OPTIMIZACIÓN DE UN MOTOR DE INFERENCIA DIFUSA MEDIANTE LA TECNOLOGÍA CUDA (Compute Unified Device Architecture).

LUZ DARY COLMENARES LLANES

CARLOS EDUARDO REYES SIERRA

UNIVERSIDAD INDUSTRIAL DE SANTANDER

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

BUCARAMANGA 2012

OPTIMIZACIÓN DE UN MOTOR DE INFERENCIA DIFUSA MEDIANTE LA TECNOLOGÍA CUDA (Compute Unified Device Architecture).

LUZ DARY COLMENARES LLANES

CARLOS EDUARDO REYES SIERRA

Proyecto de grado presentado como requisito parcial para optar al título de Ingeniero(a) de Sistemas

Directora

PhD (c) LOLA XIOMARA BAUTISTA ROZO

Codirector

MSc (c) EMIR ALEXÁNDER GALVIS TOBAR

UNIVERSIDAD INDUSTRIAL DE SANTANDER

ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA

FACULTAD DE INGENIERÍAS FISICOMECÁNICAS

BUCARAMANGA 2012

DEDICATORIA

A ***DIOS*** por concederme una oportunidad en esta vida.

A mi padre ***DARÍO COLMENARES GÓMEZ*** por instruirme en la perseverancia, guiarme con su carácter y por brindarme el apoyo económico necesario.

A mi madre ***AURORA LLANES PÉREZ*** por brindarme su amor incondicional y enseñarme que sólo con paciencia se alcanza una meta.

A mi hermana ***AZUCENA*** y a mis hermanos ***RUBÉN DARÍO***, ***JOSÉ ANDRÉS*** y ***CRISTIAN GABRIEL,*** mis ángeles, su amor y compañía es mi fortaleza.

A mis sobrinitos hermosos ***ANA LUCÍA***, ***MARIANA***, ***JUAN PABLO***, a quien está por llegar y especialmente a ***LAURA SOFÍA***: Mi angelito hermoso, tu familia te ama y jamás te olvidará, eres mi luz en medio de la oscuridad.

Finalmente, dedico este logro a todos y cada uno de los **AMIGOS, COMPAÑEROS Y MAESTROS** que me acompañaron en esta travesía de pregrado por la **UNIVERSIDAD INDUSTRIAL DE SANTANDER**, pues de todos sin duda alguna, aprendí algo.

Luz Dary Colmenares Llanes

Al Creador por darme esta oportunidad y guiarme en el proceso de conseguir este sueño.

A mi Madre **ANA DOLORES** y Padre **REYNALDO** quienes con sus consejos, cariño y ayuda incondicional soplaron las velas de este barco.

A **CHRISTIAN RENE** quien siempre tuvo la palabra correcta en momentos difíciles.

Carlos Eduardo Reyes Sierra

AGRADECIMIENTOS

Los autores expresan su agradecimiento:

A la profesora ***LOLA XIOMARA BAUTISTA ROZO***, directora del presente proyecto por darnos la oportunidad y el apoyo necesario para llevar a buen puerto este trabajo.

Al profesor ***EMIR ALEXÁNDER GALVIS TOBAR,*** codirector, gracias por todas sus enseñanzas y por hacer posible la entrega del presente proyecto.

Al ingeniero de sistemas ***ALVARO ANDRÉS OBREGÓN CARREÑO*** por su tiempo, colaboración y valioso aporte.

Al profesor ***CARLOS JAIME BARRIOS HERNÁNDEZ*** por sus valiosas instrucciones y aportes.

Al ***GRUPO DE INVESTIGACIÓN EN INGENIERÍA BIOMÉDICA (GIIB),*** a todos sus integrantes por acogernos y apoyarnos.

TABLA DE CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

GLOSARIO

1. **CUDA*:*** Siglas de *Compute Unified Device Architecture*, que hace referencia tanto a un compilador como a un conjunto de herramientas de desarrollo creadas por NVidia que permiten a los programadores usar una variación del lenguaje de programación C para codificar algoritmos en GPUs de NVidia.
2. **FIS*:*** *Fuzzy Inference System.* Sistema de Inferencia Difusa.
3. **GeForce:** Es la denominación que tienen las tarjetas gráficas que cuentan con unidades de procesamiento gráfico (GPU) desarrolladas por la empresa estadounidense NVIDIA.
4. **GPU*:*** Acrónimo del inglés *Graphics Processing Unit*, circuito electrónico dedicado al procesamiento de gráficos u operaciones de coma flotante. Aligera la carga de trabajo de la CPU.
5. ***NVidia Corporation*:** Empresa multinacional especializada en el desarrollo de unidades de procesamiento gráfico y tecnologías de circuitos integrados para estaciones de trabajo, ordenadores personales y dispositivos móviles.
6. **Threads*:*** En sistemas operativos, un hilo de ejecución o subproceso es una característica que permite a una aplicación realizar varias tareas a la vez (concurrentemente). Un hilo es básicamente una tarea que puede ser ejecutada en paralelo con otra tarea.

RESUMEN

OPTIMIZACIÓN DE UN MOTOR DE INFERENCIA DIFUSA MEDIANTE LA TECNOLOGÍA CUDA (Compute Unified Device Architecture)*

AUTORES: COLMENARES LLANES, Luz Dary, REYES-SIERRA, Carlos Eduardo**

PALABRAS CLAVE: Lógica Difusa, Modelado difuso, Java, Inteligencia Artificial, CUDA, Procesamiento en paralelo, Complejidad computacional.

RESUMEN: Este proyecto presenta el desarrollo de la herramienta NOICA, elaborada con el fin de crear y estudiar modelos exhaustivos de propósito general desarrollados mediante Lógica Difusa. Para llegar a esta herramienta se construyeron tres prototipos: el primero hace un acercamiento a la matemática de la Lógica Difusa y se le realizaron pruebas de caja blanca[1], el segundo prototipo es la generalización del algoritmo del Prototipo 1, en el cual se construyó un motor de inferencia difusa tipo *Mamdani* [**1**], a partir de los conceptos estudiados en el libro *From classical (crisp) sets to fuzzy sets: A grand paradigm shift* [**2**] y de los requerimientos obtenidos al estudiar algunas herramientas existentes [**3**]; también se diseñó un algoritmo recursivo para la creación semiautomática de la base de conocimiento, las pruebas de este prototipo fueron de caja negra[2], orientadas a la funcionalidad del mismo; finalmente para el Prototipo 3 se estudió la tecnología CUDA [**4**], cuyo objetivo es optimizar el tiempo de cómputo que toma un algoritmo en ejecutarse, dicha tecnología fue aplicada al motor de inferencia del Prototipo 2 haciendo uso de JCuda [**5**]; al Prototipo 3 le fueron realizadas pruebas de rendimiento con el fin de contrastar los resultados teóricos con los experimentales.

---

*Trabajo de grado.

** Facultad de ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas. Director: Lola Xiomara Bautista, Codirector: Emir Alexánder Galvis Tobar.

[1]Caja blanca: Pruebas al código con el fin de comprobar los resultados experimentales y teóricos básicos.

[2] Caja negra: Pruebas que realizan usuarios finales de la herramienta.

ABSTRACT

OPTIMIZATION OF A FUZZY INFERENCE ENGINE with CUDA technology (Compute Unified Device Architecture) *

AUTHORS: COLMENARES LLANES, Luz Dary, REYES-SIERRA, Carlos Eduardo**

KEYWORDS: Fuzzy Logic, fuzzy modeling, Java, Artificial Intelligence, CUDA, Parallel Processing, Computational Complexity.

ABSTRACT: This work presents the development of Noica tool, developed to create and study exhaustive of general purpose models developed by fuzzy logic. To get to this tool construction of three prototypes: the first a rapprochement to Math of Fuzzy Logic and to get to this tool construction of three prototypes: the first a rapprochement to Math of Fuzzy Logic doing white-box[3] testing, the second prototype is the generalization of the algorithm of the Prototype 1, where was built an fuzzy inference engine kind Mamdani [1], based on the concepts studied in the book From classical (crisp) sets to fuzzy sets: A grand paradigm shift [2] and the requirements obtained by studying some existing tools [3], also designed a recursive algorithm for semi-automatic creation of the knowledge base, this prototype testing have been black-box [4] oriented to the functionality thereof; finally for prototype 3 was studied CUDA technology [4] which aims to optimize the computational time it takes to execute an algorithm, this technology was applied to the inference engine using JCuda[5], the Prototype 3 were performed tests were conducted to compare theoretical and experimental results.

*Undergraduate work.

**Faculty of Physical Mechanical Engineering. School of Systems Engineering Director: Lola Xiomara Bautista, Codirector: Emir Alexánder Galvis Tobar.

[3] White box: Testing the code in order to check theoretical and experimental results.

[4] Black box: Testing by end users of the tool.

INTRODUCCIÓN

El presente libro describe el diseño, construcción y optimización de un motor de inferencia difusa, que produjo como resultado la herramienta NOICA, aportando al estudio de la Lógica Difusa [**2**] ya que ofrece una herramienta alternativa para la evaluación de modelos cuya cantidad de reglas sea considerable[5] y coloca al alcance del lector los conceptos fundamentales para la creación y aplicación de dicha lógica.

En la primera parte se hace una revisión y estudio de los conceptos básicos de temas como: Lógica Difusa, computación de alto rendimiento [**6**] y CUDA. En la segunda parte se desarrolla el proyecto estudiando los requerimientos del mismo, y haciendo un estudio de los componentes principales de un motor de inferencia tipo *Mamdani*, posterior a esto se desarrolla la herramienta final mediante la descripción e implementación de cada uno de los 3 prototipos para los cuales se muestra su objetivo principal y las pruebas realizadas con sus respectivos resultados y conclusiones, lo cual dio como fruto el motor de inferencia de la herramienta NOICA, optimizado mediante la tecnología CUDA.

El contenido de este libro en términos generales es el siguiente:

1. ESPECIFICACIONES DEL PROYECTO. Se presenta el proyecto, título, objetivos y la descripción del problema.

2. MARCO DE REFERENCIA. El propósito es presentar la terminología empleada en el Proyecto; para comodidad del lector, si éste desea ahondar en los términos debe dirigirse a la bibliografía.

3. METODOLOGÍA. Se aplicó la metodología en cascada. Se informa al lector cómo fue aplicada para lograr la culminación del proyecto, y cuáles fueron los lineamientos que permitieron su desarrollo.

[5] Volumen de reglas que toma gran cantidad de recursos por parte de la máquina, haciendo que el tiempo de cómputo incremente.

4. DESARROLLO DEL PROYECTO Muestra los 3 prototipos con su respectiva introducción, desarrollo, pruebas y conclusiones.

5. CONCLUSIONES Y RECOMENDACIONES. Una crítica constructiva del proyecto, permite obtener algunas conclusiones y recomendaciones globales del mismo.

6. BIBLIOGRAFÍA. La cual debe ser revisada por el lector si éste desea profundizar su conocimiento sobre términos clave para la comprensión del proyecto.

1. ESPECIFICACIONES DEL PROYECTO.

1.1 TÍTULO

Optimización de un motor de inferencia difusa mediante la tecnología CUDA (Compute Unified Device Architecture).

1.2 OBJETIVOS

1.2.1 Objetivo general

Creación y Optimización de un motor de inferencia difusa mediante el uso de GPUs[6] Nvidia y JCuda[7].

1.2.2 Objetivos Específicos

1. Estudiar las características y aplicaciones de la lógica difusa.
2. Diseñar el prototipo de un motor que evalúe sistemas de inferencia difusos.
3. Estudiar la arquitectura y el framework de trabajo de Nvidia para computación de alto rendimiento.
4. Implementar el motor, con el fin de evaluar sistemas de inferencia difusos con parámetros cuantitativos de entrada y salida, aprovechando la capacidad de cómputo paralelo de algoritmos que ofrece la tecnología CUDA.

   Desarrollando 3 prototipos, así:

   - Prototipo 1: Evaluará Sistemas de Inferencia Difusos con número limitado de entradas y de reglas. Entradas y resultados en modo consola.

[6] GPU: *Graphics Processing Unit*.
[7] JCuda: Referencia de las bibliotecas CUDA, para ser usadas en Java.

- Prototipo 2: Evaluará Sistemas de Inferencia Difusos con alto número de entradas y reglas[8], con parámetros de entrada variables en tiempo real de simulación, el cual se ejecute en computadoras de uso común.
- Prototipo 3: Implementación del prototipo 2 en CUDA.

1.3 DESCRIPCIÓN DEL PROBLEMA

Algunas herramientas científicas que existen hoy en día[**3**] para la evaluación de sistemas de inferencia difusos, se enfrentan a la barrera que imponen los tiempos de ejecución de los algoritmos con que fueron construidas, esto se aprecia cuando ejecutan modelos cuyo número de entradas es de tamaño considerable, ya que éstas han sido diseñadas para trabajar en forma secuencial y en computadores de propósito general. Un caso particular es el *Fuzzy ToolBox* de Matlab [**7**], que al operar con un gran volumen de conjuntos difusos, variación constante de los parámetros de entrada o ejecución de un número considerable de reglas, aumenta el tiempo para la obtención de resultados.

Con el fin de resolver este tipo de problemas nació la computación de alto rendimiento [**6**], ésta se encarga de estudiar el hardware empleado en la solución de problemas software que requieren consumir una cantidad de recursos, tal que, una máquina de uso convencional no proporciona, haciendo uso de tecnologías como: *clusters*, *grids*, supercomputadores y computación en paralelo. Sin embargo, a partir del 2006 con el anuncio de NVidia de la creación de CUDA y el chip G80, surgió un nuevo paradigma en la computación distribuida, en el cual, aplicaciones antes ejecutadas en *clusters* ahora pueden serlo en una computadora personal, esto supone ahorro en términos económicos y pone a disposición de cualquier persona la computación de alto rendimiento.

[8] Se considera alto número de entradas la cantidad que hace ineficiente el algoritmo secuencial.

2. MARCO DE REFERENCIA

2.1 LÓGICA DIFUSA.

Introducida por Lotfi Ali Asker Zadeh [**8**], la Lógica Difusa intenta describir o modelar lo que una persona percibe de su entorno. A diferencia de la lógica clásica, la cual es binaria, en donde un elemento pertenece o no pertenece a un determinado conjunto, la Lógica Difusa presenta características interesantes como son: la flexibilidad y la tolerancia con la imprecisión, la capacidad para moldear problemas no-lineales y su fundamento en el lenguaje natural y el sentido común [**9**] a través de reglas heurísticas que tienen la forma *IF antecedente THEN consecuente* [**1**]. Cuando se recopilan situaciones o hechos que se presentan a diario, es posible plantear modelos en dicha lógica, estos se construyen con ayuda de la teoría de conjuntos aplicada a los conceptos como: universo del discurso, conjunto difuso, rango, función de membresía, entre otros[**2**]. La sencillez al momento de construir modelos, ha llevado a la Lógica Difusa a estar presente en aplicaciones para la medicina, ingeniería electrónica, inteligencia artificial, ingeniería de control, bases de datos, entre otras.

2.2 COMPUTACIÓN DE ALTO RENDIMIENTO.

La ejecución de programas que requieren procesar grandes volúmenes de datos y por tanto realizar una cantidad considerable de operaciones entre los mismos, implica que las máquinas dónde éstos se ejecutan, presenten un óptimo desempeño. La computación de alto rendimiento [**6**], se encarga de estudiar el hardware empleado en la solución de problemas software, que requieren consumir una cantidad de recursos que una máquina de uso convencional no proporciona.

