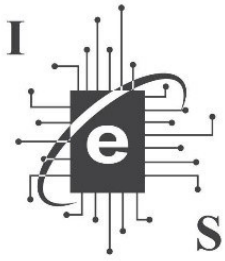


PROYECTO 2º CORTE

INTELIGENCIA ARTIFICIAL

ESTUDIANTES:

AMAURY RAFAEL ORTEGA CAMARGO

JUAN DIEGO BENÍTEZ ARIAS

PRESENTADO A:

JOHN CARLOS ARRIETA ARRIETA

UNIVERSIDAD DE CARTAGENA

PROGRAMA DE INGENIERÍA DE SISTEMAS

Cartagena, Colombia

2017

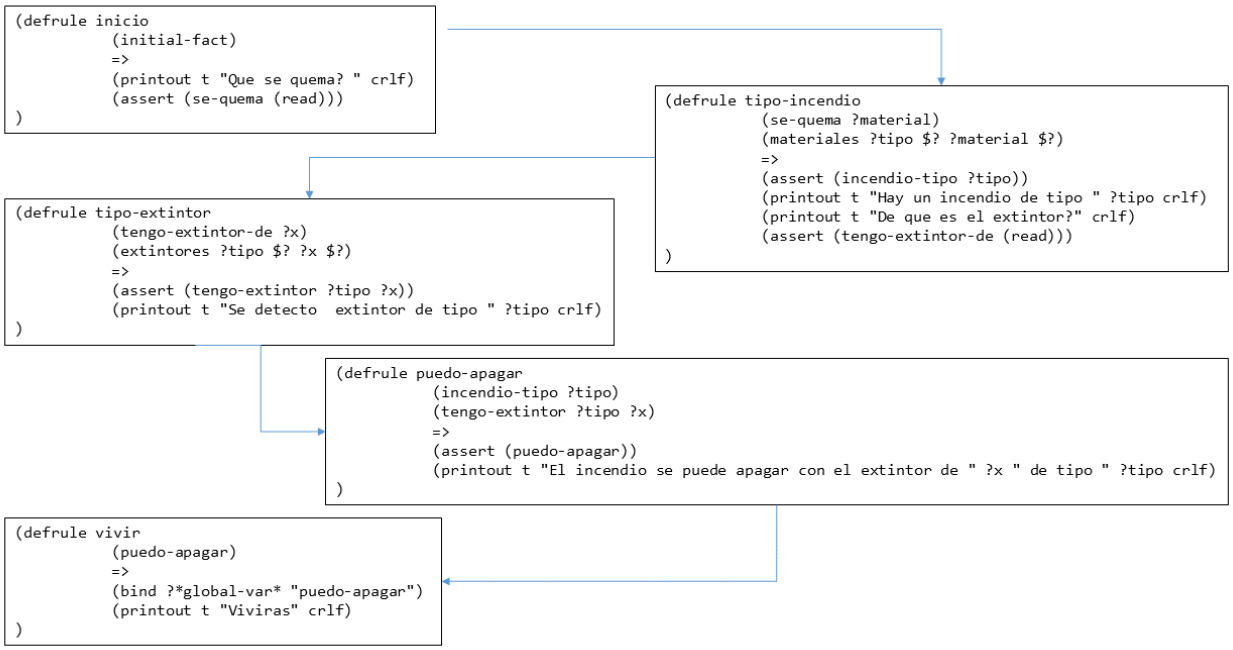
1 Bases de conocimientos

1.1 CLIPS

Se manejan los siguientes hechos y una variable global que nos servirá saber a qué conclusión llego la base de conocimiento.

```
(defglobal ?*global-var* = "")

(deffacts inicio

    (materiales A papel madera tela)

    (materiales B aceite gas lubricante)

    (materiales C magnesio sodio potasio)

    (extintores A agua sustancia-quimica-seca)

    (extintores B co2 sustancia-quimica-seca)

    (extintores C co2 espuma)

)
```

Y tenemos las siguientes reglas que se ejecutaran de forma encadenada para determinar si se puede apagar el incendio.



