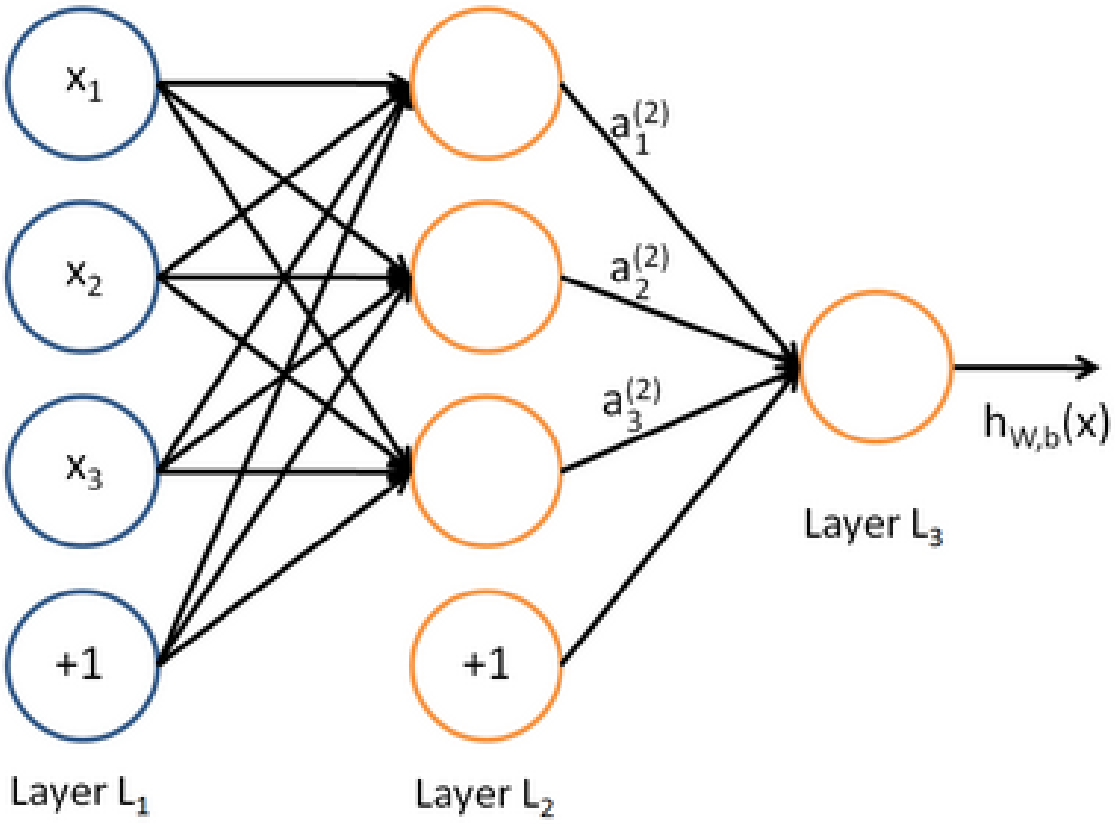
$$h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$$

Donde $f : \Re \mapsto \Re$ es la **función de activación**. Un ejemplo de función de activación *f* es la función sigmoide:

$$f(z) = \frac{1}{1 + \exp(-z)}.$$

En la figura 7 podemos ver el modelo de una red neuronal, los círculos más a la izquierda corresponden a los datos de entrada L1, el círculo +1 representa las unidades de bias que puede presentar el conjunto de datos de entrada, la segunda línea de círculos representan la capa de entrada de la red neuronal L2, la capa de salida está a la derecha. La capa entre la entrada y la salida se denomina “capa oculta” ya que sus valores no son observados en el conjunto de entrenamiento.

Figura 7. Modelo de Red neuronal de 3 capas



Fuente: Universidad de Stanford.

Si n_l es el número de capas, entonces para el caso de la figura $n_l = 3$. Etiquetamos la capa l como L_l, de manera que L_1 será la capa de entradas y la capa L_n la capa de salida. La red neuronal tendrá parámetros (W,b) =), denota el parámetro o peso asociado a la conexión entre la neurona j en la capa l y la neurona i en la capa $l + 1$. Además, es el bias o error asociado con la unidad i en la capa $l + 1$. De esta manera, en la figura $W^{(1)} \in \Re^{3\times3}$ y $W^{(2)} \in \Re^{1\times3}$. Las unidades de bias no tienen entradas y siempre tienen de salida +1. El número de nodos en la capa l se denota s_l.
es la **activación** de la neurona i en la capa l. Para $l = 1$.

Dado un conjunto de parámetros W, b la red neuronal define una hipótesis $h_{W,b}(x)$ que da como resultado un número real. La computación de esta red neuronal estará dada por:

$$
\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}) \\
h_{W,b}(x) &= a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})
\end{aligned}
$$

$z_i^{(l)}$ denota la suma ponderada total de entradas a la neurona *i* de la capa *l*

Ejemplo: $z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)}x_j + b_i^{(1)}$*)*, incluyendo el término de bias, de manera que $a_i^{(l)} = f(z_i^{(l)})$. Se puede escribir de manera más compacta:

$$
\begin{aligned}
z^{(2)} &= W^{(1)}x + b^{(1)} \\
a^{(2)} &= f(z^{(2)}) \\
z^{(3)} &= W^{(2)}a^{(2)} + b^{(2)} \\
h_{W,b}(x) &= a^{(3)} = f(z^{(3)})
\end{aligned}
$$

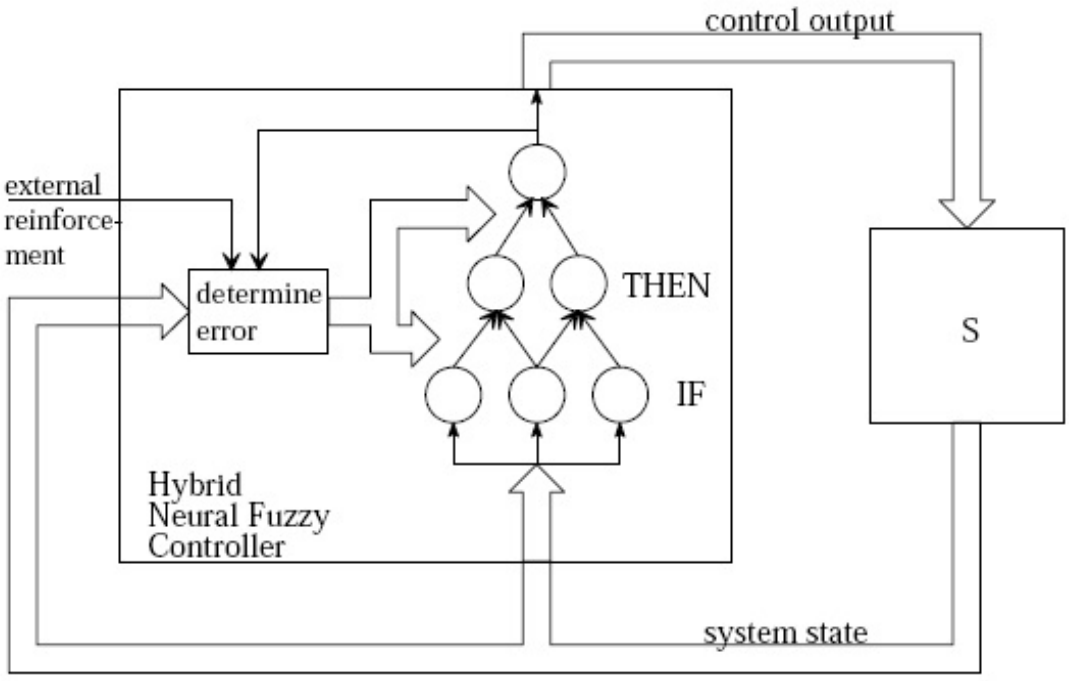
Este es el paso de **propagación hacia adelante**. Las redes neuronales también pueden tener múltiples salidas y usar algoritmos de propagación hacia atrás.

LÓGICA NEURODIFUSA

Un sistema neurodifuso es un sistema difuso que usa un algoritmo de aprendizaje derivado o inspirado por la teoría de redes neuronales para determinar sus parámetros (conjuntos difusos y reglas difusas) procesando ejemplos de datos.

Los sistemas neurodifusos pueden mejorar un sistema difuso existente apoyándose en redes neuronales o lo contrario, mejorar el proceso de aprendizaje de una red neuronal a partir de la lógica difusa.

Figura 8. Esquema de funcionamiento de un sistema neurodifuso



Fuente: NAUCK, Detleft; KRUSE, Rudolf. Choosing Appropriate Neuro-Fuzzy Model. Technical University of Braunschweig, Department of Computer Science, Germany.

Los sistemas neuro-fuzzy son a menudo descritos como una red neuronal de tres capas y propagación hacia atrás. La primera capa representa las variables de entrada, la segunda (capa oculta) representa las reglas difusas y la tercera representa las variables de salida. Tipos de sistemas neuro fuzzy son: concurrentes, cooperativos e híbridos, en estos últimos nos concentramos por ser el modelo más adecuado para la clasificación de datos, ya que nos aporta interpretabilidad de las reglas del entrenamiento de la red neuronal, configurar automáticamente los conjuntos difusos y reducir tiempo gracias a que podemos incluir conocimiento previo al sistema.