La computación de alto rendimiento usa tecnologías como: *clusters*, *grids*, supercomputadores o computación en paralelo, con lo cual se ha hecho un gran

aporte a la solución de problemas NP[9] y *NP-Hard*. Hoy día esta tecnología está al alcance de las personas del común por medio de nuevas arquitecturas en tarjetas de video como: Fermi, Tesla, G80, entre otras [**4**].

2.3 CUDA

Es un SDK elaborado por *NVidia Corporation*, el cual trabaja en base a la arquitectura de la tarjeta gráfica GPU del mismo fabricante para realizar cálculos de forma paralela, lo cual minimiza el tiempo de cómputo. Esta tecnología ha sido aplicada en dinámica de fluidos, gráficos, automatización electrónica, finanzas, procesamiento de señales entre otras áreas [**4**], esto evidencia su versatilidad en diferentes ramas de la ciencia, por lo cual ha cobrado importancia en los últimos años.

En la Figura 1 se observa el desempeño teórico de diferentes tarjetas de video versus procesadores Intel de propósito general.

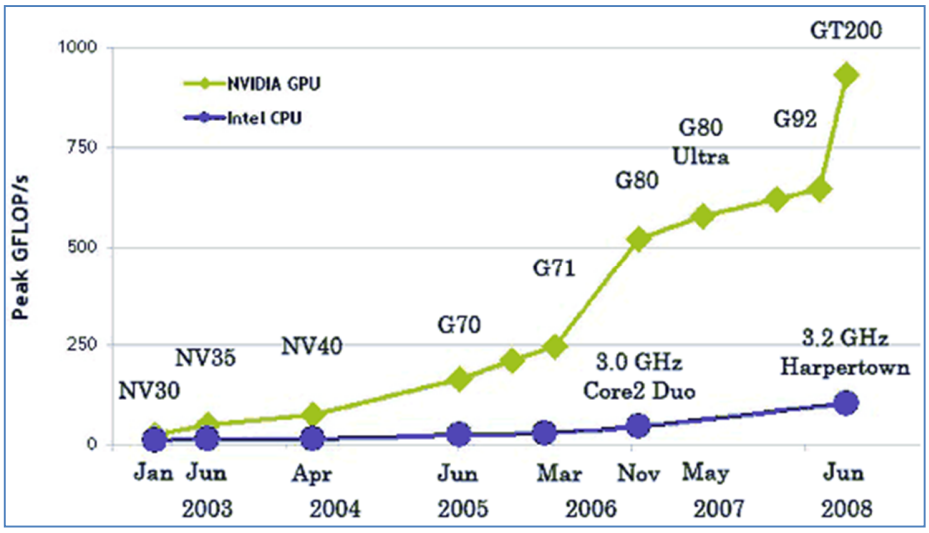


Figura 1: Desempeño teórico de diferentes tarjetas de video versus diferentes procesadores intel de propósito general.

Fuente: iXBT Labs – Computer Hardware in Detail http://ixbtlabs.com/articles3/video/cuda-1-p1.html

[9] NP: No Polinomial.

En la Figura 2 se aprecia la arquitectura de una CPU y una GPU, el poder de la GPU reside en el gran número de ALUs (*Arithmetic Logic Unit*) que posee. Esta arquitectura nació con la GeForce 8800 GTX de la corporación NVidia en el año 2006 y hoy en día continúa implementándose en las tarjetas de video de esta corporación.

Los lenguajes de programación que pueden ser utilizados con esta tecnología, hoy en día son: C, C++, Java, Fortran, OpenCl y Phyton.

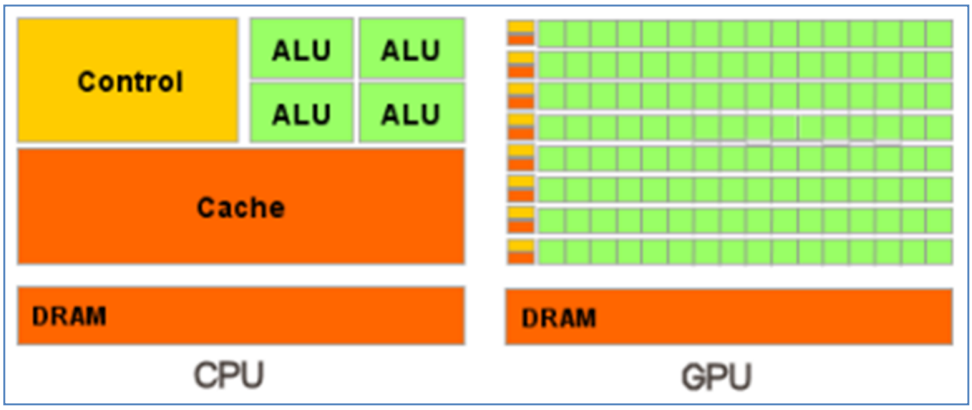


Figura 2: Arquitectura de una CPU y una GPU.

Fuente: iXBT Labs – Computer hardware in detail http://ixbtlabs.com/articles3/video/cuda-1-p1.html

3. METODOLOGÍA

Cada uno de los prototipos fue desarrollado conforme las fases generales para realizar un software: Análisis, Diseño, Implementación, Pruebas y Mantenimiento. En cada hito los diferentes prototipos tienen definidos al inicio del mismo específicamente el alcance, los requerimientos y las limitaciones de manera rigurosa, de esta forma se observa que no existen cambios en ninguna de las etapas del ciclo de construcción de la herramienta, al observar esto se escogió la metodología de desarrollo en cascada[**10**] ya que ésta, aunque permite iteraciones entre una y otra etapa, requiere definir al principio del proyecto los requerimientos a cabalidad, lo cual implica ser riguroso desde el comienzo del mismo, al cubrir por completo las especificaciones de la herramienta en lo que a teoría se refiere.

4. DESARROLLO

4.1 DEFINICIÓN DE REQUERIMIENTOS

Se estudiaron algunas de las herramientas difundidas a nivel internacional [**3**], con base en estas se realizó la ingeniería de software que corresponde al desarrollo de cada prototipo. (*Ver anexo*).

4.2 DISEÑO DEL SISTEMA Y DEL SOFTWARE:

Con base en la definición de requerimientos se implementó un Sistema tipo *Mamdani*, ya que este es el común denominador en las herramientas estudiadas, en la Figura 3 se ilustran los componentes de un sistema de este tipo.

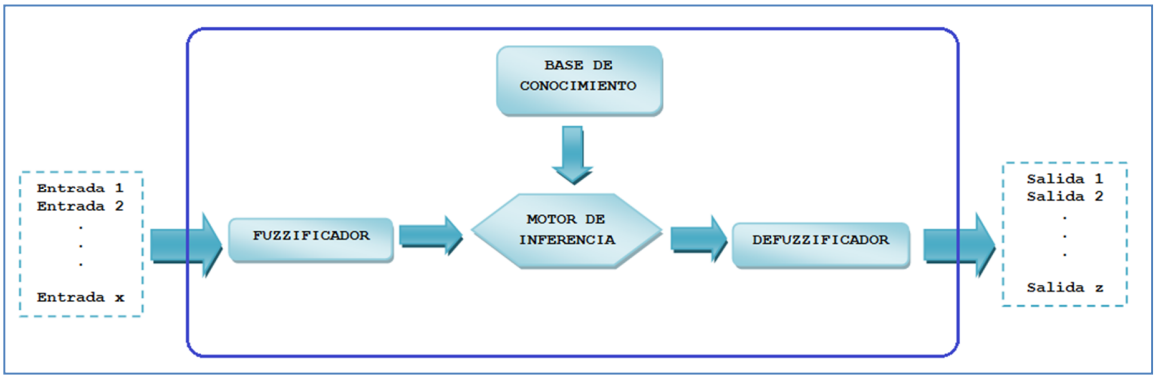


Figura 3: Sistema Tipo Mamdani

Fuente: Autores, basado en: ***From classical (crisp) sets to fuzzy sets: A grand paradigm shift.***

El *fuzificador* usado en la herramienta es el tipo *singleton*, al cual el usuario debe entregar los respectivos parámetros para evaluar cada variable de entrada, y el *defuzificador* empleado para el motor de inferencia es *centroide*.

4.3 PROTOTIPO 1: FUNCIONAMIENTO BÁSICO DE LA LÓGICA DIFUSA.

Este prototipo se desarrolló con la idea básica de hacer un acercamiento a la matemática empleada por la Lógica Difusa, es limitado ya que solo permite un

conjunto difuso de entrada y un conjunto difuso de salida; el motor de inferencia difusa es *Mamdani*, fue desarrollado con lenguaje C++, ya que, se trata de un lenguaje familiar, orientado a objetos, y se presenta como el lenguaje nativo para implementar CUDA.

La Figura 4, muestra cómo funciona un motor de inferencia *Mamdani* de forma básica, con base en la lógica que éste exhibe se realizó la implementación del Prototipo 1, para tener un acercamiento inicial a un algoritmo para el desarrollo de los modelos.

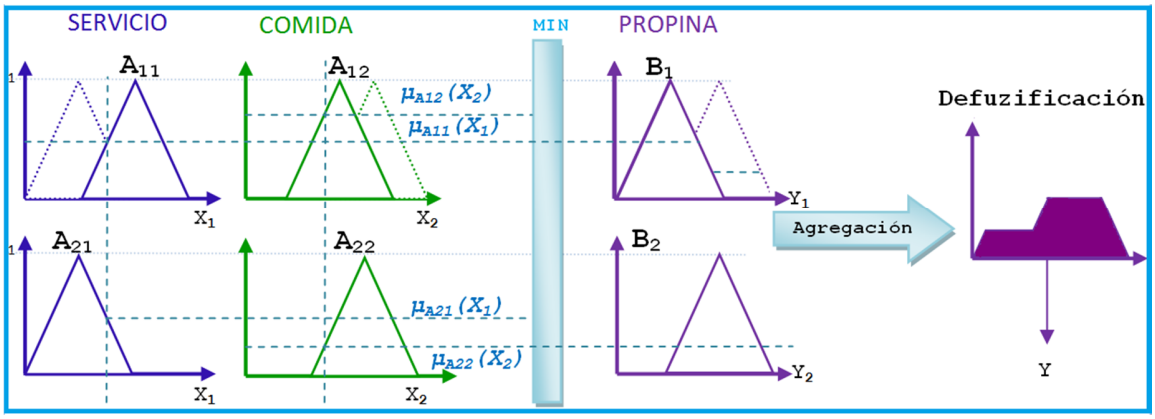


Figura 4: Ejemplo implementado en el Prototipo 1. Fuente: Autores.

4.3.1 Prototipo 1. Pruebas

Las pruebas se hicieron comparando los resultados del modelo que consta de: una entrada, una salida y 4 subconjuntos para la entrada, con diferentes funciones de membresía, entre el Fuzzy Tool Box de Matlab y el Prototipo 1. Los resultados se observan en las tablas 1 y 2.

La tabla 1 muestra los parámetros tenidos en cuenta para realizar un ejemplo en lógica difusa con el cual se busca medir la eficacia mediante el porcentaje de error del prototipo 1 respecto al Fuzzy Tool Box de Matlab, la tabla 2 muestra los resultados para diferentes ejemplos realizados en las 2 herramientas, esta tabla es un resumen de la tabla 1.

Tabla 1: Resultados de las pruebas experimentales entre el fuzzy tool box de Matlab y el Prototipo 1.

Herramienta	Ejemplo 1					And	Or	Implicación	Agregación	Resultados	Porcentaje de error
	Entradas	Subconjuntos	Salidas	Subconjuntos	Reglas					Prueba 1	
F.T. B. Matlab	1	4	1	2	2	Prod.		Min.	Máx.	21,9	0,1369863
Prot. 1	1	4	1	2	2	Prod.		Min.	Máx.	21,87	

Fuente: Autores

Tabla 2: Resultados de las pruebas experimentales entre el fuzzy tool box de Matlab y el Prototipo 1, para 10 ejemplos más.

Ejemplo	MATLAB	PROTOTIPO 1	PORCENTAJE DE ERROR
2	21,3	21,23	0,328638498
3	25,9	25,4	1,930501931
4	32,8	31,9	2,743902439
5	21,5	21	2,325581395
6	40,43	40	1,063566658
7	21,1	21,05	0,236966825
8	50,3	49	2,584493042
9	33,7	32,69	2,997032641
10	22,3	22	1,34529148
11	47,9	47,59	0,647181628

Fuente: Autores

4.3.2 Conclusiones del Prototipo 1:

1) Se realizaron pruebas de portabilidad en diferentes sistemas operativos: Fedora 11, Windows XP *professional* y Windows 7 Home *Edition*, éstas

hicieron considerar un lenguaje de programación que fuera soportado por diferentes sistemas operativos y configuraciones de hardware.

2) Para llegar al Prototipo 2 programando en lenguaje C++, era necesario trabajar con un entorno de desarrollo que aporte una GUI (*Graphics user interface*), en la cual se pueda implementar una interfaz gráfica que permita la usabilidad por parte del usuario final.
3) Para que el Prototipo 1 se ejecute de forma adecuada, se debe compilar en la maquina huésped, lo que implica diferentes compiladores para diferentes sistemas operativos.
4) Los valores obtenidos a partir de las pruebas realizadas se compararon con el Fuzzy tool box de Matlab y éstos no superan el 4% de error, lo cual indica que el planteamiento del algoritmo es correcto, dando continuidad al proyecto.
5) El algoritmo de la Figura 4 debe ser generalizado para la creación de modelos.

4.4 PROTOTIPO 2: FUNCIONALIDAD.

Con base en las Conclusiones del Prototipo 1, se indagaron los lenguajes que soportan la tecnología CUDA, y producto de esta investigación se desarrolló la Tabla 3:

Tabla 3: Lenguajes de programación que soportan CUDA.

Lenguaje →	**C++**	**C**	**Java**	**Fortran**	**OpenCl**	**Python**
Compiladores ↓	Borland C++ Builder y Microsoft Visual Studio C++,Qt, fltk, Kdevelop	Borland C++ Builder y Microsoft Visual Studio C++,Qt, fltk,	Un único compilador, siempre igual JDK	Varios compiladores	Un solo compilador	Varios compiladores

Gestión de memoria	A cargo del programador	A cargo del programador	Automática	Por el programador	Por el programador	Automática
Orientación	Orientado a objetos	Orientado a estructuras	Orientado a objetos			
Liberación de memoria	A cargo del programador	A cargo del programador	Automática			
% uso a nivel mundial	9.072	17.707	17.913	0.381	-	3.944

Fuente: Autores

De la Tabla 3, surgieron nuevos requerimientos para la implementación del lenguaje para el desarrollado el Prototipo 2, el cual debe cumplir los siguientes requerimientos:

1. Que sea orientado a objetos.
2. Que brinde la oportunidad de crear una interfaz gráfica para el usuario final.
3. Que fuese de alta difusión.
4. Que pudiese ser portable en los diferentes sistemas operativos con diferentes configuraciones de hardware.
5. Al evaluar el primer requerimiento se pudieron descartar los lenguajes: fortran y C. Para cumplir con el requerimiento 2 se descartaron los lenguajes: *python*, *opencl* y C++. En este punto se evidencia que el único lenguaje disponible para evaluar es Java, por lo cual se continuó haciendo una investigación exhaustiva, procurando que cumpla a cabalidad los requerimientos faltantes.