Debido a que CLIPS hace inferencia hacia adelante, CLIPS está constantemente verificando que hecho/s disparan que reglas. Debido a que se diseñaron las reglas para que se ejecutarán de forma encadenada, quiere decir que una regla A ingresa un hecho que es necesario para que se dispare la regla B, esto permite mantener un flujo en la ejecución. Aunque es cierto que tener las reglas de esta forma es necesario ni que para usar esta base de conocimiento siempre se debe comenzar desde el inicio, debido a que se puede disparar la regla B sin dispararse la regla A (continuando la analogía anterior), se optó por hacerlo de esta forma debido a que este fue el ejercicio con el que se explicó en clase el funcionamiento de CLIPS, para ver estos archivos puede dirigirse a la referencia [2] o [3].

1.2 SWI-PROLOG

Se manejan los siguientes hechos.

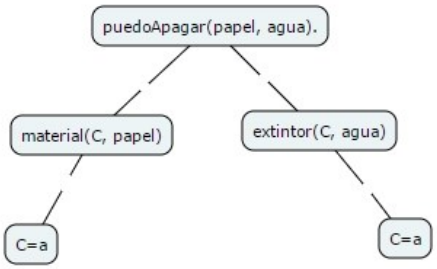
```
material(a, papel).

material(a, madera).

material(a, tela).

material(b, aceite).

material(b, gas).

material(b, lubricante).

material(c, magnesio).

material(c, sodio).

material(c, postasio).

extintor(a, agua).

extintor(a, sustancia-quimica-seca).

extintor(b, co2).

extintor(b, sustancia-quimica-seca).

extintor(c, co2).

extintor(c, espuma).
```

Y reglas.

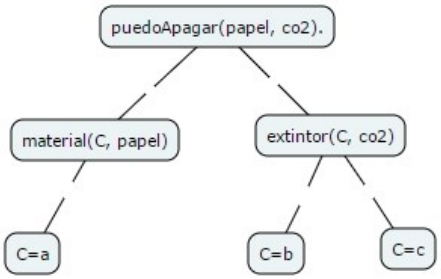
```
% X=material y Y=extintor de y C=tipo incendio

puedoApagar(X, Y) :- material(C, X), extintor(C, Y).
```

En SWI-PROLOG se optó por optimizar la base de conocimiento pues en clase fue aclarado por el docente John Arrieta que SWI-PROLOG es requiere más recursos de computo debido a la forma en como hace inferencia, la cual es hacia atrás. Esto se puede explicar usando un árbol.

Cuando retorna True.



Cuando retorna False.



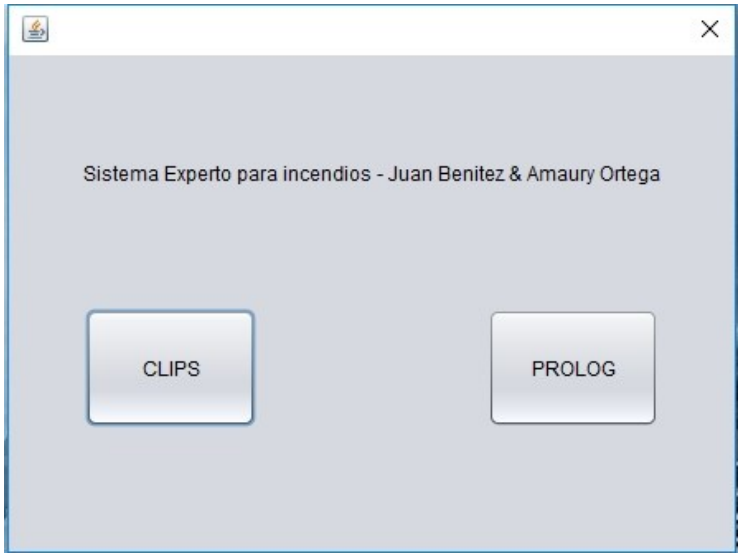
En SWI-PROLOG las variables son en mayúscula y como la regla puedoApagar recibe 2 parámetros ingresados por el usuario, es deber de SWI-PROLOG encontrar todos los hechos que coincidan con dichos parámetros, pero esos hechos deben compartir una similitud C. Esta variable C hace referencia al tipo de extintor e incendio lo cual permite determinar si un extintor de cierto tipo, es válido para un incendio de ese mismo tipo.