PATRÓN DE DISEÑO INTÉRPRETE

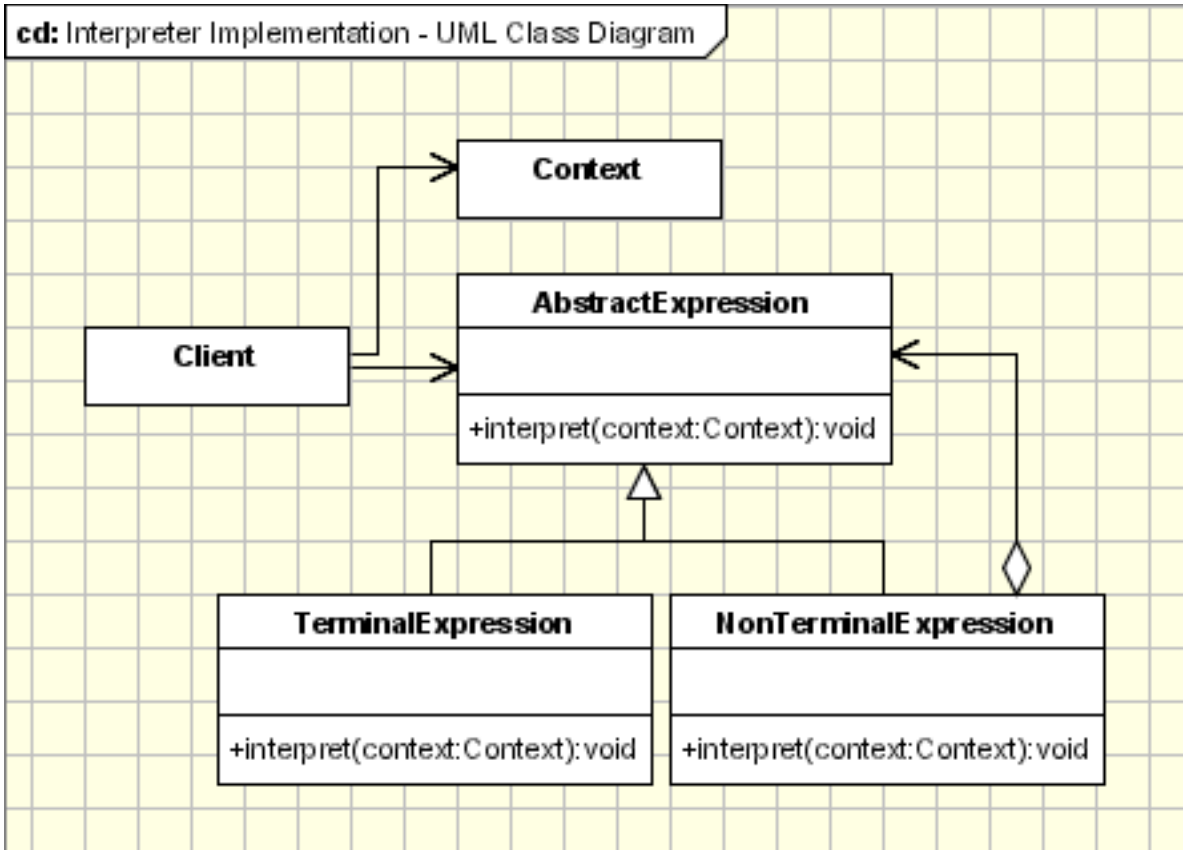
El área de diseño e implementación de sistemas de software requiere de buenas prácticas para cada etapa de su ciclo de vida. Un patrón de diseño es la descripción de un problema computacional común y la descripción de un pequeño

conjunto de clases y su estructura de interacción que ayuda a resolver dicho problema. Permitiendo reusar código y resolver problemas en diferentes contextos.

Las descripciones de los patrones se realizan mediante una plantilla que incluye:

- Nombre del patrón.
- Descripción del problema que pretende solucionar.
- Descripción de la solución (que estructuras y clases usará).
- Efectos positivos y consecuencias del uso del patrón.

Figura 9. Implementación del patrón intérprete



Fuente: OODesign.com.

El patrón de diseño intérprete fue descrito en términos de gramática formal por los autores mencionados como “The Gang of Four” pero su área de aplicación puede ser extendida.

Propósito del patrón: dado un lenguaje, define una representación para su gramática junto con un intérprete que usa la representación para interpretar oraciones. Mapea un dominio a un lenguaje, el lenguaje a una gramática y la gramática a un diseño jerárquico orientado a objetos.

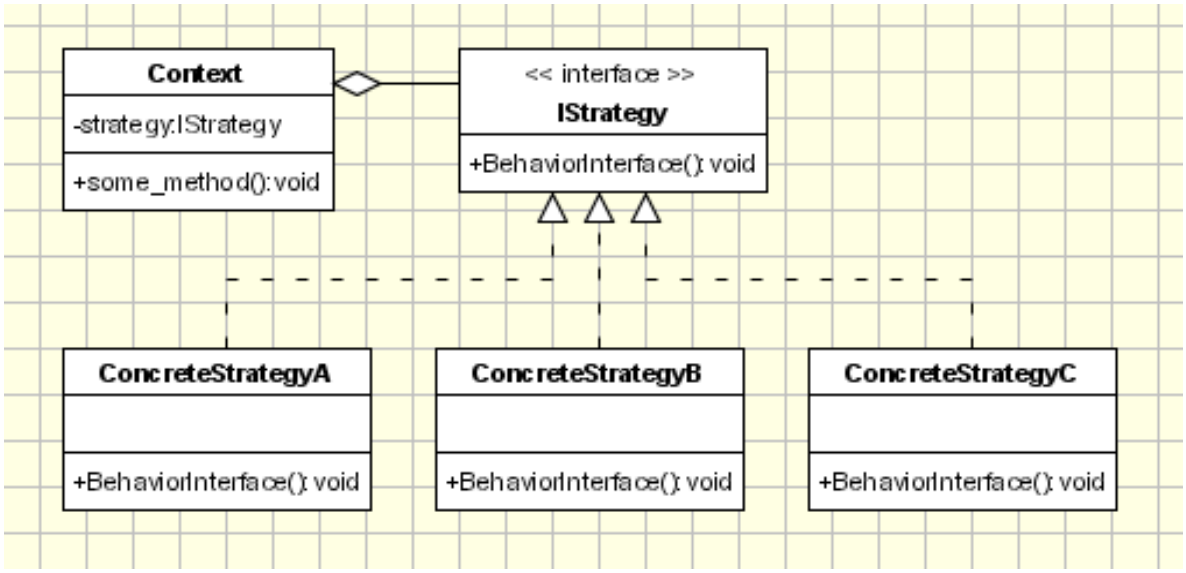
Aplicabilidad:

- El patrón intérprete es usado exhaustivamente para definir gramáticas, tokenizar o marcar los datos de entrada y almacenarlos.
- Motores de normas, para validar condiciones.
- Puede ser utilizado para añadir funcionalidad al patrón Objeto compuesto.

PATRÓN DE DISEÑO ESTRATEGIA

El patrón estrategia tiene como fin definir una familia de algoritmos, encapsularlos y hacerlos intercambiables, así el patrón permite que el algoritmo varíe independientemente de qué cliente lo usa.

Figura 10. Implementación del patrón estrategia



Fuente: OODesign.com.

El patrón de diseño estrategia es de utilidad cuando existen clases que difieren solamente en su comportamiento y se quiere aislar los algoritmos en diferentes clases para tener la habilidad de seleccionar diferentes algoritmos en tiempo de ejecución.

ANEXO C. DIAGRAMAS UML

Figura 11. Diagrama de casos de uso, caso “Preprocesar”

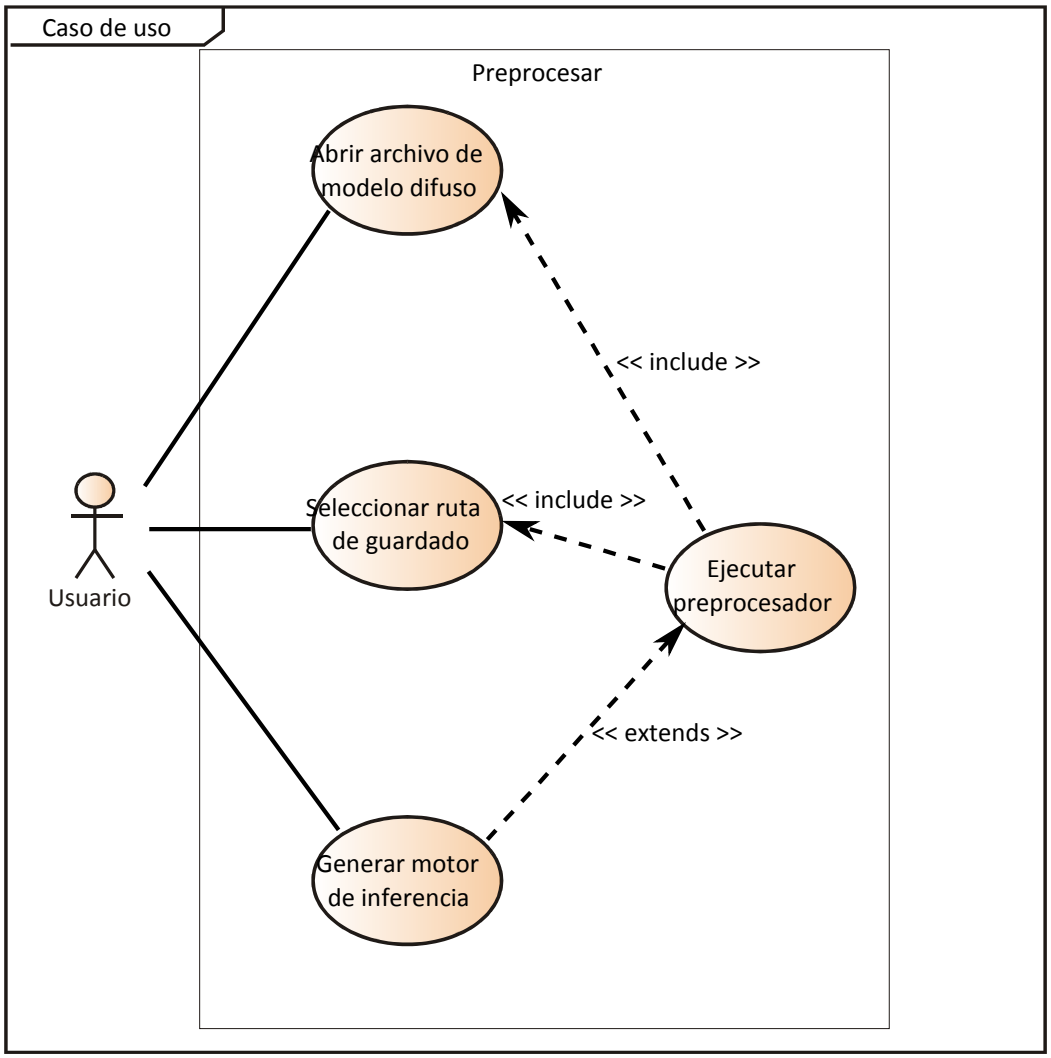


Tabla 1. Caso de uso “preprocesar”