El lenguaje Java cuenta con un intérprete universal para ejecutarse en los diferentes sistemas operativos, llamado JRE (*Java Runtime*), lo que hace posible que una implementación en este lenguaje, sea independiente de la maquina huésped. Java posee de forma nativa librerías para desarrollar y proveer un

entorno gráfico en el momento de implementar una GUI (*Graphical user interface*).

A partir de esto se llegó a la conclusión que Java, es el mejor calificado de acuerdo a los requerimientos anteriores y da continuidad al proyecto, fue elegido ya que cumple las consideraciones anteriores, además de:

- Tener un solo compilador JDK (*Java Development Kit*).
- El manejo de memoria se hace automático protegiendo la estabilidad del sistema operativo en la administración.

4.4.1 Algoritmo del motor de inferencia tipo Mamdani.

Continuando con el desarrollo del Prototipo 2, se generalizó y desarrolló un algoritmo (Figura 5) para el motor de inferencia tipo *Mamdani*. Este fue creado a partir de los conceptos estudiados en el libro *From classical (crisp) sets to fuzzy sets: A grand paradigm shift* [**2**].

El algoritmo de la Figura 5, funciona así: en el primer ciclo *FOR* se procesa cada regla dentro de la base de conocimiento, (*j* indica cuál regla se está procesando y *m* indica el número total de reglas), y en el segundo ciclo *FOR*, mediante el método Eval_func_mem_regla(antecedente[*i*]) se procede a evaluar cada antecedente de la misma (*i* indica el antecedente que se evalúa y *n* el total de antecedentes de dicha regla), posterior a esto se determina el valor de verdad de los conectores en la regla (*and* u *or*) del antecedente para obtener su grado de pertenencia en el consecuente.

Para las reglas tipo *and* se usa el método *EVALUAR_IMPLICACION* con alguno de los siguientes criterios:

- Mínimo: $X_1 \cap X_2 \Rightarrow \mu_{X_1 \cap X_2}(u) = min\left(\mu_{X_1}(U), \mu_{x_2}(U)\right)$ (1)
- Producto: $X_1.X_2 \Rightarrow \mu_{X_1.X_2}(u) = \mu_{X_1}(U).\mu_{X_2}(U)$ (2)

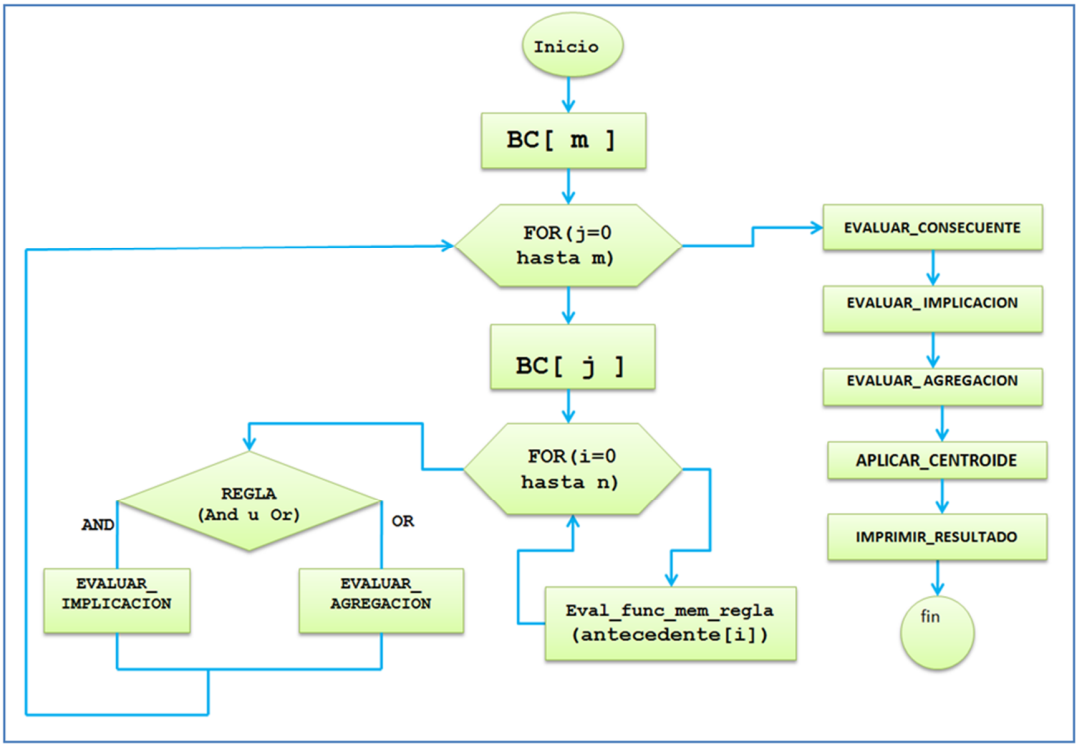


Figura 5: Algoritmo del motor de inferencia tipo Mamdani. Fuente: Autores.

- Así mismo, para las reglas tipo *or* se usa el método *EVALUAR_AGREGACION* con alguno de los siguientes criterios:

- Máximo: $X_1 \cup X_2 \Rightarrow \mu_{X_1 \cup X_2}(u) = max\left(\mu_{X_1}(U), \mu_{X_2}(U)\right)$ (3)

- Probor: $X_1, X_2 \Rightarrow \mu_{X_1, X_2}(u) = (\mu_{X_1}(U) + \mu_{X_2}(U) - \mu_{X_1}(U).\mu_{X_2}(U)$ (4)

Siendo u el parámetro de entrada con el que el usuario evalúa el modelo, y $\mu_{X_x}(U)$ el resultado del mismo.

Al finalizar la evaluación de la base de reglas, se obtiene un arreglo unidimensional con valores numéricos que determinan el grado de pertenencia al conjunto difuso que hace referencia en el consecuente (en el método EVALUAR_CONSECUENTE), luego de esto, el flujo continúa con el cálculo de la implicación (en el método EVALUAR_IMPLICACION), el cual ofrece:

- Mínimo: $Y_1 \cap Y_2 \Rightarrow \mu_{Y_1 \cap Y_2} = min\left(\mu_{y_1}, \mu_{y_2}\right)$ (5)
- Producto: $Y_1 . Y_2 \Rightarrow \mu_{Y_1 . Y_2} = \mu_{Y_1} . \mu_{Y_2}$ (6)

Luego se calcula la agregación (en el método EVALUAR_AGREGACION), la cual ofrece:

- Máximo: $Y_1 \cup Y_2 \Rightarrow \mu_{Y_1 \cup Y_2} = max\left(\mu_{Y_1}, \mu_{Y_2}\right)$ (7)
- Probor: $Y_1, Y_2 \Rightarrow \mu_{Y_1, Y_2} = \left(\mu_{Y_1} + \mu_{Y_2} - \mu_{Y_1} . \mu_{Y_2}\right)$ (8)
- Suma: $Y_1, Y_2 \Rightarrow \mu_{Y_1, Y_2} = \left(\mu_{Y_1} + \mu_{Y_2}\right)$ (9)

Finalmente se realiza el proceso de defuzificación mediante el método del *centroide*.

Para construir la base de reglas de forma semiatendida se desarrolló un algoritmo (Figura 6), que de forma recursiva, hace las diferentes combinaciones entre todas las entradas y salidas, así construye el total de la base de conocimientos de los modelos planteados.

4.4.2 Algoritmo para generar la base de reglas.

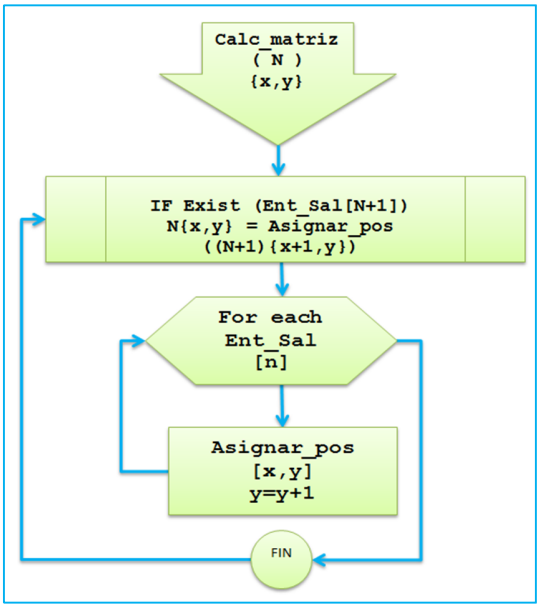


Figura 6: Algoritmo generador de la base de reglas. Fuente: Autores.

El algoritmo descrito en la Figura 6, muestra cómo generar la base de reglas de forma semiatendida con base en los subconjuntos difusos previamente descritos por el modelo diseñado por el usuario para solucionar un problema, se debe tener en cuenta que estos conjuntos se representan de acuerdo a una función de pertenencia que puede ser básicamente de 3 tipos: gaussiana, trapezoidal y triangular [**2**], éste algoritmo se implementó en la base de conocimiento, éste realiza la combinación de los antecedentes y consecuentes para el modelo que se plantee, de esta forma genera automáticamente la totalidad de la base de reglas. Inicia calculando y reservando el espacio de una matriz mediante el método Calc_matriz (N). N es un arreglo en el cual están almacenadas las entradas y salidas del modelo. Dentro de cada elemento de ese arreglo N se almacena otro arreglo que posee los diferentes subconjuntos para cada elemento. Calc_matriz(N) almacenará la respectiva combinación de los antecedentes y consecuentes teniendo en cuenta cada entrada y salida del modelo, lo cual dará paso a la formación de cada regla. El desarrollo de este algoritmo se basó en los conceptos planteados en el libro *From classical (crisp) sets to fuzzy sets: A grand paradigm shift* [**2**].

4.4.3 Diagrama de clases del Prototipo 2

Para implementar los algoritmos mencionados en la herramienta se diseñó el diagrama de clases, Figura 7, donde aparecen las clases significativas que tiene la herramienta. Estas son: **Motor**, que es el núcleo de la herramienta, tiene implementado el algoritmo descrito en la Figura 5. **Regla**, en la cual se construye la base de conocimiento a partir de las entradas y salidas del modelo, tiene implementado el algoritmo de la Figura 6. **Conjunto**, el cual es el universo del discurso para cada entrada y salida, se construye a partir de los subconjuntos que da el usuario para cada entrada y salida, y las coordenadas en las cuales se definen los mismos. La clase Main es la encargada de unificar todas las clases descritas para ejecutar el modelo.

Para evitar la redundancia de datos, el motor es un objeto que puede moverse entre las diferentes clases cuando requiera datos de las mismas.

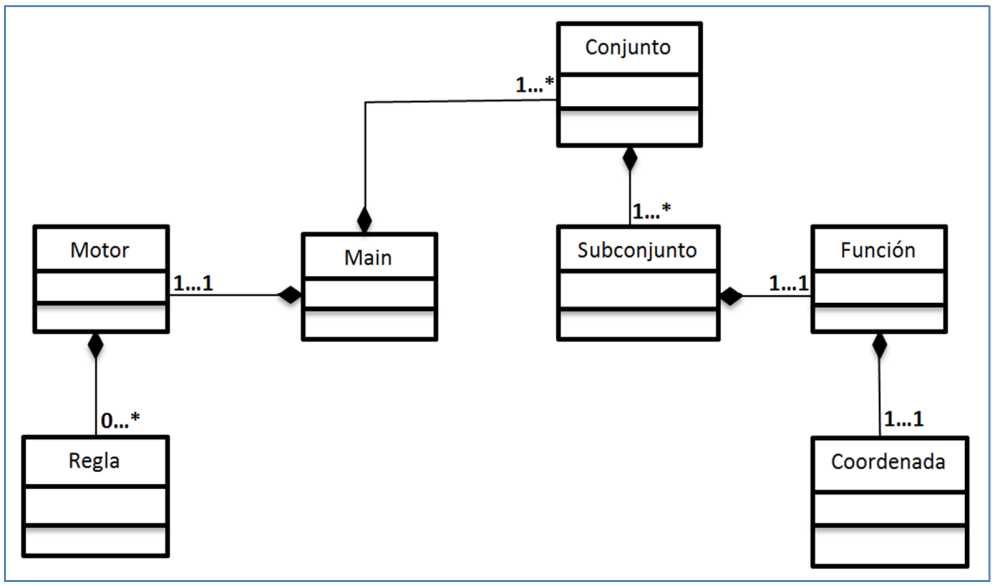


Figura 7: Diagrama de clases del Prototipo 2.

Fuente: Autores:

4.4.4 Funcionamiento del Prototipo 2.

La construcción de modelos, se hace a través de un archivo de texto plano, como se muestra en Figura 9, donde las entradas y salidas inician con el *token* “>”, los subconjuntos de las mismas se ingresan con el símbolo “>>”, para separar las entradas de las salidas se emplea el *token* “<”.

En la interfaz principal (Figura 10), el usuario interactúa directamente con los modelos introducidos; allí puede crear y configurar la base de reglas, editar los parámetros del motor de inferencia, y de forma interactiva las entradas y salidas del modelo. Cuando se da *click* sobre alguno de los elementos señalados en la Figura 10, se desplegarán las respectivas ventanas de configuración para el modelo que se haya planteado.

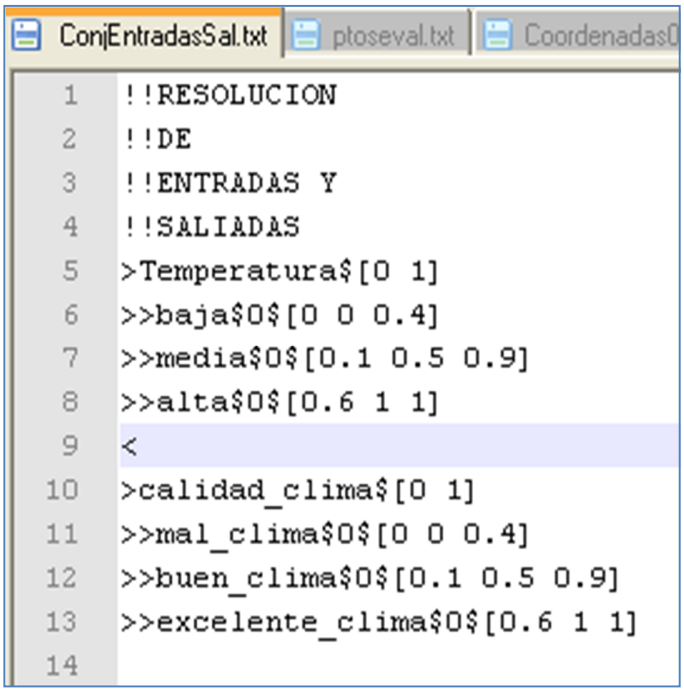


Figura 8: Inserción de entradas y salidas para la construcción de modelos.