2 Interfaz de usuario e implementación

Para la implementación se utilizó Java con su librería swing que nos permite tener una interfaz gráfica. Para tratar con ambas bases de conocimiento se utilizó CLIPSJNI.jar para CLIPS y jpl.jar para SWI-PROLOG. A continuación, se mostrará la implementación de cómo se interactúa con la base de conocimiento en ambos casos y la interfaz gráfica que observa el usuario. Para ver la totalidad del código, puede dirigirse a la referencia [3].

2.1 INICIO

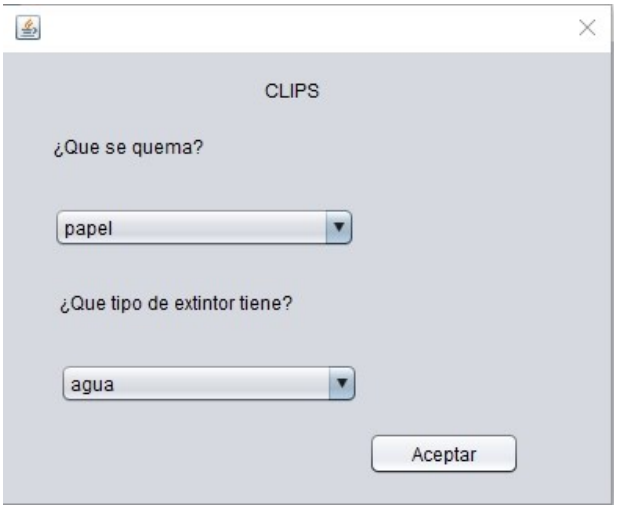
Aquí el usuario puede seleccionar que base de conocimiento usar. Cada botón hace que se cree otra ventana, cada uno llamara a una clase distinta debido a que tendremos una clase para CLIPS y otra para SWI-PROLOG.

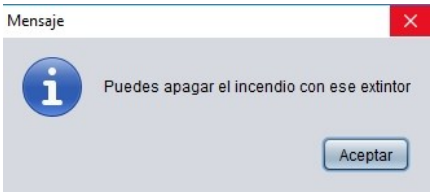


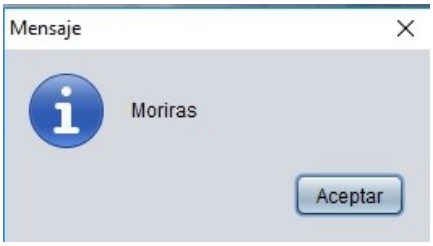

```
private void jBClipsActionPerformed(java.awt.event.ActionEvent evt) {
    Clips clips = new Clips(null, true);
    clips.setVisible(true);
}

private void jBPrologActionPerformed(java.awt.event.ActionEvent evt) {
    Prolog prolog = new Prolog(null, true);
    prolog.setVisible(true);
}
```

2.2 CLIPS

Cuando el usuario presiona el botón CLIPS, vera la siguiente ventana donde podrá seleccionar que material se está quemando y que contiene su extintor para que al presionar el botón aceptar, java hará la conexión con la base de conocimiento de CLIPS y determinara si el extintor le sirve para apagar dicho incendio o si morirá debido a que el extintor no es del tipo correspondiente para ese incendio. Se puede ver, después de la siguiente imagen, lo que sucede cuando es y no es posible apagar el incendio.