Nombre	**Preprocesar**
Actor(es)	Usuario
Descripción	El usuario ingresa un archivo de modelo difuso, una ruta de guardado de los archivos (motor de inferencia y librerías) e invoca la generación del motor de inferencia.
Precondición(es)	Se debe contar con un archivo de modelo difuso con sintaxis y semántica válidas.
Flujo principal	Se selecciona el archivo de modelo difuso, se elige la ruta donde se van a almacenar los archivos y se genera el motor de inferencia y una copia de las librerías que dan soporte al sistema difuso contenido en el archivo de modelo.
Poscondición(es)	Ninguna.
Subflujos	Interpretar archivo de modelo difuso, normalizar el sistema difuso y escribir los archivos de salida.
Excepciones	Ingreso de archivo de modelo difuso con sintaxis y/o semántica no válidas.

Figura 12. Diagrama de clases de la herramienta de manejo de archivos de entrada

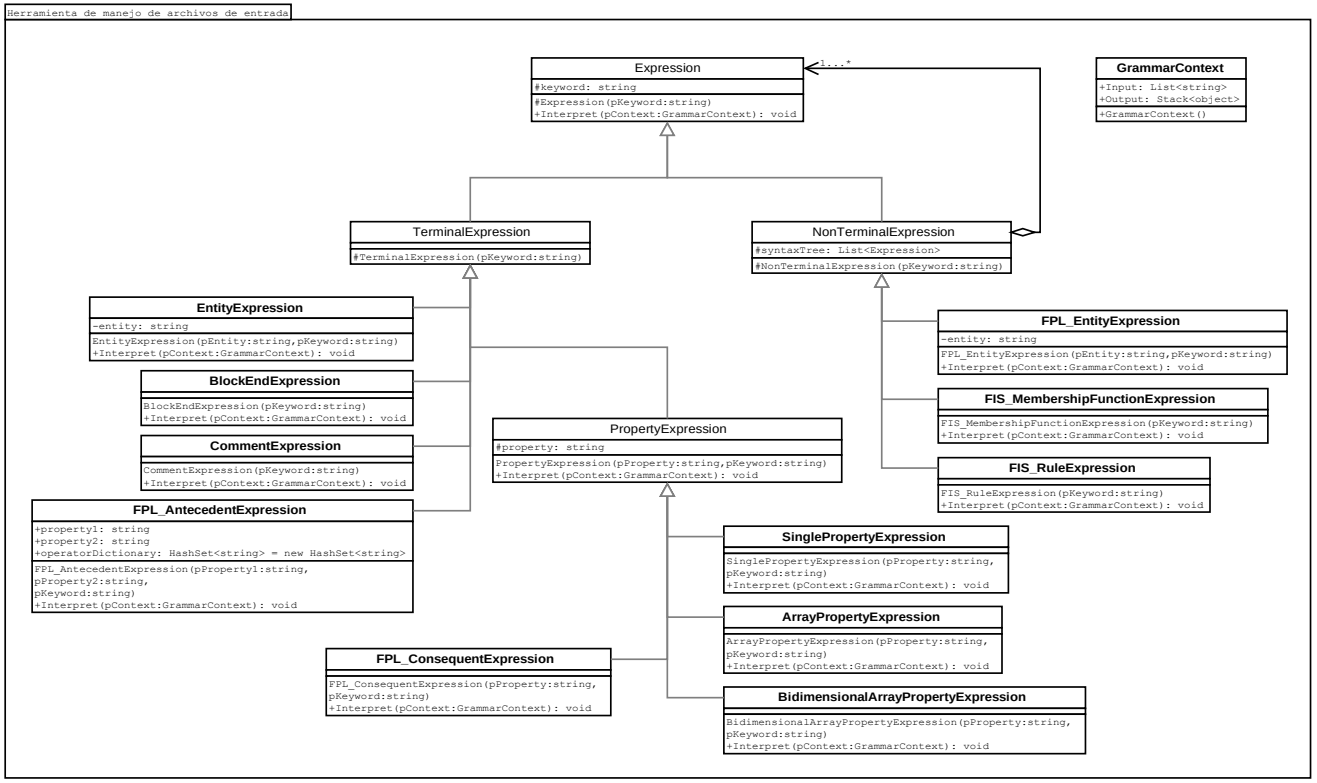


Figura 13. Diagrama de clases del modelo de datos

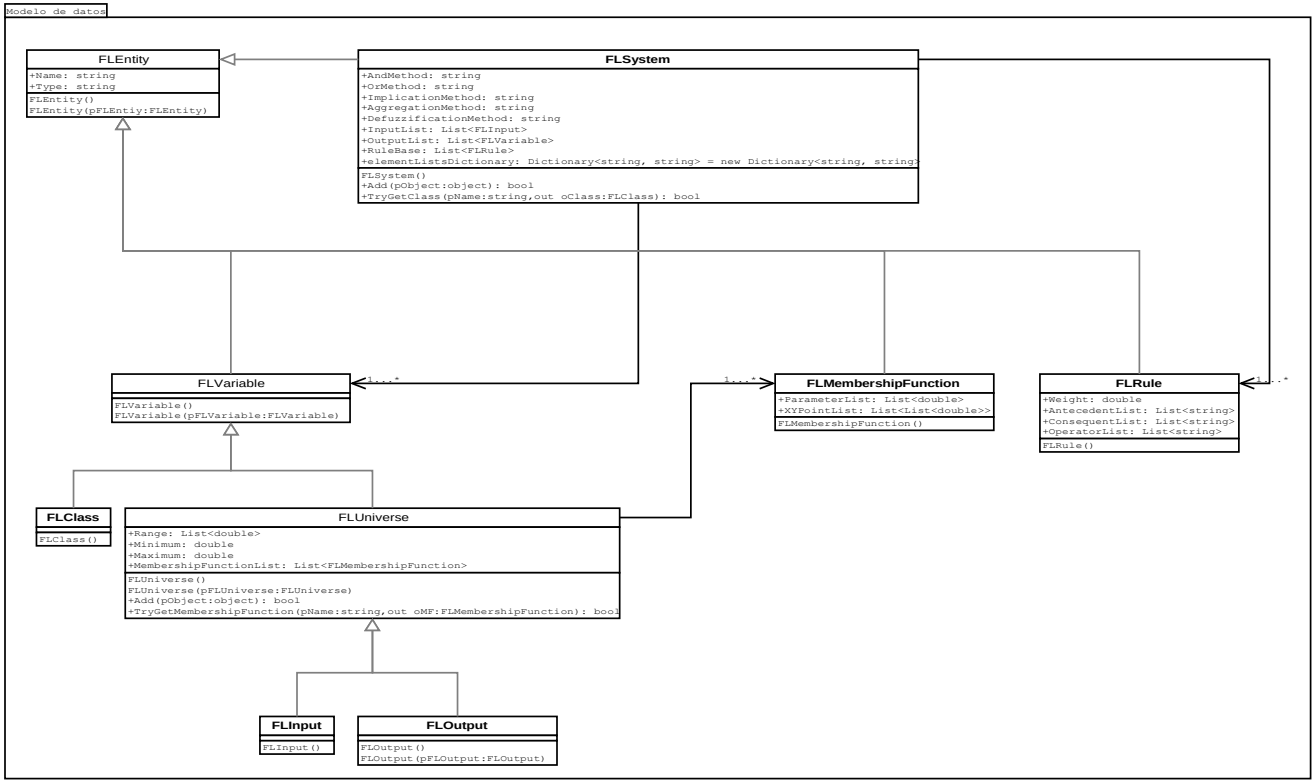


Figura 14. Diagrama de clases del módulo “herramienta de manejo de archivos de salida”