Fuente: Autores

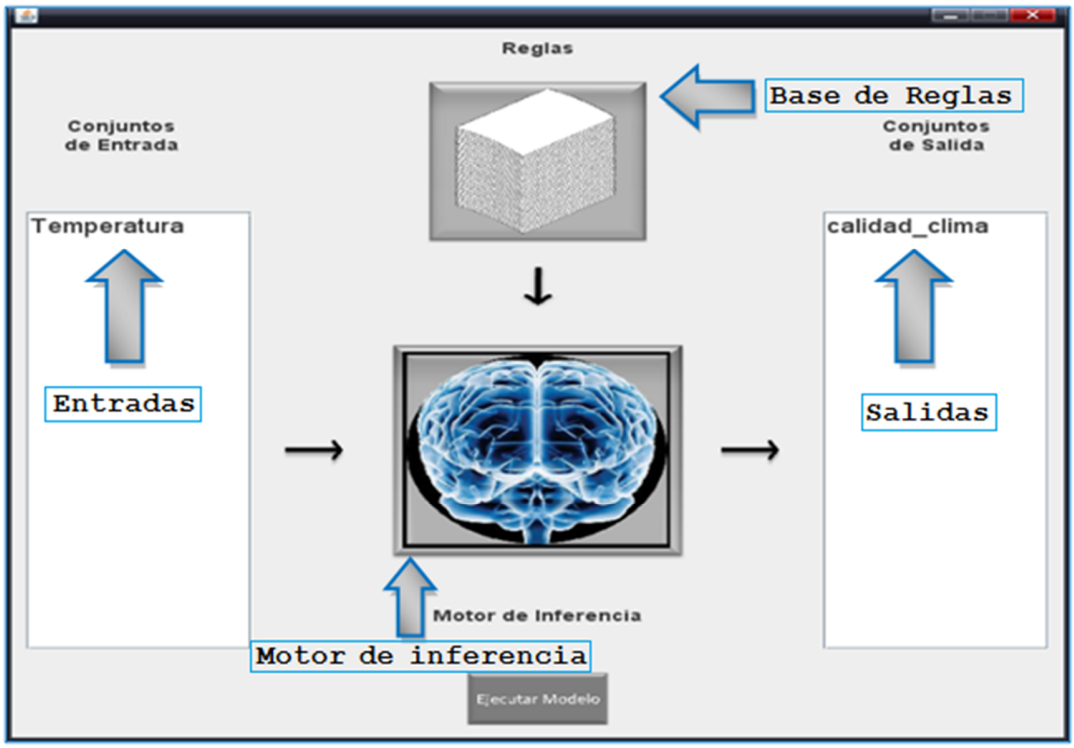


Figura 9: Interfaz principal, Prototipo 2.

Fuente: Autores

En la ventana de configuración de la base de reglas (Figura 11), el usuario puede crear y editar la base de conocimiento, solo necesita dar *click* en el botón Calcular para generar la totalidad de reglas del modelo, y si lo desea puede deshabilitar una regla haciendo *click* sobre ella, esto hará que sombree en color rojo; al dar doble *click* sobre una regla, puede cambiar el tipo de conexión *and* u *or*, de igual forma tiene la posibilidad de convertirlas todas en tipo *and* u *or*, con los botones (todas *or*, todas *and*) lo cual depende de su criterio. Al terminar creación y edición procede a guardarlas con el botón Guardar regla.

Cuando el usuario accede al motor de inferencia, tiene la interfaz de la Figura 11; allí puede definir los parámetros de decisión del motor, para las reglas a*nd u or*, así como la implicación y la agregación del mismo.

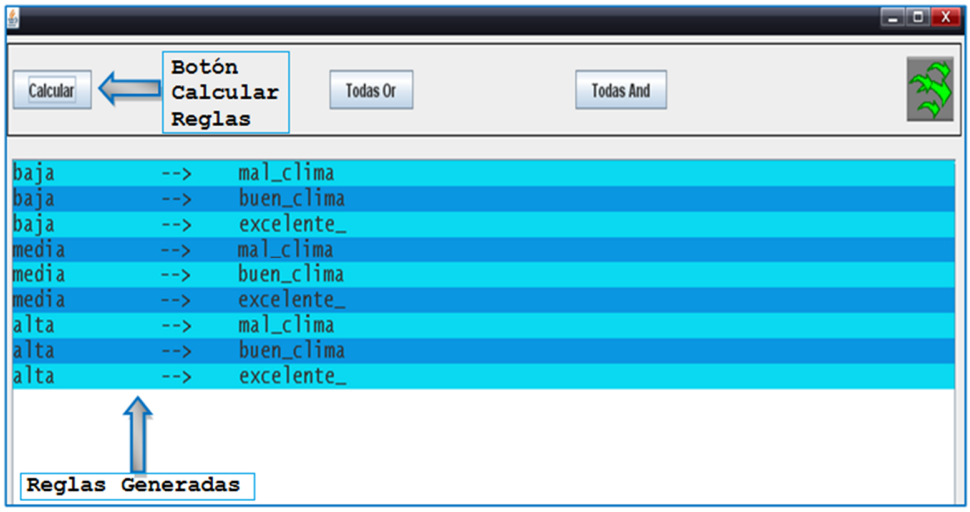


Figura 10: Creación y Edición de Reglas. Fuente: Autores

Finalmente para ejecutar el modelo, el usuario debe ingresar valores para cada una de las entradas definidas, lo cual hace mediante un archivo de texto llamado *ptoseval* (Figura 12), en el cual debe escribir los valores para cada una de las entradas separadas con el símbolo ***$***. Luego de esto en la interfaz principal se da *clic* al botón Ejecutar modelo y se obtiene la respuesta correspondiente.

Figura 11: Configuracion del motor de inferencia. Fuente: Autores

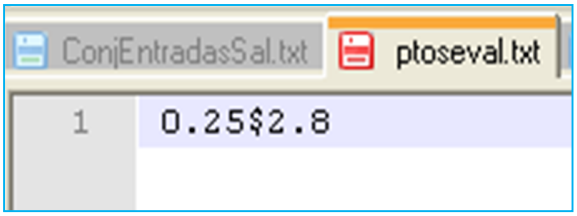


Figura 12: Insercion de valores para las entradas. Fuente: Autores.

4.4.5 Prototipo 2. Pruebas.

Para evaluar la herramienta, se diseñó una prueba de caja negra [**11**] en la cual los usuarios ingresaban el modelo descrito en la Tesis de Maestría: *Detección de enfermedades de la retina que propenden a la ceguera*[**12**], que trata el problema del glaucoma, el cual consta de 17 entradas y una salida. Para las pruebas realizadas se tomaron 5 entradas y la única salida. Los participantes de la prueba ingresaron el modelo en el "Fuzzy logic tool box" de Matlab R2008b[**7**] y luego ingresaban el mismo modelo en la herramienta propuesta *NOICA*. Para esto se les facilitó una guía de cada herramienta, la cual describía cómo ingresar modelos en cada una. Posteriormente valoraban las dos herramientas mediante una encuesta, en la cual ponderaban de 1 (baja satisfacción) a 10 (alta satisfacción),

en 5 preguntas las cuales tuvieron en cuenta los siguientes criterios: rapidez y agilidad en la creación de modelos, utilidad y rapidez que tiene la automatización de reglas, intuitivita y efectividad de la herramienta.

Esta prueba tuvo en cuenta la norma ISO/IEC9126 [**13**], la cual muestrea aspectos generales a tener en cuenta en el momento de evaluar un software, algunos de estos son: funcionalidad, confiabilidad, eficiencia, facilidad de uso, mantenimiento y portabilidad.

El requisito que debían cumplir los usuarios para realizar la prueba era tener conocimientos básicos en Lógica Difusa, tales como plantear un modelo sencillo a partir de eventos que se presentan a diario, mediante conjuntos y subconjuntos y describir las posibles reglas relevantes en dicho modelo. Teniendo en cuenta este antecedente se diseñó la prueba, esta se hizo con 22 estudiantes de pregrado y postgrado de la Universidad Industrial de Santander (UIS).

La Figura 14 muestra los resultados obtenidos. Cada columna expresa el promedio obtenido en la prueba realizada por cada pregunta, junto con el criterio aplicado a las dos herramientas. Producto de esta se aprecian las siguientes conclusiones:

- En las dos barras que representan la pregunta 1, se observa que existe una calificación de 7.72 y 7.40, resultado que las reconoce como rápidas y agiles en la creación de modelos.

- En las dos barras que representan la pregunta 2, cuyo promedio de calificación es 8.34 y 7.36, los usuarios valoran de forma positiva la rapidez que brinda cada herramienta para seleccionar y editar las reglas.

- En las dos barras que representan la pregunta 3, los usuarios aprecian de forma significativa la utilidad que brinda cada herramienta en la selección y edición de reglas arrojando los resultados de 8.38 y 7.29.

- En las dos barras que representan la pregunta 4, los usuarios consideran que el manejo de cada herramienta si es intuitivo, obteniendo resultados de 8.18 y 7.5.

- En las dos barras que representan la pregunta 5, los usuarios valoran la efectividad de cada herramienta al observar los resultados obtenidos con una calificación de 8.59 y 8.63.

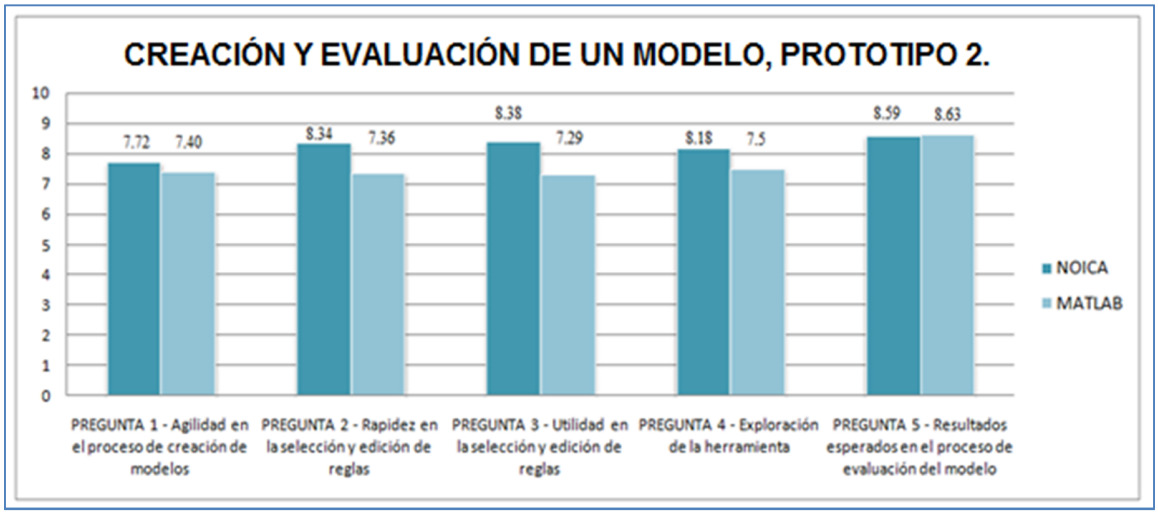


Figura 13: Pruebas del Prototipo 2. Fuente: Autores.

4.4.6 Conclusiones del Prototipo 2.

1) La herramienta propuesta *NOICA,* facilita la creación y edición de modelos de propósito general. Al ser transparente el diseño y los algoritmos empleados en la herramienta, los diferentes usuarios, pueden reforzar y adquirir conocimientos sobre la forma como trabaja la Lógica Difusa.

2) Al generarse el total de las reglas de forma semiatendida, los usuarios tienen la opción de hacer pruebas exhaustivas sobre los modelos que planteen, habilitando o deshabilitando reglas, lo cual depura y clarifica que reglas son más importantes para sus modelos.

3) A partir de las pruebas realizadas, se puede observar que la herramienta *NOICA* es competitiva frente a una de las herramientas más populares del mercado (Fuzzy Tool Box, MatLab), en cuanto al modelado de Lógica Difusa de propósito general se refiere.

4) Cuando el número de reglas crece la herramienta tarda más tiempo en la ejecución y la entrega de resultados del modelo planteado.
5) El modelo de datos implementado presenta estabilidad al no tener redundancia(Los datos son leídos por todos y tienen núcleo común).
6) El JRE (*Java runtime enviroment*) garantiza que la aplicación desarrollada con la tecnología Java pudiese ser ejecutada en cualquier sistema operativo. Al ejecutar el Prototipo 2 en los Sistemas Operativos: Fedora 11, Windows 7/64, Windows XP/32, se probó la portabilidad de software.
7) El Prototipo 2 es presentado en el artículo: *NOICA: Herramienta para la creación y estudio de modelos exhaustivos desarrollados en lógica difusa.* Expuesto en el Congreso Internacional de Investigación de Nuevas Tecnologías Informáticas, CIINTI 2012 [**14**]

4.5 PROTOTIPO 3

El objetivo principal de este prototipo es optimizar el rendimiento en la entrega de resultados del Prototipo 2, para esto se estudió la tecnología CUDA.

4.5.1 Optimización del motor de inferencia tipo Mamdani.

El motor de inferencia, es la parte de la herramienta que tiene como función procesar modelos y emular la forma de razonamiento del ser humano, tomando la base de conocimiento creada por el usuario, que es dada en reglas heurísticas y están compuestas de antecedentes y consecuentes, de tal manera que el motor pueda procesarlas de forma matemática, entregando una salida cuantitativa, siendo entonces la parte de la herramienta que se debe optimizar.

El primer paso para optimizar el motor de inferencia del Prototipo 2, es el estudio de la complejidad computacional del mismo y de esta manera detectar dónde aplicar una optimización; dicho análisis arroja como resultado que la sección que presenta mayor complejidad dentro del algoritmo es al evaluación de la base de reglas (ver Figura 14).

Por otro lado se realizó un estudio del estado del arte sobre este problema con el objetivo de plantear la metodología, de este estudio se encontraron dos artículos, el primero: *Speedup of fuzzy logic through stream processing on graphics processing units* [**15**], describe la forma de organizar las reglas, las cuales pasan de una matriz de antecedentes y consecuentes a un solo vector donde están concatenadas para ser procesadas, posteriormente se almacena dicho vector en la memoria de textura de la GPU,en este *paper* se utiliza *GPGPU*[10] mediante *OpenGL*[11] y *Cg*[12] para su implementación; el segundo *paper: Parallelisation of fuzzy inference on a graphics processor unit using the Compute Unified Device Architecture* [**16**]*,* toma la idea del primero para ser implementada en CUDA, teniendo en cuenta la limitación de la memoria global y la memoria local de la tarjeta de video, realiza pruebas usando dos tipos de funciones de membresía: trapezoidal y gaussiana, usa la tarjeta de video NVidia 8800 BFG GTX para desarrollar las pruebas .

En la Figura 15, se muestra un ejemplo para estudiar la complejidad computacional cuando se crean modelos en Lógica Difusa de propósito general, descrito así: $A = [A_1, A_2, \dots, A_n] \subseteq X$ son subconjuntos contiguos de cada variable lingüística de entrada *X*, y $B = [B_1, B_2, \dots, B_n] \subseteq Y$ son subconjuntos contiguos de cada variable lingüística de salida *Y*, el modelo consta de: $X = [X_1, X_2, \dots, X_p]$ variables lingüísticas de entrada con $A = [A_1, A_2, \dots, A_q]$ subconjuntos por cada *X* de entrada, y $Y = [Y_1, Y_2, \dots, Y_r]$ variables lingüísticas de salida con $B =$

[10] GPGPU: Aprovechamiento de las capacidades de cómputo de una GPU.
[11] OpenGL: Open graphics library.
[12] Cg: C for graphics.