En implementación, para poder usar CLIPS desde java se hizo lo siguiente. Al comienzo de la clase Clips.

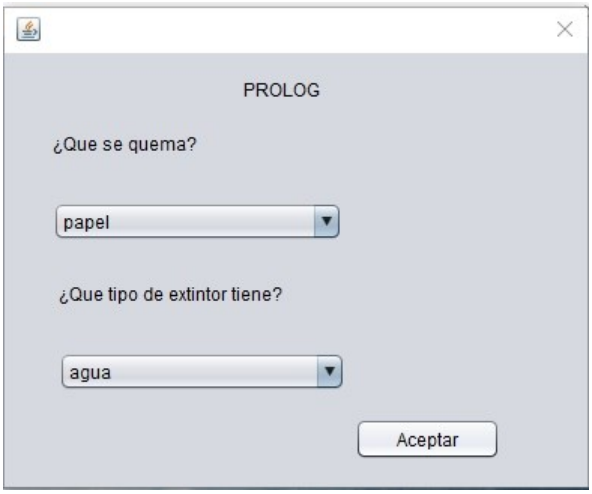
```
//CLIPS 1. Importando de CLIPSJNI.jar
import net.sf.clipsrules.jni.*;

public class Clips extends javax.swing.JDialog {

    //CLIPS 2. Se añade un atributo Environment para tener acceso a la base
    //         de conocimiento
    private Environment clips;

    public Clips(java.awt.Frame parent, boolean modal) {
        super(parent, modal);
        //CLIPS 3. Se crea y se carga la base de conocimiento en el atributo
        //          atributo clips de tipo Environment
        clips = new Environment();
        clips.load("programa.clp");
        initComponents();
    }
```

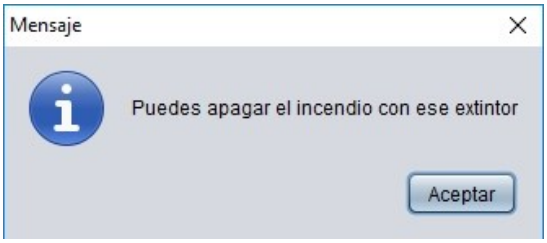
Y finalmente en el método del botón aceptar se hace uso de la base de conocimiento ya cargada. Se hace reset para que se borren los hechos y se carguen los hechos iniciales que no incluye el conocimiento generado por el sistema. Luego se usa el método eval que nos permite usar sintaxis de CLIPS para agregar los hechos ingresados por el usuario. Se usa el método run para ejecutar aquellas reglas que hayan sido disparadas hasta este momento en la base de conocimiento y se usa la clase PrimitiveValue para poder obtener el valor contenido en la variable global de la base de conocimiento y así determinar a qué conclusión llego la base de conocimiento.

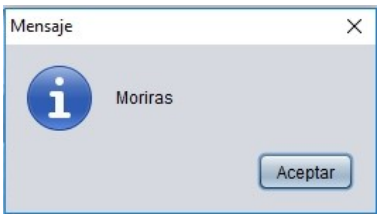
```
private void jBAceptarActionPerformed(java.awt.event.ActionEvent evt) {
    //CLIPS 3. Se hace uso de (reset) para preparar los hechos de la base de
    //          conocimiento usando los deffacts que estan en ella
    clips.reset();

    //CLIPS 4. Se añaden los hechos que el usuario ingresa
    String material = (String) jCBMaterial.getSelectedItem();
    String Extintor = (String) jCBExtintor.getSelectedItem();
    clips.eval("(assert (se-quema " + material + "))");
    clips.eval("(assert (tengo-extintor-de " + Extintor + "))");

    //CLIPS 5. Se ejecutan los hechos que estan almacenados en la agenda de clips
    clips.run();

    //CLIPS 6. Se busca el valor de la variable global y se determina si puede apagar el incendio
    String evalString = "?*global-var*";
    PrimitiveValue rv;
    rv = clips.eval("?*global-var*");
    try {
        if (rv.toString().equals("\"puedo-apagar\"")) {
            JOptionPane.showMessageDialog(this, "Puedes apagar el incendio con ese extintor");
        } else {
            JOptionPane.showMessageDialog(this, "Moriras");
        }
    } catch (Exception e) {
        System.out.println("Problemas leyendo variable global de clips");
    }
```