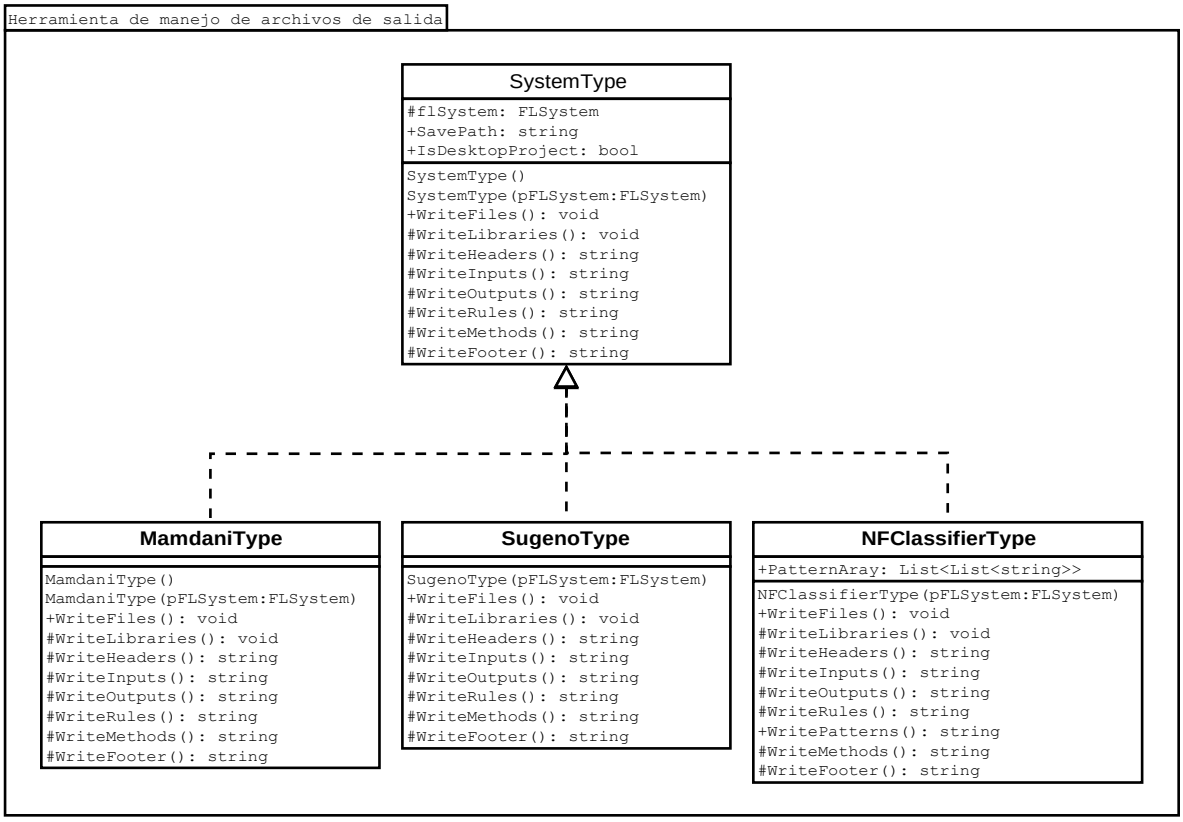


Figura 15. Diagrama de clases del Preprocesador de Fuzzy Inference System para Plataformas Embebidas (FISPES)

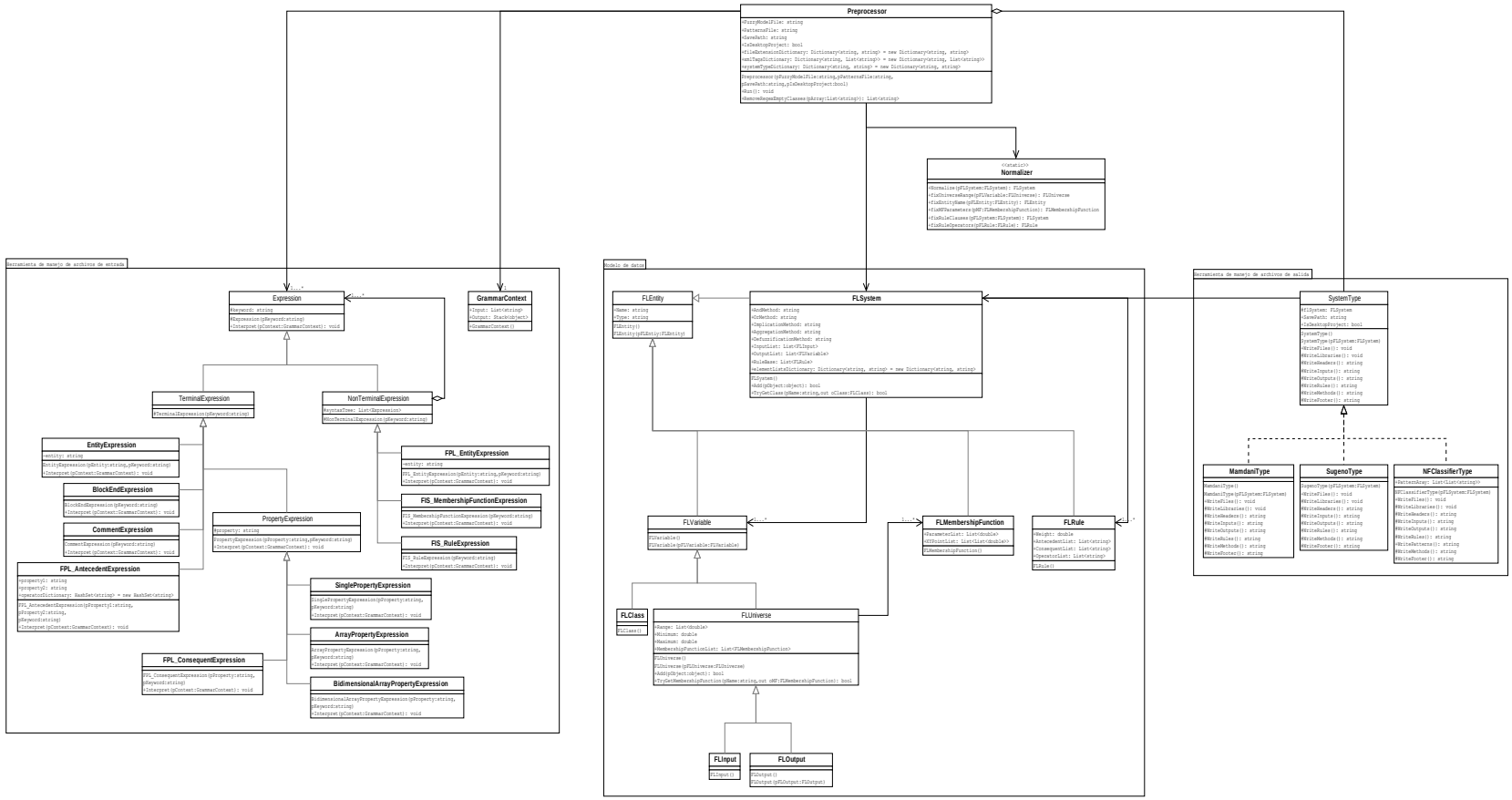


Figura 16. Diagrama de máquina de estados del preprocesador FISPES

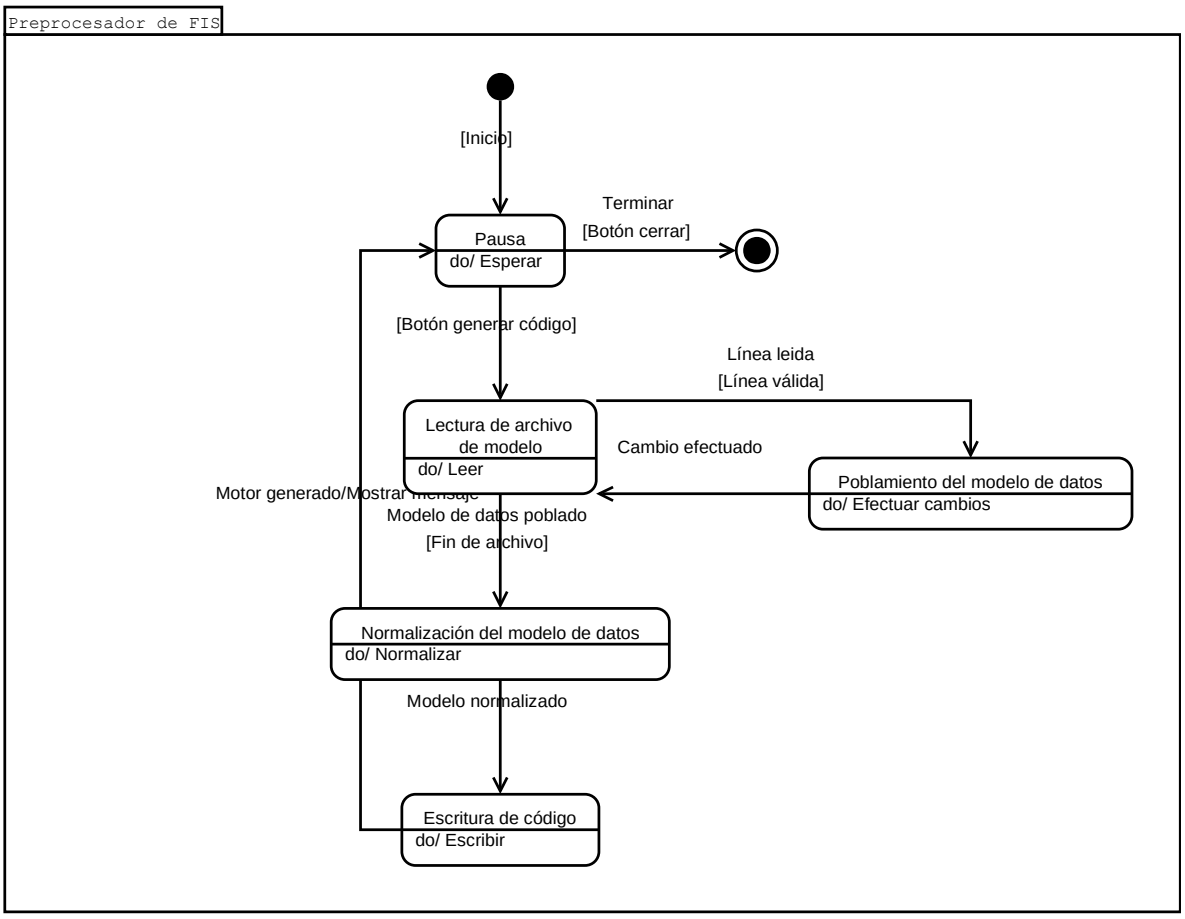


Figura 17. Diagrama de máquina de estados para el motor de inferencia tipo Mamdani