$[B_1, B_2, \ldots, B_s]$ subconjuntos por cada *Y* de salida, se definen el número total de reglas posibles (*R*) como la combinación de los subconjuntos de las *X* entradas con los subconjuntos de las *Y* salidas. En el caso particular, cuando todas las variables lingüísticas de entrada (*X*) y salida (*Y*) tienen el mismo número de subconjuntos, la expresión para calcular el número total de reglas R se puede sintetizar así:

$$q^p . s^r$$

(1)

En la Figura 15, en donde el eje "y" es el número de reglas y el eje "x" las entradas procesadas.

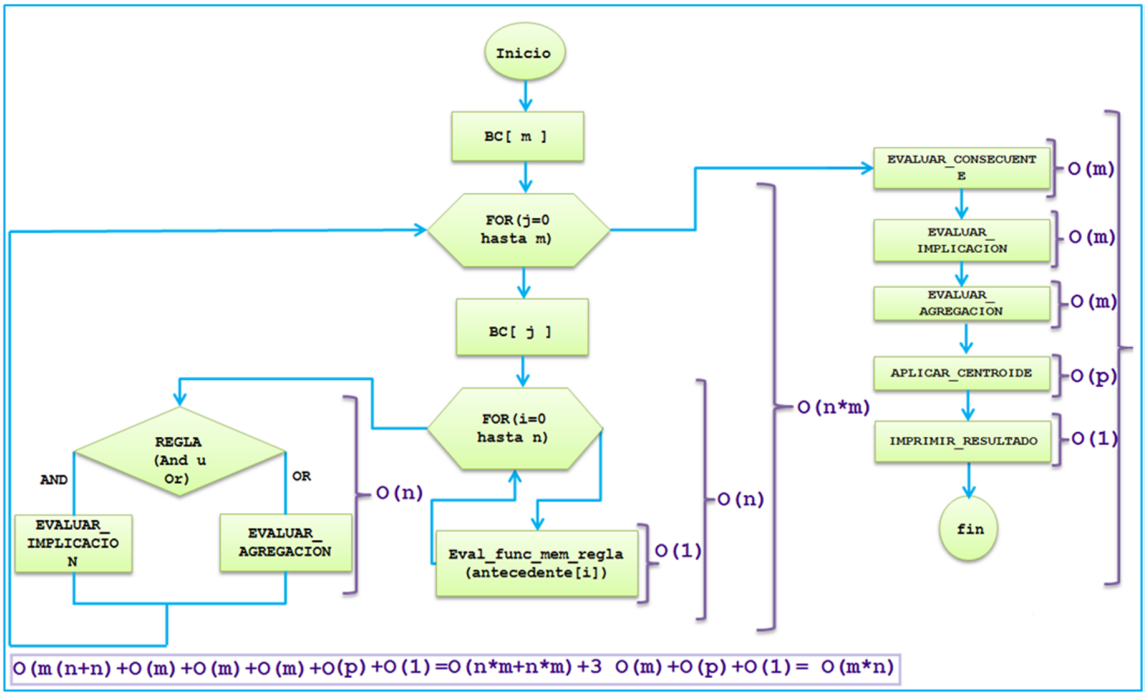


Figura 14: Complejidad computacional del algoritmo del Motor de Inferencia. Fuente: Autores.

4.5.2 Metodología para optimizar el Prototipo 2.

Del estudio de la complejidad computacional del Prototipo 2 y de la revisión de los dos artículos se plantea la siguiente metodología para la optimización del motor de inferencia del Prototipo 2:

1. Tomar la base de reglas y transformarla de una matriz de reglas a un vector único, donde éstas queden concatenadas.
2. Estudiar la forma de enviar dicho vector a la memoria global de la GPU para que sean procesadas.
3. Estudiar la forma como la GPU procesaría la información usando el algoritmo del Prototipo 2, adaptado con CUDA
4. Revisar los resultados teóricos que se desean obtener.

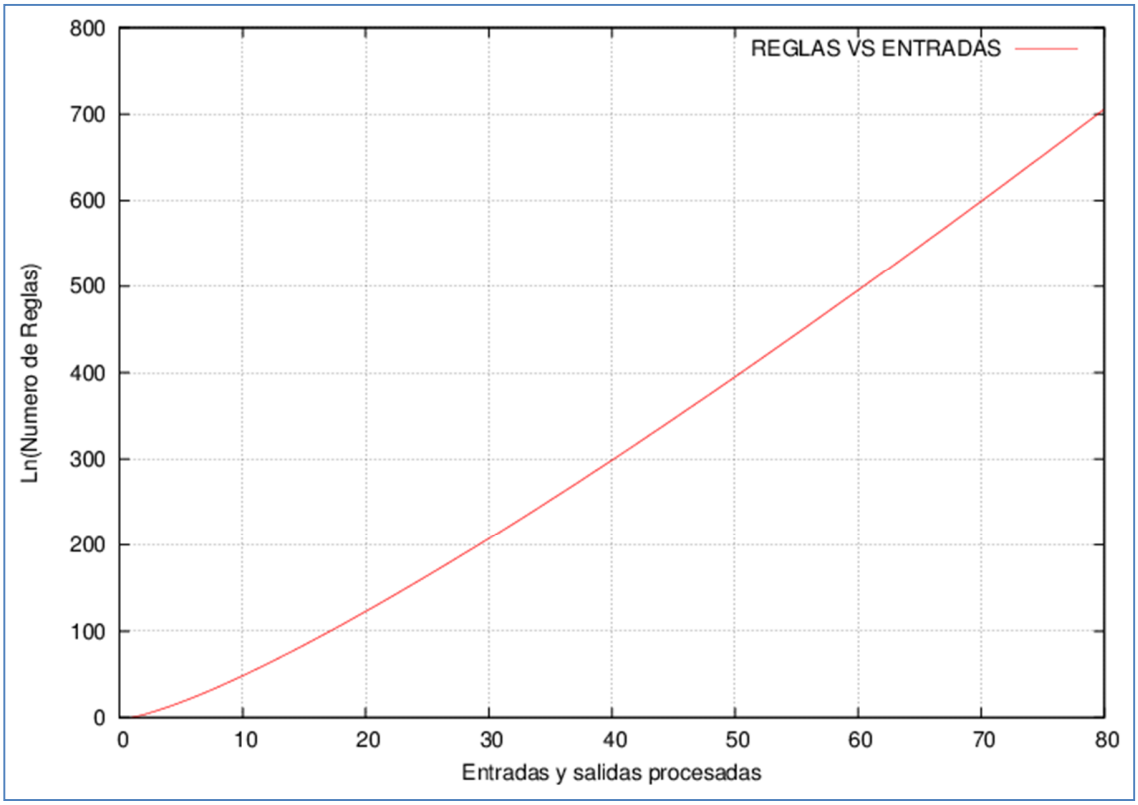


Figura 15: Reglas vs entradas y salidas procesadas Fuente: Autores

Ejecución:

1. Para transformar la base reglas de una matriz a un vector se debe tener en cuenta la información que tiene cada antecedente de una regla, esto es: la entrada a la cual esta asociada, el tipo de subconjunto que tiene cada uno, los límites inferiores y superiores de las mismas, así como los puntos que defininen la función de membresía que se aplica y los valores que el usuario desea evaluar para cada entrada. De esta manera surgen 7 vectores que poseen la información total de la base de reglas.

En el primer vector se tienen las entradas como un arreglo, Figura 16 parte (a), en el cual $\mu_{A'2}(X_N)$ representa cada subconjunto que contiene la misma. En la Figura 16 parte (b), se muestra como quedan transformadas las entradas en el arreglo para ser procesado.

En el segundo vector, las reglas pasan de una matriz de m*n, Figura 17 parte (a) (donde m es el numero de reglas, n es el número de antecedentes de cada una y $\mu_{A2}(X_N)$ indica el subconjunto al que pertenece cada entrada), a un vector como se muestra en la Figura 17 parte (b), donde "pos_con" indica la posición del subconjunto dentro del conjunto global de cada entrada.

En el tercer y cuarto vector, Figura 18 (a) y Figura 18 (b) respectivamente, "Li_N" indica el límite inferior y "Ls_N" el límite superior, del subconjunto que se procesa.

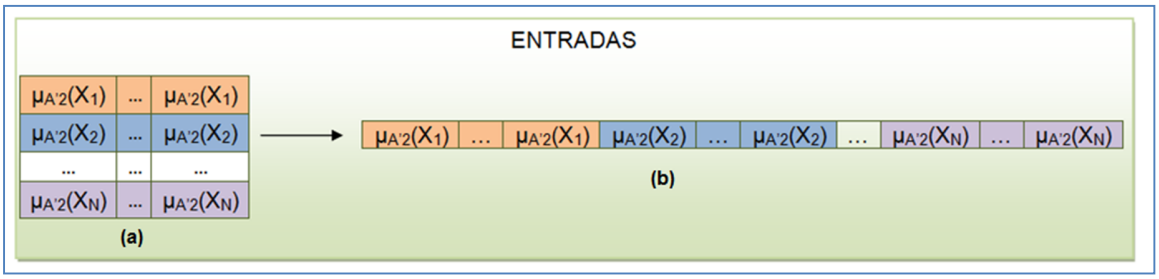


Figura 16: Transformación del arreglo de entradas(a) en un vector de entradas (b). Fuente: Autores.

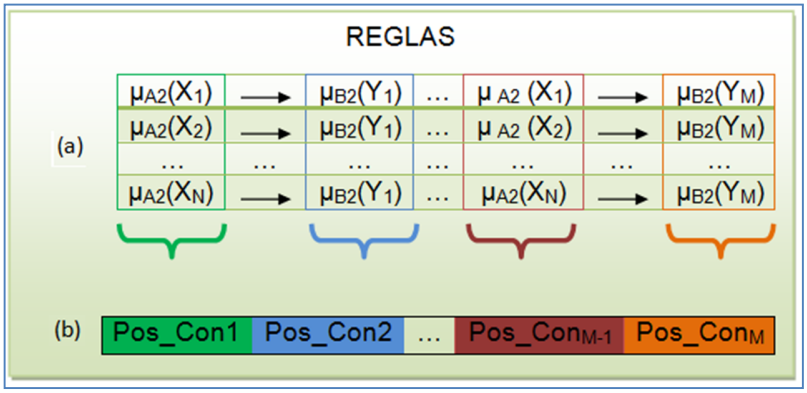


Figura 17: Transformación de la matriz de reglas(a) en un vector de reglas (b).Fuente: Autores.

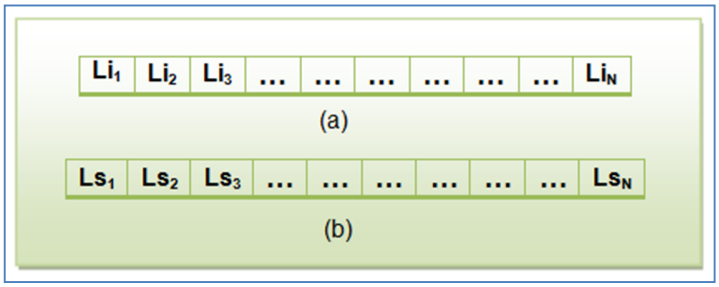


Figura 18: Vector de límites inferiores (a) y vector de límites superiores (b). Fuente: Autores.

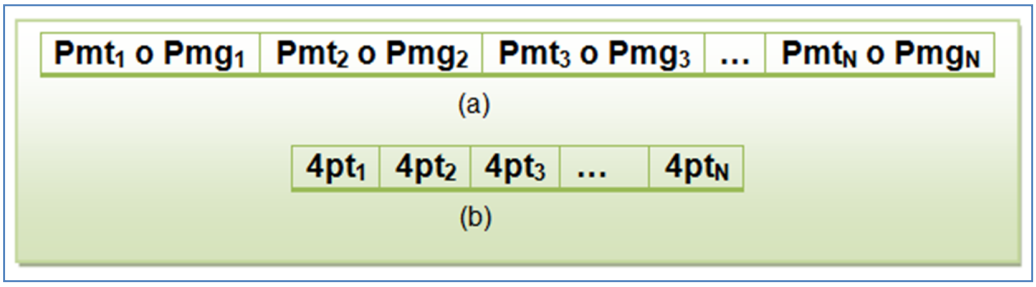


Figura 19: Vector que almacena el punto medio de la función de membresía triangular o gaussiana (a) y vector que almacena el cuarto punto de la función trapezoidal (b). Fuente: Autores

El quinto vector, Figura 19 (a), almacena los puntos medios faltantes para las funciones de membresía triangular y gaussiana, representados como "Pmt_N" y

"Pmg_N" respectivamente. El cuarto punto restante para formar la función trapezoidal se almacenan en el sexto vector, Figura 19 (b), y es representado como "$4pt_N$".

El séptimo vector, Figura 20, "Vf_N" el valor a fuzzificar[13] Con estos valores se evalúa el modelo.

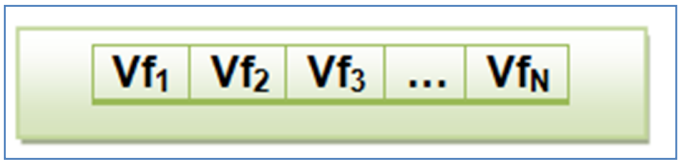


Figura 20: Vector de valores a fuzzificar. Fuente: Autores

2. Para enviar estos vectores a la memoria global de la GPU, se usan las siguientes instrucciones en una clase escrita en lenguaje JAVA.

```
CUdeviceptr AA = newCUdeviceptr();
cuMemAlloc(AA, numEnt * Sizeof.INT);
cuMemcpyHtoD(AA, Pointer.to(hostAA),numEnt * Sizeof.INT);
```

Con "*CUdeviceptr AA = newCUdeviceptr()*" se crea el vector que se almacenara en la memoria global de la GPU, con la instrucción "*cuMemAlloc(AA, numEnt * Sizeof.INT)*" se reserva la memoria necesaria para alojar el vector AA, con "*cuMemcpyHtoD(AA, Pointer.to(hostAA),numEnt * Sizeof.INT)*" se copia el vector a la memoria global de la tarjeta de video.

En la instrucción "*CUfunctionfunction = new CUfunction()*", se crea la función que se a enviar al kernel que se desea lanzar en la GPU, con "*Pointer kernelParameters = Pointer.to(Pointer.to(hostAA))*" se envía el vector al kernel, y finalmente con "*cuLaunchKernel(function,grid,block)*" se lanza el kernel, donde *function* es la función que se llama dentro del *kernel*, *grid* tiene las dimensiones de x, y, z de la *grid* y *block* tiene las dimensiones en x, y que se desean para cada bloque procesado [**5**].

[13]Se entiende por fuzzificar a hallar el grado de pertinencia a partir de la evaluación de un valor real en el antecedente de una entrada.

3. Para formar el antecedente de cada regla, se deben accesar todos los vectores anteriormente descritos, los cuales contienen la información de cada regla; luego, cada regla es procesada en un bloque, dentro de cada bloque, cada antecedente es procesado por un *thread* en la GPU, luego de esto, se revisa a qué tipo de función de membresía pertenece el antecedente y se evalúa mediante la función correspondiente (triangular, trapezoidal o gaussiana), los resultados de la evaluación de estos antecedentes de cada regla se almacenan en un vector, el cual posteriormente es revisado para aplicar el método correspondiente a la regla cuando sea *or* o *and*. Sin embargo, se debe tener en cuenta la capacidad y limitaciones de la GPU en la cual se ejecute el modelo, debido a que variables como el número de *threads*, el número de bloques, la cantidad de memoria y por tanto la capacidad máxima de cómputo difiere entre cada modelo de GPU.