2.3 SWI-PROLOG

Cuando el usuario presiona el botón PROLOG, vera la siguiente ventana donde podrá seleccionar que material se está quemando y que contiene su extintor para que al presionar el botón aceptar, java hará la conexión con la base de conocimiento de SWI-PROLOG y determinara si el extintor le sirve para apagar dicho incendio o si morirá debido a que el extintor no es del tipo correspondiente para ese incendio. Se puede ver, después de la siguiente imagen, lo que sucede cuando es y no es posible apagar el incendio.







En implementación, para poder usar SWI-PROLOG desde java se hizo lo siguiente. Al comienzo de la clase Prolog.

```
//PROLOG 1. Importando de jpl.jar
import org.jpl7.*;
```

```
    private void jBAceptarActionPerformed(java.awt.event.ActionEvent evt) {
        //PROLOG 2. Se crea un query para cargar la base de conocimientos
        Query q1 = new Query("consult('programa.pl')");
        //PROLOG 3. Hay que terminar la ejecucion de q1 verificando sus soluciones
        //          Aqui se retorna true o false
        q1.hasSolution();

        //PROLOG 4. Se extrae lo que el usuario ingresa
        String material = (String) jCBMaterial.getSelectedItem();
        String extintor = (String) jCBExtintor.getSelectedItem();
        //PROLOG 4. Se preara un query consultando a la base de conocimiento
        //          Usando la entrada del usuario
        Query q2 = new Query("puedoApagar(" + material + ", " + extintor + ")");
        //PROLOG 5. Se consulta si q2 retorno true o false
        if(q2.hasSolution()){
            JOptionPane.showMessageDialog(this, "Puedes apagar el incendio con ese extintor");
        }else{
            JOptionPane.showMessageDialog(this, "Moriras");
        }
    }
```

Y finalmente en el método del botón aceptar se crea una sentencia usando la clase Query para cargar la base de conocimiento. Debido a que SWI-PROLOG al momento de retornar, espera que algo o alguien reciba su salida, es necesario ver esta salida usando el método hasSolution para completar la sentencia dedicada a cargar la base de conocimiento, debido a que esto retornara true cuando se cargue la base de conocimiento correctamente, no es necesario guardar este valor.

Seguido a esto se obtiene lo que ingresa el usuario de la vista y se crea una segunda sentencia para hacer la consulta a la base de conocimiento, esta consulta retornara true cuando sea posible apagar el incendio y false en lo contrario, por lo que se vuelve a hacer uso del método hasSolution para esto.

3 Bibliografía

[1] Engineering, D. o. (s.f.). *Clipsjni: Clips Java Native Interface*. Obtenido de https://www.csie.ntu.edu.tw/~sylee/courses/clips/clipsjni.htm

[2] Ortega, A. (2017). *Github.* Obtenido de Amaury Ortega: https://github.com/AmauryOrtega/Clips-example

[3] Ortega, A., & Benitez, J. (2017). *Github.* Obtenido de Juan Benitez y Amaury Ortega: https://github.com/AmauryOrtega/IA-SE

[4] Riley, G. (s.f.). *CLIPS Reference Manual Volume II*. Obtenido de http://clipsrules.sourceforge.net/documentation/v630/apg.pdf

[5] Singleton, P., Dushin, F., & Wielemaker, J. (s.f.). *JPL: A bidirectional Prolog/Java interface*. Obtenido de http://www.swi-prolog.org/packages/jpl/