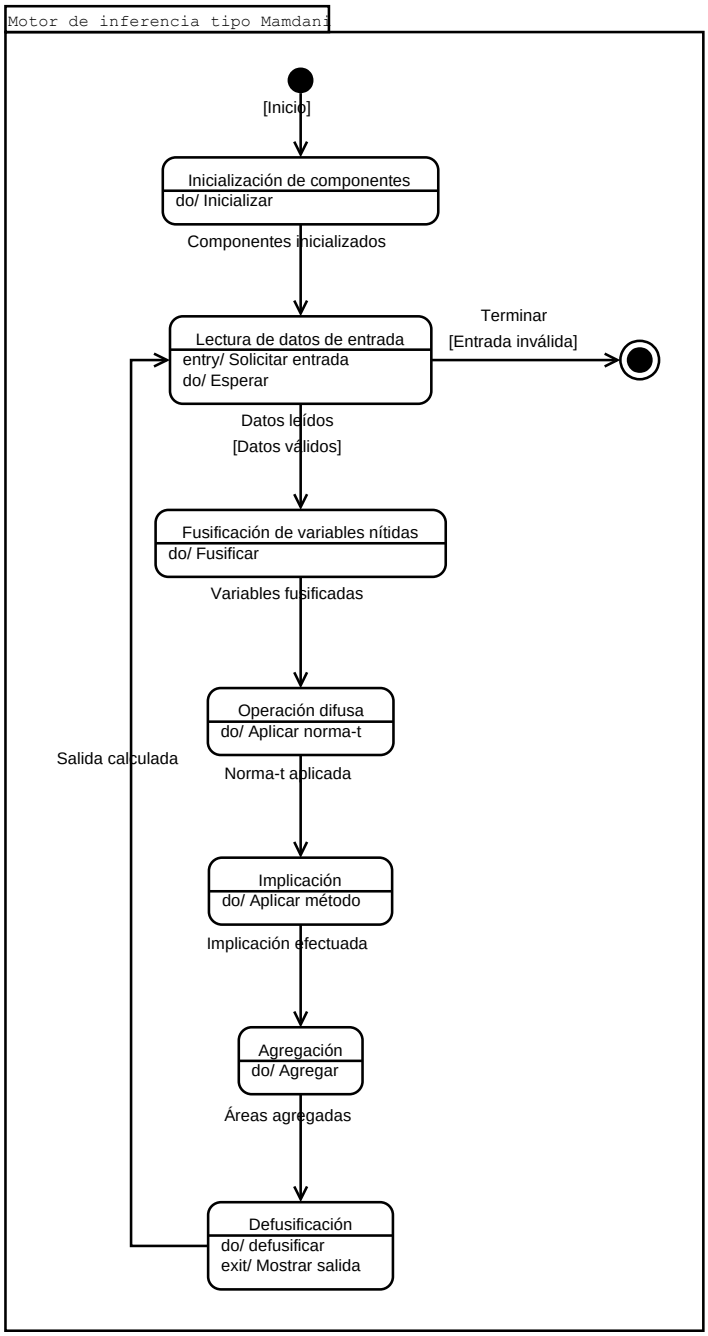


Figura 18. Diagrama de máquina de estados para el motor de inferencia tipo Sugeno

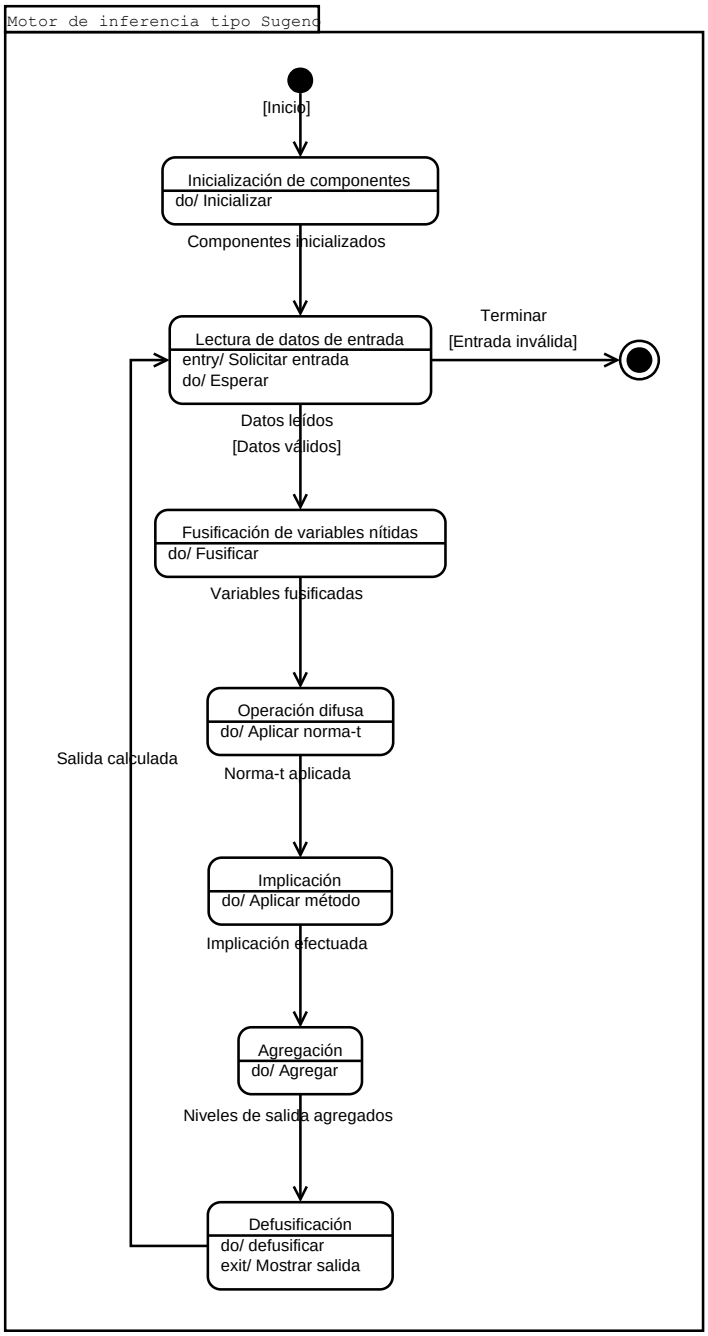


Figura 19. Diagrama de máquina de estados para el motor de inferencia tipo Clasificador Neurodifuso