4. En la Figura 21, se observa los resultados teóricos de la paralelización del motor de inferencia, éstos muestran que al aumentar el número de *threads* "T" disminuye la complejidad computacional en el procesamiento de las reglas, también se observa que el rendimiento esperado al variar el número de *threads* mejora al incrementar la granularidad de las mismas; la granularidad se refiere a la segmentación de cada regla en los antecedentes que la conforman. A medida que las reglas poseen más antecedentes para evaluar y el número de *threads* para tratarlas también aumenta, la complejidad de las mismas disminuye.

Los variables disponibles que se tienen en la GPU son: *threads*, bloques y *grids*, los *threads* son la mínima expresión que se tiene en la GPU, cuando se habla de bloques cada uno de estos contiene el mismo número de *threads* y puede tener dimensiones en "x" y "y", así mismo las *grids* contienen los bloques y pueden darse en dimensiones de "x", "y" y "z", sin embargo, estos son recursos limitados en cada GPU, por esta razón el programador debe conocer las limitaciones propias de la GPU en la cual desea ejecutar diferentes aplicaciones mediante CUDA; en la Figura 21 se plantea el desarrollo de la metodología a partir de los

threads de la GPU, ya que éstos son los que proveen la granularidad de la metodología para el problema de la optimización del motor de inferencia.

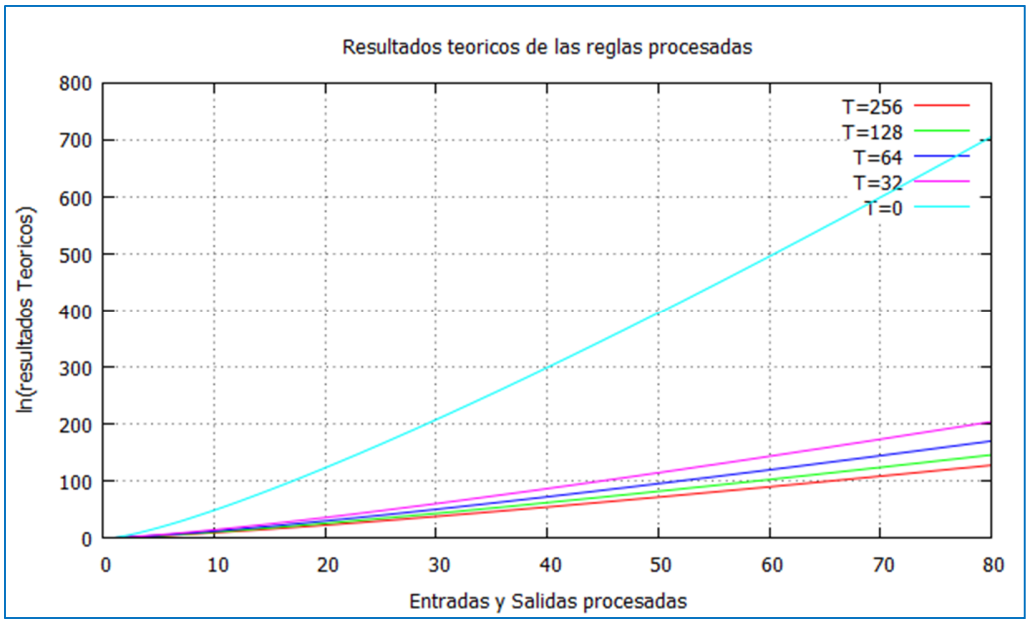


Figura 21: Resultados Teóricos de las reglas procesadas. Fuente: Autores.

La expresión: ln(reglasprocesadas/T), muestra como cada regla es dividida en un número de *threads*, lo cual define la variación de cada línea para "T", dicha ecuación surge del estudio de la metodología para la paralelización de las reglas del artículo *Parallelisation of fuzzy inferenceon a graphics processor unit using the Compute Unified Device Architecture* [**16**] y es linealizada mediante logaritmo natural. Para T=0 se tiene la línea de las reglas sin optimización, para las demás líneas el aumento de los *threads* mejora el rendimiento en la evaluación de las mismas.

4.5.3 Desarrollo de la metodología planteada.

Para que el Prototipo 3 funcione con CUDA se interrumpió el algoritmo del motor secuencial en el momento de evaluar las reglas y se insertó el algoritmo del motor

paralelizado, de esta manera el usuario puede escoger si usar la CPU o la GPU, Figura 22.

Figura 22: Interfaz gráfica para elegir entre GPU y CPU. Fuente: Autores.

La Figura 23 muestra la forma como se acopló la tecnología CUDA al Prototipo 3. En la Clase *JFuzzyCuda* se declara la forma como se envían y reciben los datos en la *GPU* accediendo a *archivo.cu* el cual tiene el código que fue paralelizado en *CUDA*, esto se hace mediante los compiladores *NVCC*[14] de CUDA y C++ de Visual C++ y se obtiene *archivo.ptx,* el cual queda escrito en lenguaje *Assembler*. Para integrarla clase *JFuzzyCuda* al Prototipo 2 se creó la clase *Adapter,* la cual los comunica entre sí, dicha clase toma los datos de la base de conocimiento y los envía a la clase *JFuzzyCuda* y ésta envía resultados al Prototipo 2. Finalmente el algoritmo continúa tal como fue descrito en el Prototipo 2.

[14] NVCC: NVidia Cuda Compiler.

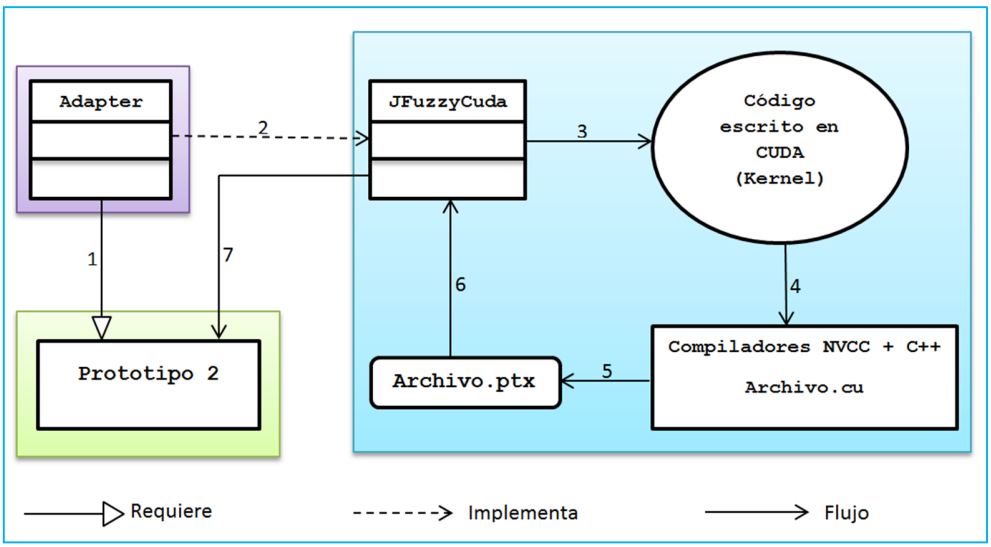


Figura 23: Acople de CUDA al Prototipo 2. Fuente: Autores.

4.5.4 Prototipo 3. Pruebas.

Con el objetivo de medir el rendimiento y la optimización del motor de inferencia se realizaron las siguientes pruebas al Prototipo 3:

- Realizar pruebas con metodología similar a la aplicada en *Parallelisation of fuzzy inference on a graphics processor unit using the Compute Unified Device Architecture* [**16**], con función de membresía trapezoidal (Tabla 4) y gaussiana (Tabla 5).
- Confrontar la metodología aplicada (Figura 21) en la cual se tiene las reglas procesadas versus las entradas y salidas procesadas aumentando el número de *threads,* con una prueba experimental, en la cual se aumentan progresivamente el número de reglas y se mida el tiempo que tardan en ser procesadas por la GPU, variando la cantidad de *threads.*

La tabla 4 muestra el resultado general de los tiempos medidos al procesar, 16, 32, 64 y 128 reglas con función de membresía tipo gaussiana. Los resultados de la columna “JAVA” fueron procesados por la CPU y su unidad de medida es nano-segundo (ns), los resultados de la columna “JCuda” fueron procesados por la

GPU y su unidad de medida es el nano-segundo, la columna “JAVA/JCuda” muestra la efectividad de la GPU, dada en ratio. Un ratio mayor a 1, indica que la GPU ahorra tiempo en el proceso y un ratio menor que 1, indica que la CPU hace el proceso más rápido. El nivel de discretización muestra el incremento en la asignación de recursos de la GPU. Éstos resultados se dieron utilizando el siguiente hardware: GPU, GeForce GT 540M de 1 GB de memoria RAM y CPU procesador core i3 2.53GHz x64 con 4GB de memoria RAM, el máximo número de *threads* que puede manejar dicha GPU es 1024 por bloque, por tanto, para aumentar la discretización es necesario manejar más bloques (Figura 24).

La tabla 4, la cual se trabajó con función de membresía gaussiana, se interpreta de dos maneras, la primera (Figura 25), donde se incrementa el número de reglas procesadas y se mantiene un número fijo de *threads,* allí se observa que mejora el tiempo en la entrega de resultados al evaluar las reglas, esto se evidencia en el aumento del ratio; la segunda (Figura 26), donde se incrementan los *threads* y se mantiene una cantidad de reglas fija, el ratio no aumenta, lo que indica que el aumento de recursos en la GPU no aumenta el rendimiento en la entrega de resultados de las reglas procesadas.

La tabla 5, la cual se trabajó con función de membresía trapezoidal, posee una interpretación similar a la de la tabla 4, y de la misma manera se interpretan las figuras 27 y 28.

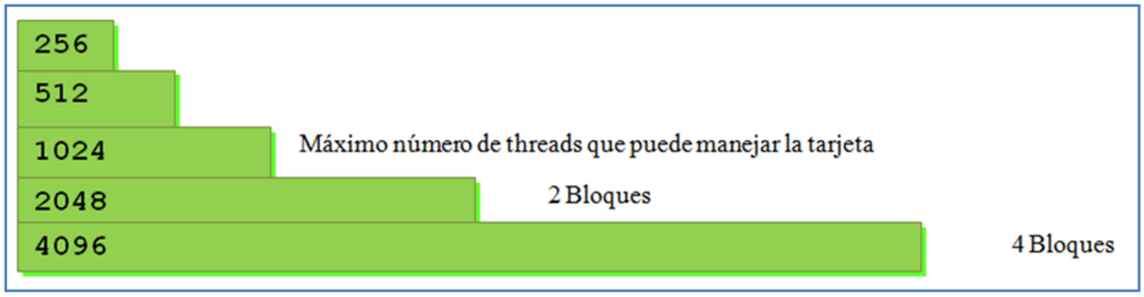


Figura 24: Asignación de *threads* Fuente: Autores.

Tabla 4: Resultados de las pruebas realizadas incrementando el número de reglas procesadas. Función de membresía Gaussiana.

No de reglas procesadas			16	32	64	128
Nivel de discretización.	256	CPU	153550	131672	365441	572875
		GPU	995442	363820	335055	385698
		CPU/GPU	0.15	0.36	1.09	1.48
	512	CPU	110199	136534	370708	566393
		GPU	833789	409197	368682	370303
		CPU/GPU	0.11	0.37	1.10	1.46
	1024	CPU	106148	136939	368277	608123
		GPU	1199635	352882	429454	354502
		CPU/GPU	0.08	0.38	0.85	1.71
	2048	CPU	100071	139775	372328	612580
		GPU	958978	350450	497519	348020
		CPU/GPU	0.10	0.39	0.74	1.76
	4096	CPU	142611	136939	365846	606097
		GPU	978426	363820	354097	434316
		CPU/GPU	0.14	0.37	1.03	1.39

Fuente: Autores.

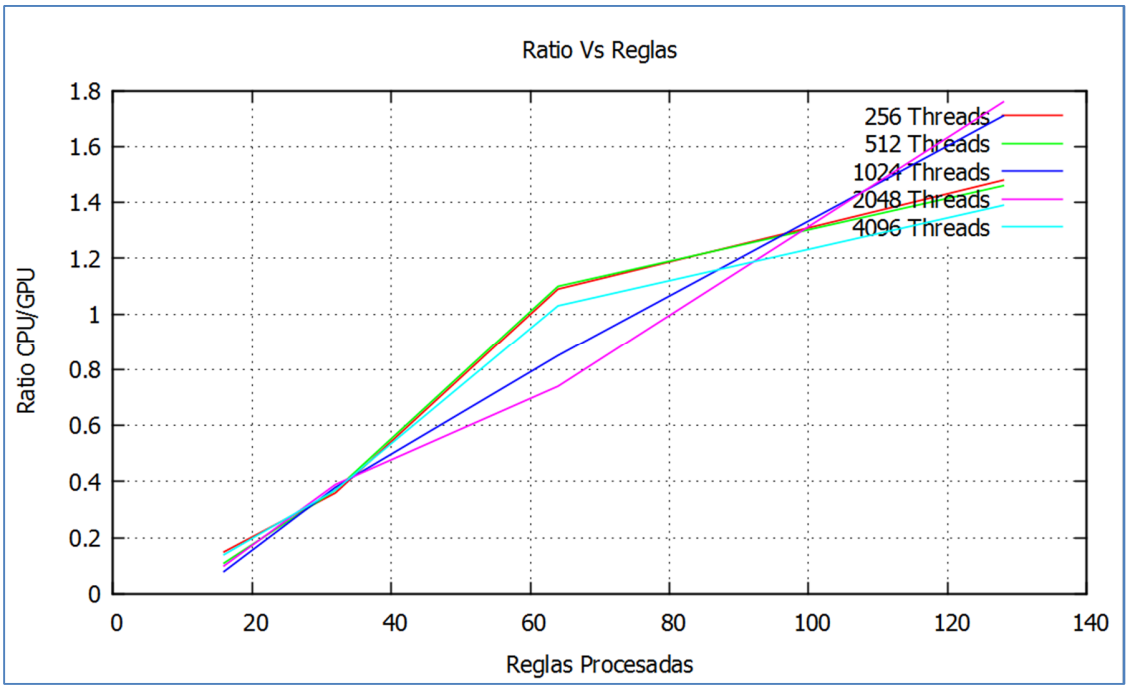


Figura 25: Resultado del ratio con el incremento del número de reglas procesadas, aplicando función de membresía gaussiana. Fuente: Autores.