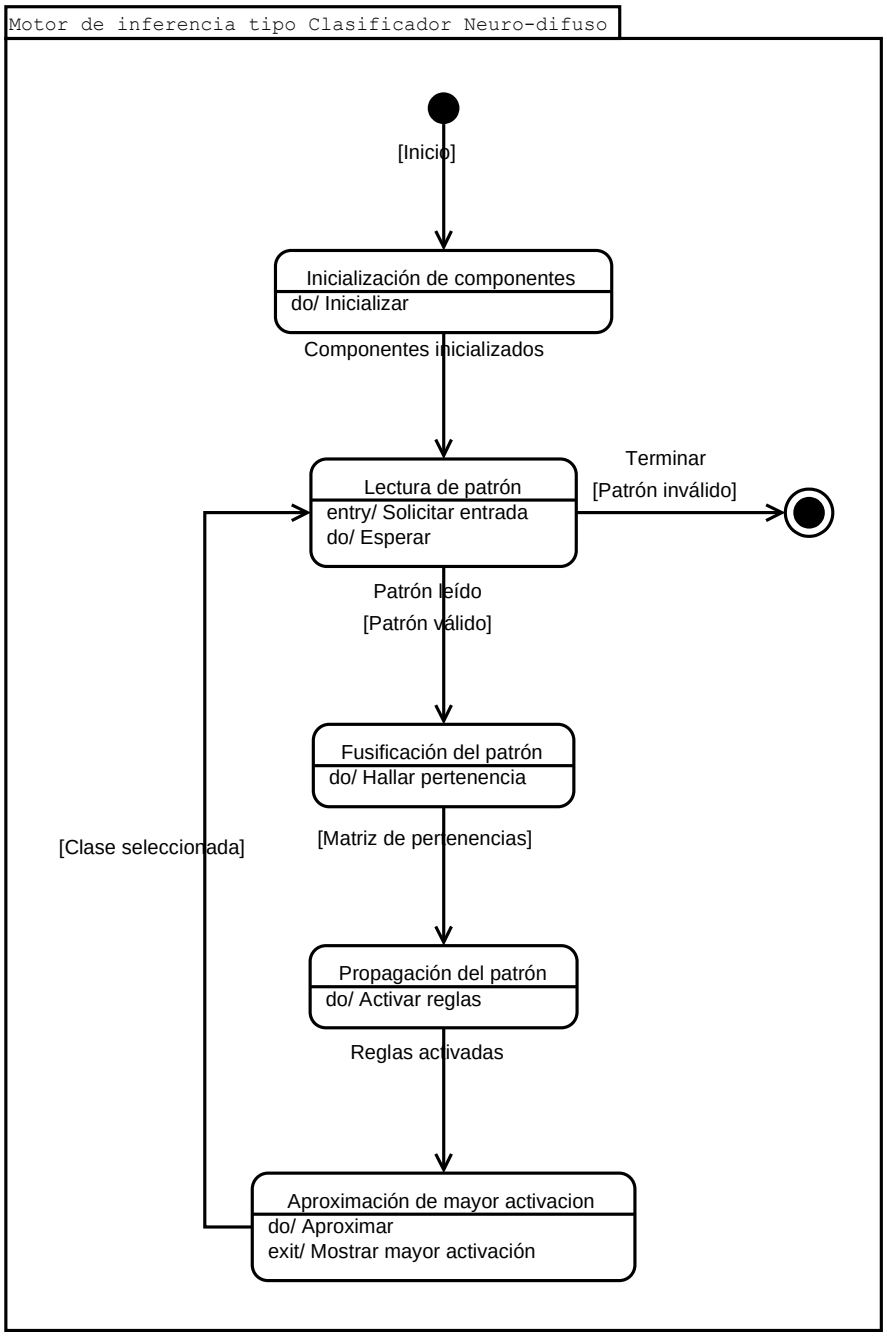
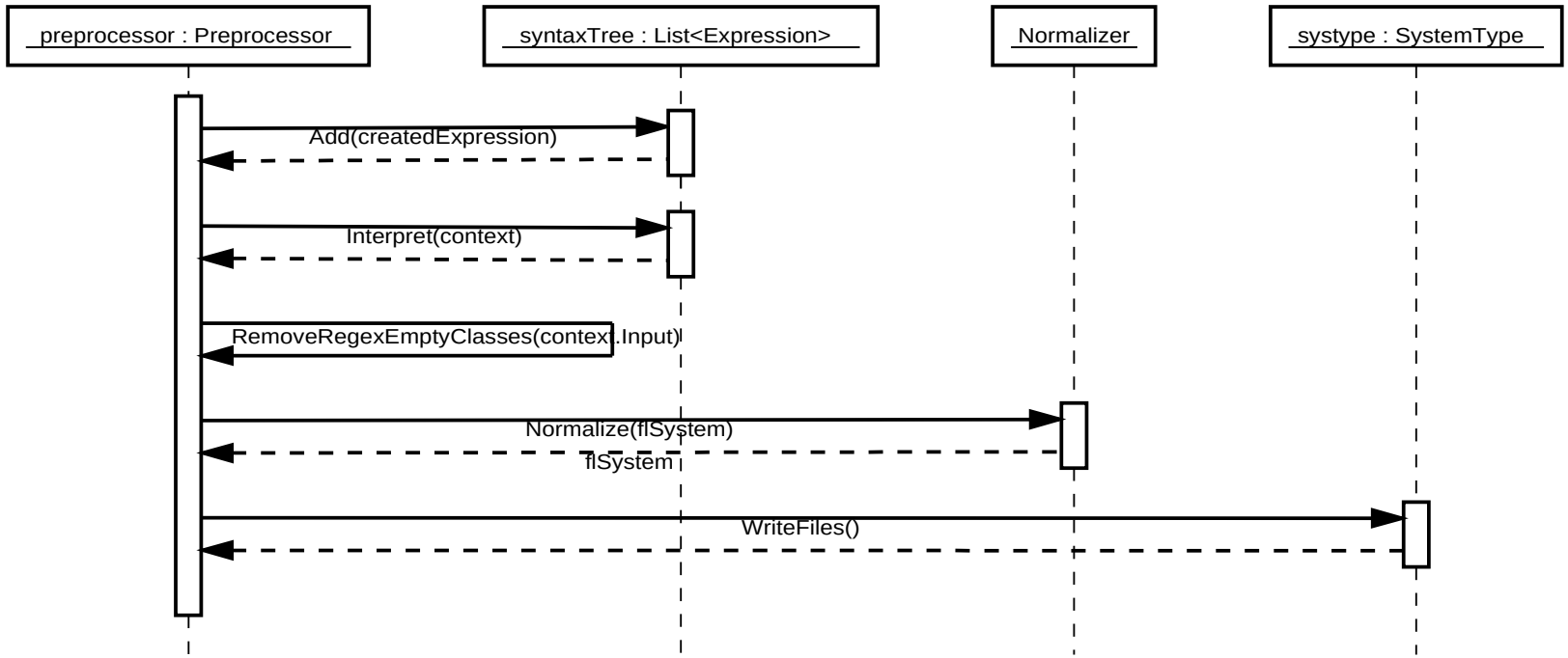


Figura 20. Diagrama de secuencia del preprocesador FISPES



ANEXO D. REPLANTEAMIENTO DE REQUISITOS

Durante la puesta en marcha de la primera etapa del ciclo de desarrollo se llevaron a cabo reuniones periódicas con el codirector del proyecto, el profesor M.Sc. Jorge Enrique Meneses Flórez, para socializar los avances realizados, las dificultades presentadas y establecer qué ajustes se debían realizar en el momento. Se realizó una presentación al finalizar la primera etapa del ciclo de desarrollo donde se expuso el prototipo obtenido, lo cual llevo al replanteamiento de los requerimientos del proyecto y dió inicio a la siguiente fase del trabajo.

A continuación se mencionan los requerimientos que surgieron de la exposición del primer prototipo.

Tabla 2. Requisitos planteados al final de la primera etapa del ciclo de desarrollo

No. de Item	Requerimiento	Componente	Tipo de requerimiento
Item 1	Agregar un módulo que se encargue de realizar correcciones sobre los atributos del sistema difuso para evitar errores en la ejecución del motor de inferencia generado.	Preprocesador.	Funcional.
Item 2	Modificar el algoritmo de sistema difuso para permitir la ejecución continua (en bucle) del motor de inferencia y la captura de datos desde una ventana deslizante.	Herramienta de manejo de archivos de salida.	Funcional.
Item 3	Diseñar una interfaz de usuario gráfica que le permita al usuario ajustar los parámetros del preprocesador sin recurrir a la edición del código fuente.	Preprocesador.	Funcional
Item 4	Permitir al usuario el ingreso de la ruta del archivo de modelo difuso y de guardado de los archivos generados a través de la interfaz gráfica.	Preprocesador.	Funcional
Item 5	Personalizar la interfaz del preprocesador para imprimirle carácter institucional (logotipo de la universidad e ícono de la aplicación).	Preprocesador	No funcional
Item 6	Incluir mensajes de estado de la operación del motor de inferencia.	Herramienta de manejo de archivos de salida.	No funcional

ANEXO E. PLANES DE PRUEBAS

PLAN DE PRUEBAS DE INTEGRACIÓN DEL PREPROCESADOR

1. Elementos de la prueba

Los módulos del preprocesador FISPES:

- Herramienta de manejo de archivos de entrada.
- Modelo de datos.
- Herramienta de manejo de archivos de salida.

2. Características a ser probadas

Integridad en los datos al pasar por cada módulo del preprocesador.

3. Características que no se probarán

Eficiencia en tiempo y consumo de recursos computacionales.

4. Metodología

Ingresar un archivo de modelo difuso de ejemplo “Iris” y verificar para cada módulo si los datos almacenados tienen relación con los que contiene el modelo difuso.

5. Criterio de aprobación o fallo de los elementos

Se verifica que los datos en los módulos están relacionados con los datos contenidos en archivo de modelo.

6. Criterio de suspensión y requerimientos para la reanudación

Los datos en los módulos perdieron la relación con la información contenida en el modelo.

7. Tareas de la prueba

- Ingresar archivo de modelo difuso.
- Verificar los datos en la pila de salida al finalizar la ejecución de la herramienta de manejo de archivos de entrada.
- Verificar los datos en el modelo de datos al finalizar la ejecución del submódulo normalizador de sistema difuso.

- Verificar los datos registrados en el motor de inferencia difusa al finalizar la ejecución de la herramienta de manejo de archivos de salida.

8. Necesidades del entorno

- IDE Visual Studio.

9. Riesgos y contingencias

Ninguno.

10. Aprobaciones

Se confirmó la integridad en los datos a través de su paso por los módulos del preprocesador.

PLAN DE PRUEBAS DE SISTEMA DEL PREPROCESADOR

1. Elementos de la prueba

Preprocesador FIS para Plataformas Embebidas.

2. Características a ser probadas

Eficacia en la creación de motores de inferencia equivalentes a los modelos difusos ingresados en el preprocesador FISPES.

3. Caraccterísticas que no se probarán

Eficiencia en tiempo y consumo de recursos computacionales.

4. Metodología

Ingresar un archivo de modelo difuso de ejemplo (IRIS) y verificar que el motor de inferencia clasifica los patrones de ejemplo de manera correcta.

5. Criterio de aprobación o fallo de los elementos

Los patrones presentan la clasificación esperada.

6. Criterio de suspensión y requerimientos de la reanudación

Más del 51% de los patrones es clasificado erróneamente por el motor de inferencia difusa.

7. Tareas de la prueba

- Ingresar archivo de modelo difuso de prueba.
- Cargar el motor de inferencia generado en la plataforma embebida.
- Verificar que el resultado de la clasificación de los patrones de prueba es el esperado.

8. Necesidades del entorno

- IDE Visual Studio.
- IDE Kinetis Design Studio.
- Plataforma embebida Freescale TWR-K70F120M.

9. Riesgos y contingencias

Ninguno.

10. Aprobaciones

Se confirmó la eficacia en la creación de motores de inferencia equivalentes a los modelos difusos ingresados en el preprocesador.

ANEXO F. FIGURAS Y TABLAS ADICIONALES

Figura 21. Archivo de modelo difuso “Temperaturas” en lenguaje FPL

```
PROJECT ModeloTemp
	TYPE Mamdani

	VAR sustancia1
		TYPE float
		MIN 0
		MAX 1

		MEMBER frio
			POINTS -0.36,0 -0.04,1 0.04,1 0.36,0
		END

		MEMBER tibio
			POINTS 0.1,0 0.5,1 0.9,0
		END

		MEMBER caliente
			POINTS 0.64,0 0.96,1 1.04,1 1.36,0
		END
	END

	VAR sustancia2
		TYPE float
		MIN 0
		MAX 1

		MEMBER frio
			POINTS -0.36,0 -0.04,1 0.04,1 0.36,0
		END

		MEMBER tibio
			POINTS 0.1,0 0.5,1 0.9,0
		END

		MEMBER caliente
```

```
                    POINTS 0.64,0 0.96,1 1.04,1 1.36,0
            END
    END

    VAR mezcla
            TYPE float
            MIN 0
            MAX 1
            DEFAULT 0

            MEMBER frio
                    POINTS -0.36,0 -0.04,1 0.04,1 0.36,0
            END

            MEMBER tibio
                    POINTS 0.1,0 0.5,1 0.9,0
            END

            MEMBER caliente
                    POINTS 0.64,0 0.96,1 1.04,1 1.36,0
            END
    END

    RULEBASE Temperaturas
            RULE Rule1
                    IF (sustancia1 IS frio) AND (sustancia2 IS frio)
                    THEN mezcla = frio
            END

            RULE Rule2
                    IF (sustancia1 IS tibio) AND (sustancia2 IS frio)
                    THEN mezcla = frio
            END

            RULE Rule3
                    IF (sustancia1 IS caliente) AND (sustancia2 IS frio)
                    THEN mezcla = tibio
            END

            RULE Rule4
```

```
77                      IF (sustancia1 IS frio) AND (sustancia2 IS tibio)
78                      THEN mezcla = frio
79              END
80
81              RULE Rule5
82                      IF (sustancia1 IS tibio) AND (sustancia2 IS tibio)
83                      THEN mezcla = tibio
84              END
85
86              RULE Rule6
87                      IF (sustancia1 IS caliente) AND (sustancia2 IS tibio)
88                      THEN mezcla = caliente
89              END
90
91              RULE Rule7
92                      IF (sustancia1 IS frio) AND (sustancia2 IS caliente)
93                      THEN mezcla = tibio
94              END
95
96              RULE Rule8
97                      IF (sustancia1 IS tibio) AND (sustancia2 IS caliente)
98                      THEN mezcla = caliente
99              END
100
101             RULE Rule9
102                     IF (sustancia1 IS caliente) AND (sustancia2 IS caliente)
103                     THEN mezcla = caliente
104             END
105     END
106
107     CONNECT
108             FROM sustancia1
109             TO Temperaturas
110     END
111
112     CONNECT
113             FROM sustancia2
114             TO Temperaturas
115     END
116
```

```
        CONNECT
                FROM Temperaturas
                TO mezcla
        END
END
```

Figura 22. Archivo de configuración del lenguaje FIS

```
<?xml version="1.0"?>
<Language name="FIS" regularExpression="[^A-Za-z0-9_\-\.\%\s\n]+">
        <!-- Entity Expressions -->
        <EntityExpression entity="FLSystem" keyword="System"/>
        <EntityExpression entity="FLInput" keyword="Input"/>
        <EntityExpression entity="FLOutput" keyword="Output"/>
        <EntityExpression entity="FLClass" keyword="Class"/>
        <!-- FIS-specific Rule Expressions -->
        <FIS_RuleExpression entity ="FLRule" keyword=""/>
        <!-- FIS-specific Membership Function Expresions -->
        <FIS_MembershipFunctionExpresion entity="FLMembershipFunction"
        keyword="MF"/>
        <!-- Property Expressions -->
        <SinglePropertyExpression property="Name" keyword="Name"/>
        <SinglePropertyExpression property="Type" keyword="Type"/>
        <SinglePropertyExpression property="AndMethod" keyword="AndMethod"/>
        <SinglePropertyExpression property="OrMethod" keyword="OrMethod"/>
        <SinglePropertyExpression property="ImplicationMethod" keyword="ImpMethod"/>
        <SinglePropertyExpression property="AggregationMethod" keyword="AggMethod"/>
        <SinglePropertyExpression property="DefuzzificationMethod"
        keyword="DefuzzMethod"/>
        <!-- Array Property Expressions -->
        <ArrayPropertyExpression property="Range" keyword="Range"/>
        <!-- Block End Expressions -->
        <BlockEndExpression keyword=""/>
        <!-- Comment Expressions -->
        <CommentExpression keyword="%"/>
        <!-- Unused Expressions -->
        <!--
        <PropertyExpression property="NumMembershipFunctions" keyword="NumMFs"/>
        <PropertyExpression property="Version" keyword="Version"/>
```

```
        <PropertyExpression property="NumInputs" keyword="NumInputs"/>
        <PropertyExpression property="NumOutputs" keyword="NumOutputs"/>
        <PropertyExpression property="NumRules" keyword="NumRules"/>
        -->
</Language>
```

Figura 23. Archivo de configuración del lenguaje FPL

```
<?xml version="1.0"?>
<Language name="FPL" regularExpression="[^A-Za-z0-9_\=\-\.\,\n]+">
        <!-- Entity Expressions -->
        <FPL_EntityExpression entity="FLSystem" keyword="PROJECT"/>
        <FPL_EntityExpression entity="FLUniverse" keyword="VAR"/>
        <FPL_EntityExpression entity="FLClass" keyword="CLASS"/>
        <FPL_EntityExpression entity="FLMembershipFunction" keyword="MEMBER"/>
        <FPL_EntityExpression entity="FLRule" keyword="RULE"/>
        <EntityExpression entity="FLConnection" keyword="CONNECT"/>
        <!-- FPL-specific Rule Expressions -->
        <FPL_AntecedentExpression property1 ="AntecedentList" property2
        ="OperatorList" keyword="IF"/>
        <FPL_ConsequentExpression property="ConsequentList" keyword="THEN"/>
        <!-- Property Expressions -->
        <SinglePropertyExpression property="Type" keyword="TYPE"/>
        <SinglePropertyExpression property="Minimum" keyword="MIN"/>
        <SinglePropertyExpression property="Maximum" keyword="MAX"/>
        <SinglePropertyExpression property="Weight" keyword="WITH"/>
        <SinglePropertyExpression property="FromEntity" keyword="FROM"/>
        <SinglePropertyExpression property="ToEntity" keyword="TO"/>
        <!-- Bidimensional Array Property Expressions -->
        <BidimensionalArrayPropertyExpression property="XYPointList"
        keyword="POINTS"/>
        <!-- Block End Expressions -->
        <BlockEndExpression keyword="END"/>
        <!-- Comment Expressions -->
        <CommentExpression keyword="//"/>
</Language>
```

Figura 24. Diagrama de flujo de la función escribirEntradas

Figura 25.Diagrama de flujo de la función escribirSalidas

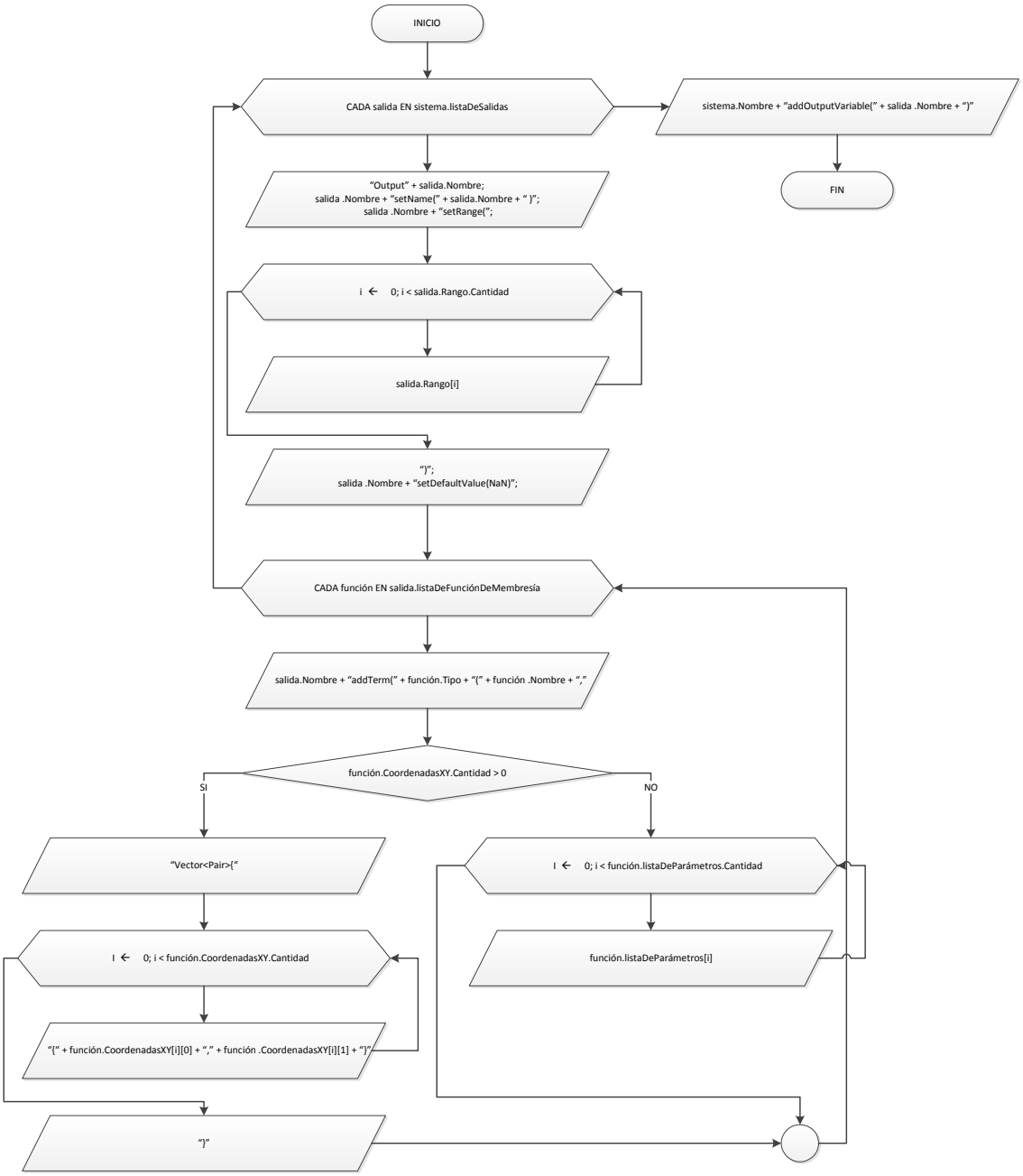
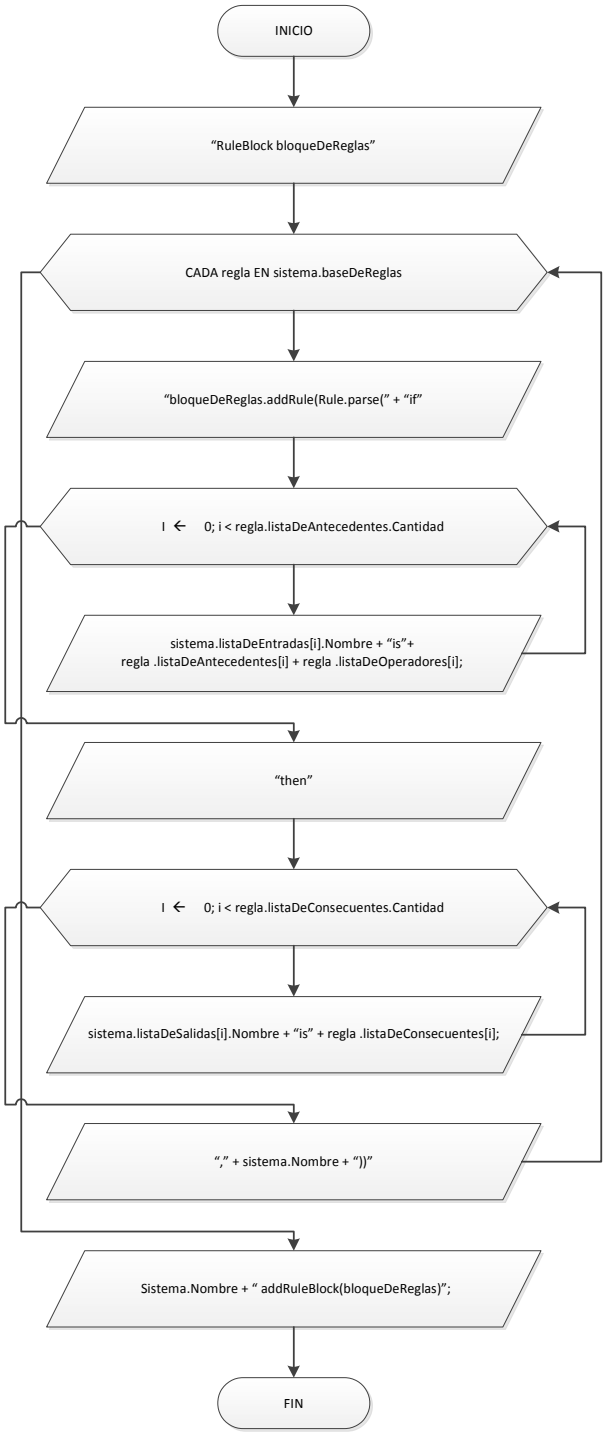