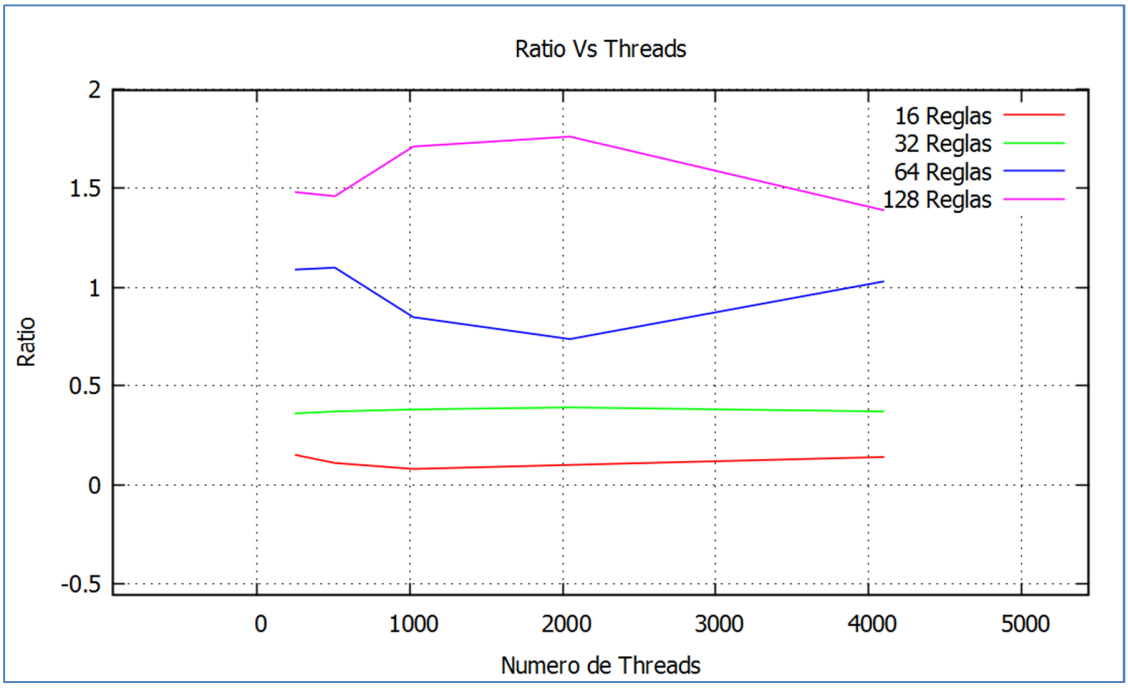


Figura 26: Incremento de recursos por parte de la GPU para un número fijo de reglas, aplicando función de membresía gaussiana. Fuente: Autores.

Tabla 5: Resultados de las pruebas realizadas incrementando el número de reglas procesadas. Función de membresía Trapezoidal.

No de reglas procesadas	NIVEL DE DISCRETIZACIÓN				
	256	512	1024	2048	4096
	JAVA/JCuda	JAVA/JCuda	JAVA/JCuda	JAVA/JCuda	JAVA/JCuda
16	0.16	0.23	0.19	0.17	0.15
32	0.23	0.27	0.32	0.28	0.21
64	0.51	0.60	0.71	0.58	0.72
128	2.04	2.96	1.94	2.73	0.27

Fuente: Autores.

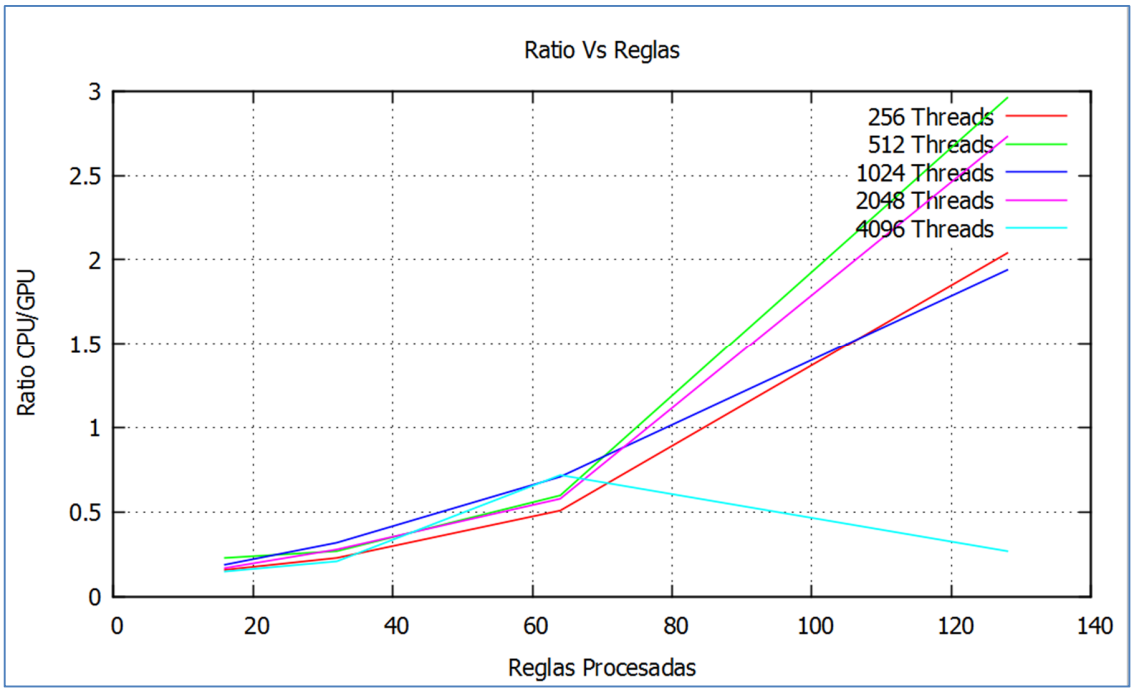


Figura 27: Resultado del ratio con el incremento del número de reglas procesadas, aplicando función de membresía trapezoidal. Fuente: Autores.

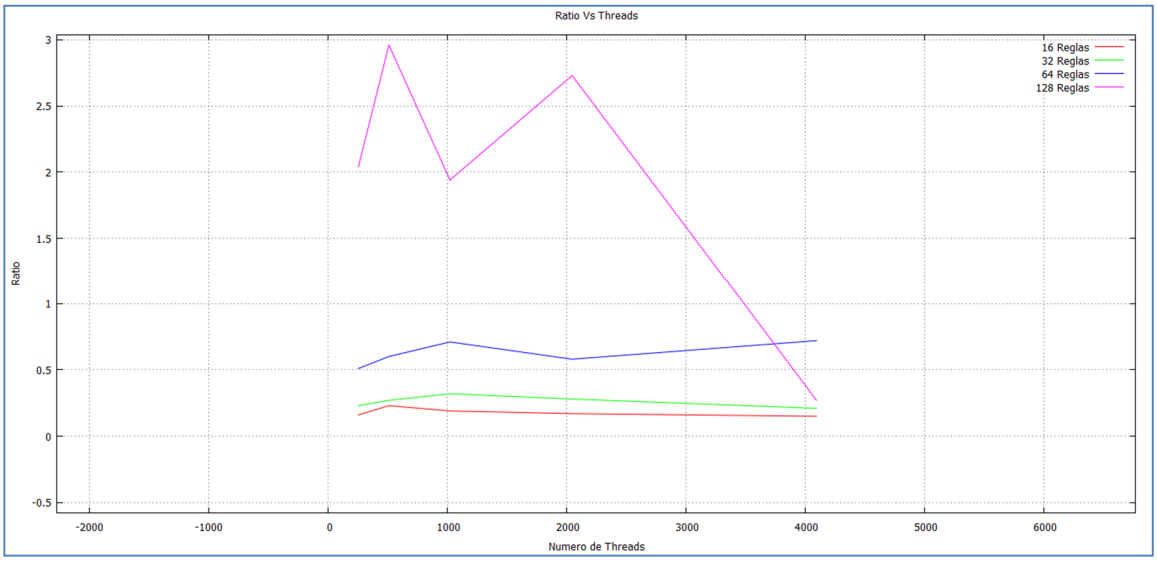


Figura 28: Incremento de recursos por parte de la GPU para un número fijo de reglas, aplicando función de membresía trapezoidal. Fuente: Autores.

La Figura 29 muestra en el eje X el aumento de las reglas procesadas por el motor de inferencia, en el eje Y se tiene el tiempo que toma éste en procesar las reglas, dado en ns, para medir dicho tiempo se interrumpió el flujo del algoritmo del motor de inferencia (Figura 5), después de enviar la base de conocimiento y los vectores mencionados en la metodología (4.4.1.1 Metodología para optimizar el Prototipo 2, pág. 35) a la GPU y cuando se recibe el vector de resultados utilizado para evaluar el "consecuente"; el tiempo de procesamiento de la GPU no parte de cero, debido a *overhead*[15] de comunicación entre la GPU y la CPU (interrupción del proceso por parte de la CPU) .

Las Figuras 25, 26, 27, 28 y 29 fueron realizadas con la herramienta GNUplot.

[15] *Overhead*: Cualquier combinación o exceso indirecto en tiempo de computación, memoria, ancho de banda u otros recursos requeridos para llegar a un objetivo particular.

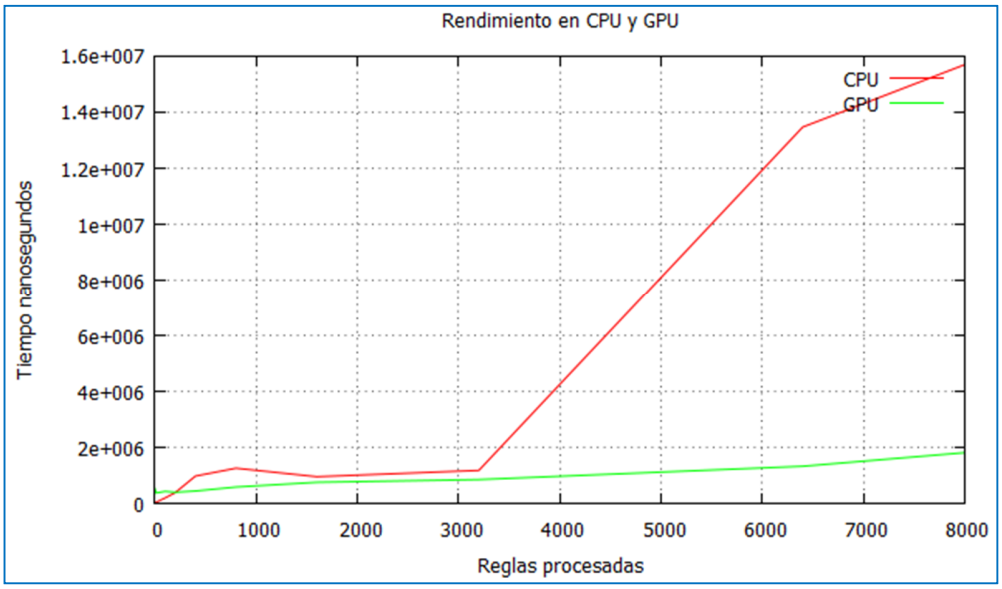


Figura 29: Desempeño de la CPU y GPU. Fuente: Autores.

Tras comparar las Figuras 21 y 29 se observa que los resultados obtenidos experimentalmente, están acorde a los resultados teóricos.

4.5.5 Conclusiones del Prototipo 3.

- Al comparar las figuras 25 y 27 se observa que la complejidad computacional de la función de membresía gaussiana es mayor que la presentada por la función trapezoidal, esto debido a la carga extra en el cálculo de la función exponencial gaussiana.
- La tendencia de las figuras 25, 27 y 29 muestra que la implementación del problema en la GPU mejora el tiempo en la entrega de resultados cuando el volumen de reglas evaluadas es mayor.
- Aumentar los recursos disponibles en la GPU y dejar fijo el número de reglas no mejora los tiempos en la entrega de resultados, figuras 26 y 28.

- Al observar los picos o máximos en las figura 28, se concluye que los recursos asignados deben ser acordes al problema tratado (asignar una cantidad de recursos mayor y lo más cercana posible al nivel de discretización de las reglas procesadas).
- Al comparar los resultados teóricos con los experimentales se observa que las tendencias de las Figuras 21 y 29 son similares, al procesarse con la GPU, lo cual indica que la metodología planteada para optimizar el prototipo 2, fue correctamente desarrollada.
- Se observa que la implementación de la GPU mejora el tiempo en la entrega de resultados del prototipo 3 cuando el número de reglas que se desea evaluar, también se incrementa, es decir que al evaluar modelos a gran escala, el procesamiento con CUDA ofrece una alternativa para el ahorro en el tiempo de cómputo.

5. CONCLUSIONES GENERALES Y RECOMENDACIONES

5.1 CONCLUSIONES GENERALES.

- La herramienta NOICA provee a la comunidad científica una alternativa para elaboración de modelos que no pueden ser planteados mediante la matemática convencional o que hagan uso de la inteligencia artificial.
- La construcción de la herramienta NOICA permitió aplicar y afianzar conocimientos en el campo de la Ingeniería de Sistemas ya que se usó la ingeniería del software, inteligencia artificial, programación distribuida, construcción y análisis de algoritmos, entre otros tópicos indispensables para la elaboración de un producto software.
- Se complementó la Lógica Difusa mediante el uso de nuevas tecnologías como CUDA, mediante JCuda, brindando una herramienta a la comunidad de la Universidad Industrial de Santander para el apoyo en investigaciones que hagan uso de la lógica difusa.

5.2 RECOMENDACIONES

- Emplear un algoritmo de entrenamiento para que de la base de reglas se retiren aquellas que no son necesarias, según los requerimientos del usuario.
- Continuar con la investigación para añadir más motores de inferencia a la herramienta NOICA.
- Comunicar la herramienta con los diferentes puertos del PC para el manejo de otro tipo de tecnologías.
- Implementar una metodología para exportar la base de conocimiento y que ésta se ejecute en un súper computador.
- Llevar NOICA a la comunidad para que sea utilizada por diferentes ramas del conocimiento.

6. BIBLIOGRAFÍA

[1] E. H. Mamdani, *Application of Fuzzy Logic to Approximate Reasoning Using Linguistic Synthesis*.: IEEE Transactions on Computers, 1977.

[2] G. J., & Yuan, B. Klir, *FROM CLASSICAL (CRISP) SETS TO FUZZY SETS: A Grand Paradigm Shift*.: Prentice-Hall, 1995.

[3] Durán Vicente M.I. Matías T.B, *Lógica Borrosa*.: Universidad Carlos III, 2009.

[4] NVIDIA Corporation. (2011) NVIDIA. [Online]. http://www.nvidia.es/object/what_is_cuda_new_es.html

[5] jcuda@jcuda.org. jcuda.org. [Online]. http://www.jcuda.de/

[6] Sands T Kent R, *High Performance Computing Systems and Applications, Kluwer Academi Publishers*., 2003.

[7] MathWorks. (2012, Mayo) Fuzzy Logic Toolbox. [Online]. http://www.mathworks.com/products/fuzzy-logic/.

[8] L. A. Zadeh, *Fuzzy Logic, Neural Networks and Soft Computing*.: Communications of the ACM, 1994.

[9] Sosa M.C, *INTELIGENCIA ARTIFICIAL EN LA GESTION FINANCIERA EMPRESARIAL*., 2007.

[10] Roger Pressman, *Ingeniería del Software, un enfoque práctico.*

[11] S. Reid, *The Art of Software Testing, Second edition. Glenford J. Myers.*

Revised and updated by Tom Badgett and Todd M. Thomas, with Corey Sandler. John Wiley and Sons, New Jersey., 2004.

[12] B.I. Santos, *Deteccion de enfermedades de la retina que propenden a la ceguera. Tesis de maestría. Instituto Politécnico Nacional, México D.F.*, 2009.

[13] ISO/IEC 9126-1, *International Organization for Standardization (ISO). Software engineeringProduct quality - Part 1: Quality model.*, 2001.

[14] Bautista L. X, Galvis E Reyes C. E, *NOICA: Herramienta para la Creación y Estudio de Modelos Exhaustivos Desarrollados en Lógica Difusa*., 2012.

[15] Luke R., Keller J., Anderson D. Harvey N., *Speedup of Fuzzy Logic through Stream Processing on Graphics Processing Units*., 2008.

[16] Coupland s. Anderson D., *Parallelisation of Fuzzy Inference on a Graphics Processor Unit Using the Compute Unified Device Architecture, Proceedings of the 2008 UK Workshop on Computational Intelligence.*, 2008.

ANEXOS

ANEXO A: INFORMACIÓN COMPLEMENTARIA DE LOS PROTOTIPOS.

PROTOTIPO I:

Con el fin de hacer una prueba efectiva en tiempo corto, para estudiar la matemática de la lógica difusa, se hizo un programa piloto en un lenguaje que representa una rápida implementación y evaluación de resultados. Se optó por implementarlo en modo consola no amigable para un usuario final, sino más bien fiel a los propósitos iniciales.

ALCANCE DEL PROTOTIPO 1:

- Ingresar la variable de entrada y sus respectivo nombre y rango o universo del discurso.
- Ingresar la variable de salida y su respectivo nombre y rango.
- Ingresar máximo 1 variable de entrada, con máximo con 5 subconjuntos difusos.
- Ingresar máximo 1 variable de salida, con máximo 3 subconjuntos difusos.
- El usuario inserta de forma manual las reglas.
- El prototipo entrega el cálculo de la respuesta.

Actor

Basado en las características que presenta el motor de inferencia y los servicios implementados, se presenta usuario como único Actor (Usuario es el único actor en cada prototipo).

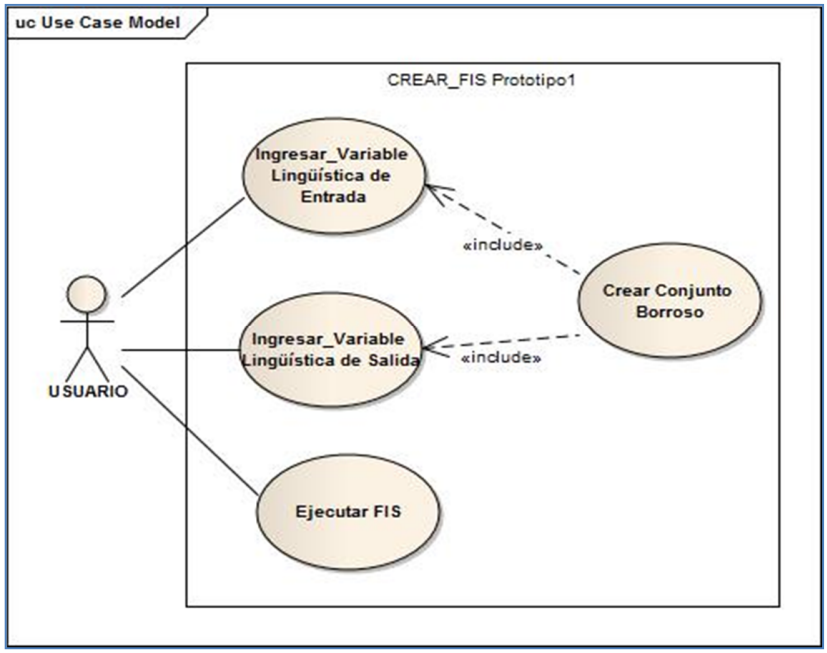
Tabla a. Descripción del actor involucrado.

ACTOR	DESCRIPCIÓN	RESPONSABILIDADES
Usuario	Representa la persona que maneja la herramienta. (Usuario que maneja un área de conocimiento que aplica Lógica Difusa.)	Ingresa variable lingüística de entrada. Ingresa variable lingüística de salida. Ingresa reglas. Ejecuta el FIS.

Fuente: Autores

La siguiente figura corresponde a l diagrama de casos de uso del Prototipo 1 y tiene como actor al usuario en el escenario crear_FIS.

Figura a. Diagrama de casos de uso. Prototipo 1



Fuente: Autores.

La siguiente es la tabla para el caso de uso Ingresar Entrada.

Tabla b. Caso de uso: Ingresar variable lingüística de entrada.

Nombre:	**Ingresar variable lingüística de entrada.**
Actor(es):	Usuario
Descripción:	El usuario ingresa el número de entradas del FIS, coloca nombre y a cada entrada agrega un conjunto difuso.
Precondición(es):	Se necesita tener un modelo planteado.
Flujo Principal:	Después de ejecutar el programa se presenta un menú para crear un Nuevo Fis o salir del programa. Al crear un nuevo fis se pide el ingreso del universo de entrada, se ingresa la entrada y se coloca el nombre, luego se agregan los subconjuntos borrosos para la entrada.
Pos-condición(es):	Continuar con el ingreso del modelo.
Sub-flujos:	Agregar subconjunto borrroso: ingresar número de subconjuntos borrosos: ingresar variables lingüísticas, función de membresía y rango.
Excepciones:	Sólo nombres de funciones de membresía autorizadas.

Fuente: Autores

PROTOTIPO 2

Los requisitos de la aplicación, necesaria para el desarrollo del prototipo 2; se presentan a continuación, junto con las funcionalidades de la aplicación representadas en el *Diagrama de Casos de Uso* con su respectiva descripción y se identifica el usuario final de la aplicación.

ALCANCE DEL PROTOTIPO 2:

- Crear y configurar la base de reglas.
- Editar los parámetros del motor de inferencia.
- Editar de forma interactiva las entradas y salidas del modelo, ya que éstas se encuentran en un archivo de texto plano.
- Ejecutar el modelo planteado.

Actor

Basado en las características que presenta el Prototipo 2 y los servicios implementados, se presenta el usuario como único Actor.

Tabla c. Descripción del actor involucrado el Prototipo 2.

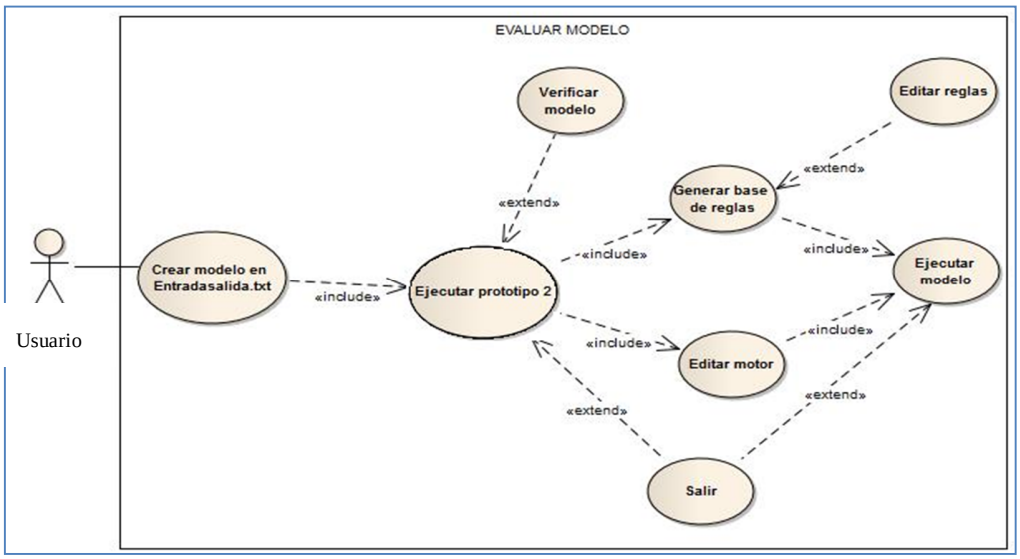
ACTOR	DESCRIPCIÓN	RESPONSABILIDADES
Usuario	Representa la persona que maneja el Prototipo 2.	• Ingresa los valores para cada una de las entradas y salidas, definidas mediante el archivo *ptseval:* se escriben los valores para cada una de las entradas o salidas separadas con el símbolo $. • Crea la base de reglas. • Genera todas las reglas. • Deshabilita reglas. • Elige AND u OR para las reglas. • Define AND u OR en el motor de inferencia, así como agregación o implicación.

		• Ejecuta el modelo.

Fuente: Autores

Diagrama de casos de uso del prototipo 2.

Figura b. Diagrama de casos de uso. Prototipo 2.



Fuente: Autores.

La siguiente es la tabla para el caso de uso Ingresar Entrada.

Tabla d. Caso de uso: Ejecutar Prototipo 2.

Nombre:	**Ejecutar Prototipo 2.**
Actor(es):	Usuario
Descripción:	El usuario construye un modelo en un archivo de texto plano y lo ejecuta.
Precondición(es)	Se necesita tener un modelo planteado.

:	
Flujo Principal:	Se ingresan los valores de Entradas y Salidas en un archivo de texto plano llamado ptoseval, se carga el modelo y se ejecuta y uego se hace la configuración de la base de reglas y el motor de inferencia (si es requerido).
Pos-condición(es):	Continuar con el ingreso de modelos.
Sub-flujos:	Configurar motor de inferencia y base de reglas.
Excepciones:	Ingreso del modelo con el lenguaje se símbolos autorizado.

Fuente: Autores

PROTOTIPO 3

Los requisitos de la aplicación, necesaria para el desarrollo del prototipo 3; se presentan a continuación, junto con las funcionalidades de la aplicación representadas en el *Diagrama de Casos de Uso*.

ALCANCE DEL PROTOTIPO 3:

- Crear y configurar la base de reglas.
- Editar los parámetros del motor de inferencia.
- Editar de forma interactiva las entradas y salidas del modelo, ya que éstas se encuentran en un archivo de texto plano.
- Ejecutar el modelo planteado con CPU o con GPU.

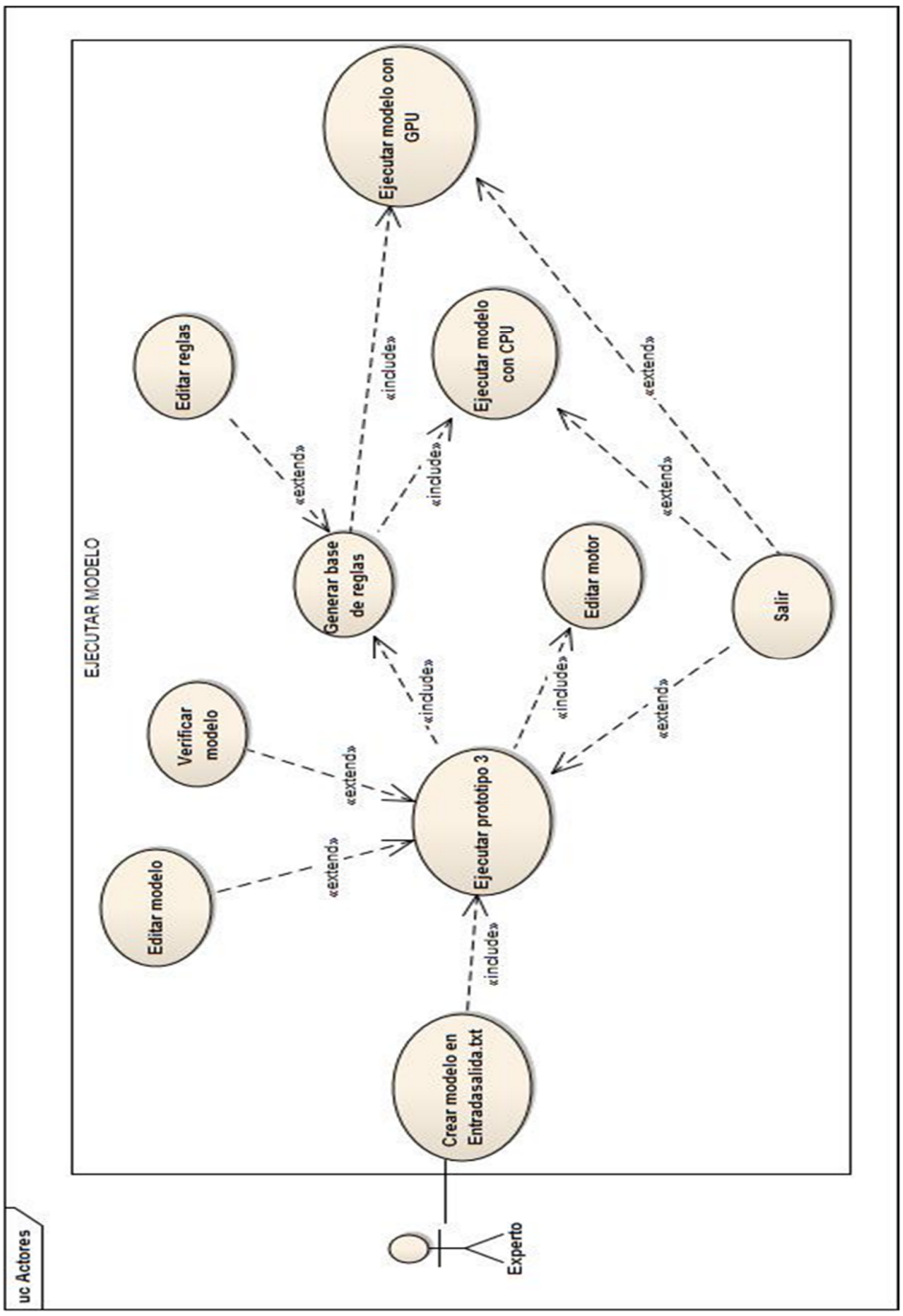
Actor

Basado en las características que presenta La herramienta prototipo II y los servicios implementados, se presenta el Usuario o Usuario como único Actor (Usuario es el único actor en cada prototipo).

Tabla e. Descripción del actor involucrado en La herramienta Prototipo I.

ACTOR	DESCRIPCIÓN	RESPONSABILIDADES
Usuario	Representa la persona que maneja la herramienta.	• Ingresa los valores para cada una de las entradas y salidas, definidas mediante el archivo *ptseval:* se escriben los valores para cada una de las entradas o salidas separadas con el símbolo $. • Crea la base de reglas. • Genera todas las reglas. • Deshabilita reglas. • Elige AND u OR para las reglas. • Define AND u OR en el motor de inferencia, así como agregación o implicación. • Ejecuta el modelo con CPU o con GPU.

Figura c. Diagrama de casos de uso. Prototipo 3.



Fuente: Autores.

La siguiente es la tabla para el caso de uso Ejecutar Prototipo 3.

Tabla f. Caso de uso: Ejecutar Prototipo 3.

Nombre:	**Ejecutar Prototipo 3.**
Actor(es):	Usuario
Descripción:	El usuario construye un modelo en un archivo de texto plano y lo ejecuta.
Precondición(es):	Se necesita tener un modelo planteado.
Flujo Principal:	Se ingresan los valores de Entradas y Salidas en un archivo de texto plano llamado ptoseval, se carga el modelo y se ejecuta con CPU o con GPU y luego se hace la configuración de la base de reglas y el motor de inferencia (si es requerido).
Pos-condición(es):	Continuar con el ingreso de modelos.
Sub-flujos:	Configurar motor de inferencia y base de reglas.
Excepciones:	Ingreso del modelo con el lenguaje se símbolos autorizado.

Fuente: Autores