Figura 26. Diagrama de flujo de la función escribirReglas



ANEXO G. BIBLIOGRAFÍA DE LOS ANEXOS

ALEGRE, Lideniro; MOROOKA, C.K; DA ROCHA, A.F. Intelligent Diagnosis of Rod Pumping Problems. Society of Petroleum Engineers. [En línea]. 1993. [Consultado en octubre de 2014]. ISBN: 978-1-55563-481-0. Disponible en: http://dx.doi.org/10.2118/26516-MS

AMITRANO, Sergio; DI CHIAZZA, Rodolfo; LORENZO, Jorge. Sistema experto basado en lógica difusa para la Central Nuclear Embalse. [En línea]. Buenos Aires: Universidad de Buenos Aires. 1998. [Consultado en enero de 2015]. Disponible en: https://www.dc.uba.ar/inv/tesis/licenciatura/1998/amitrano.pdf

Baker Hughes Incorporated. Intelligent Well Systems. En: Baker Hughes. [En línea]. Recuperado en junio de 2015. Disponible en: http://www.bakerhughes.com/products-and-services/production/intelligent-production-systems/intelligent-well-systems

BARNES, David; KÖLLING, Michael. Programación Orientada a Objetos con Java. Madrid: Prentice Hall, 2007. ISBN: 9788483227916

CABALLERO, Fabio; SALAMANCA, Julian. Herramienta De Aplicación De Software Para Clasificación De Patrones De Datos Implementando Una Arquitectura Neuro-Fuzzy. Bucaramanga: Universidad Industrial de Santander, 2011.

Carnegie Mellon University School of Computer Science. What is Fuzzy Logic. En: Carnegie Mellon University School of Computer Science. [En línea]. Recuperado en octubre de 2014. Disponible en: https://www.cs.cmu.edu/Groups/AI/html/faqs/ai/fuzzy/part1/faq-doc-2.html

CORREA, Jose F. LEPKISON, Herman. BITTENCOURT, Antonio C. Intelligent Distributed Management System for Automated Wells (SGPA): Expereince and Results. Society of Petroleum Engineers. [En línea]. 2003. [Consultado en octubre de 2014]. ISBN: 978-1-55563-152-9. Disponible en: http://dx.doi.org/10.2118/84225-MS

GAFA, Carmel. Fuzzscript: A Fuzzy Logic Control Language. En: CodeProject. [En línea]. Recuperado en junio de 2015. Disponible en http://www.codeproject.com/Articles/47795/FuzzScript-A-Fuzzy-Logic-Control-Language

Halliburton. Intelligent Completions. En: Halliburton. [En línea]. Recuperado en agosto de 2015. Disponible en http://www.halliburton.com/en-US/ps/well-dynamics/well-completions/intelligent-completions/default.page?node-id=hfqel9vs&nav=en-US_completions_public

HULL, Rob. What is a Mature field?. En: Halliburton. [En línea]. Recuperado en agosto de 2015. Disponible en http://halliburtonblog.com/what-is-a-mature-field

JANSEN, J. D. Smart wells. Delft University of Technology, Department of Applied

Earth Sciences, Shell E&P Technology Applications and Research, Rijswijk. [En línea]. 2001. [Consultado en agosto de 2014]. Disponible en: http://www.citg.tudelft.nl/fileadmin/Faculteit/CiTG/Over_de_faculteit/Afdelingen/Afdeling_Geotechnologie/secties/Sectie_Petroleum_Engineering/staff/Jansen,_J._D./Publications/doc/Smartwells.pdf

KOVARIK, Bill. The oil reserve fallacy 3. En: Prof. Kovarik's web. [En línea]. Recuperado en octubre de 2014. Disponible en http://www.environmentalhistory.org/billkovarik/about-bk/research/oil-fallacy1/the-oil-reserve-fallacy-3

MENESES, Edxon; GARAVITO, Fredy. Sistema Neuro-fuzzy: Prospectiva de aplicación en la detección de fallas en equipos de subsuelo de unidades de levantamiento mecánico. Bucaramanga: Universidad Industrial de Santander, 2014.

NAUCK, Detlef D. Fuzzy data analysis with NEFCLASS. International Journal of Approximate Reasoning, Volume 32. [En línea]. 2003. [Consultado en julio de 2015]. ISBN: 0-7803-7078-3. Disponible en: http://www.sciencedirect.com/science/article/pii/S0888613X02000798

NAUCK, Detlef D. What are Neuro-Fuzzy Systems?. En: Arbeitsgruppe Computational Intelligence, Universität Magdeburg, Fakultät für Informatik. [En línea]. Recuperado en enero de 2015. Disponible en http://fuzzy.cs.uni-magdeburg.de/nfdef.html

Object Oriented Design. Interpreter Pattern. En: oodesign.com. [En línea]. Recuperado en noviembre de 2015. Disponible en: http://www.oodesign.com/interpreter-pattern.html

Object Oriented Design. Strategy Pattern. En: oodesign.com. [En línea]. Recuperado en noviembre de 2015. Disponible en: http://www.oodesign.com/strategy-pattern.html

PARSHALL, Joel. Mature Fields Hold Big Expansion Opportunity. Society of Petroleum Engineers. [En línea]. 2012. [Consultado en ocutbre de 2014]. ISSN: 0149-2136. Disponible en: http://dx.doi.org/10.2118/1012-0052-JPT

RAWI, Zaid. Machinery Predictive Analytics. Society of Petroleum Engineers. [En línea]. 2010. [Consultado en octubre de 2014]. ISBN: 978-1-55563-284-7. Disponible en: http://dx.doi.org/10.2118/128559-MS

Schlumberger. Intelligent Completions. En: Schlumberger. [En línea]. Recuperado en noviembre de 2014. Disponible en: http://www.slb.com/services/completions/intelligent.aspx

Stacey Higginbotham. BP teams up with GE to make its oil wells smart. En: Fortune. [En línea]. Recuperado en noviembre de 2014. Disponible en http://fortune.com/2015/07/08/bp-teams-up-with-ge-to-make-its-oil-wells-smart/

Stanford University. Neural Networks. En: Unsupervised Feature Learning and Deep Learning. [En línea]. Recuperado en mayo de 2015. Disponible en

http://ufldl.stanford.edu/wiki/index.php/Neural_Networks

Statoil. Advanced wells. En: Statoil. [En línea]. Recuperado en julio de 2015. Disponible en: http://www.statoil.com/en/TechnologyInnovation/OptimizingReservoirRecovery/SmartWells/Pages/default.aspx

Weatherford. Production and Reservoir Monitoring. En: Weatherford. [En línea]. Recuperado en julio de 2015. Disponible en: http://www.weatherford.com/products-services/production/production-and-reservoir-monitoring