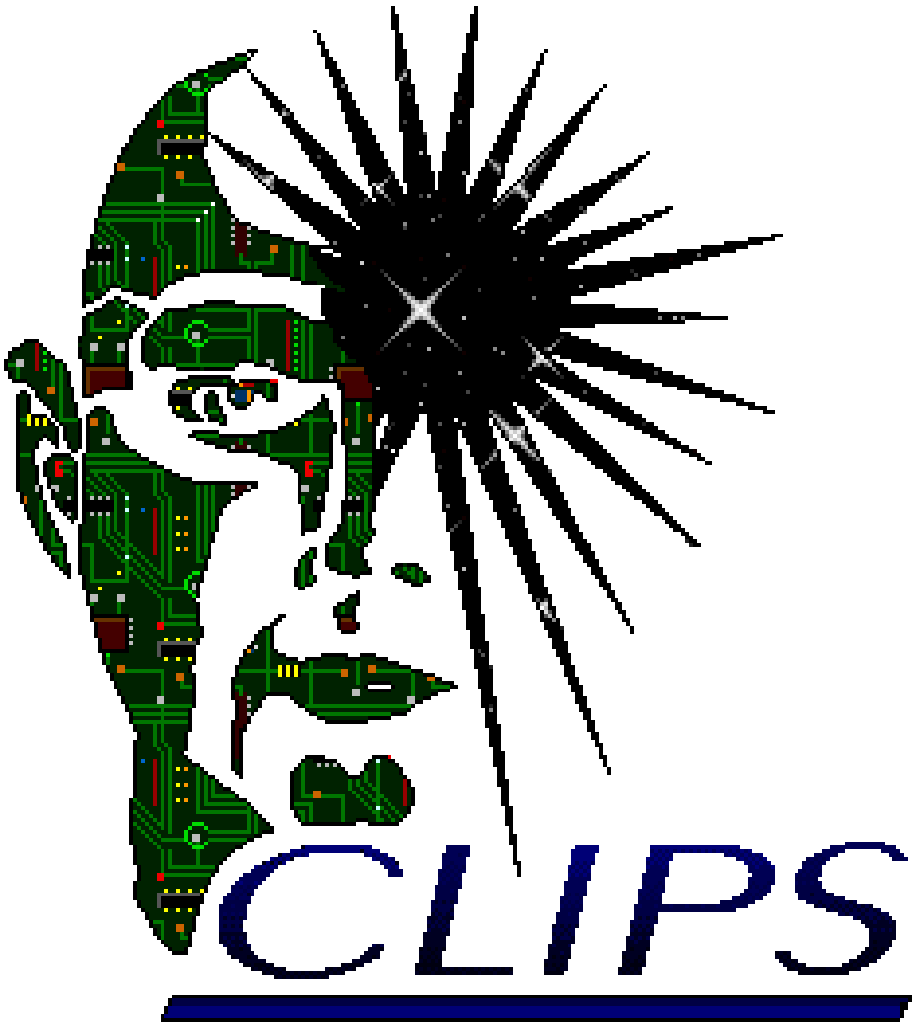
SISTEMA DE CONTROL PARA UN CENTRO DE DATOS



David Fernández Prieto 44663898Z davidvispo4211@gmail.com
Daniel Barandela Cachafeiro 45163395C danielbaracacha@gmail.com

ÍNDICE

DESCRIPCIÓN DEL PROBLEMA

Este proyecto se trata de un sistema de control basado en reglas para un Centro de Datos. El sistema controlará las medidas de temperatura y humedad en sus diferentes zonas, así como el acceso por niveles a estas y reaccionará ante la presencia de humo con distintas acciones (dependiendo de si esa zona es sensible o no).

Hemos simplificado la propuesta respecto a la original, eliminando la comprobación de los racks y unificando en control de acceso a zonas a un solo sensor que mide el nivel de acceso de quién intenta entrar y cuántas personas hay en esa sala.

Por otra parte, añadimos un detector de metales para la entrada y como evento, la aparición de una persona ajena con nivel de acceso cero.

MANUAL DE USO

Requisitos previos:

-Tener instalados python3 y la librería clipspy.

-Para descargar clipspy en Linux es necesario crear previamente un entorno y ejecutar el comando: ‘pip install clipspy’

Ejecución del programa:

-Debe abrirse la carpeta llamada ‘clips-main’ como workspace y ejecutar el archivo main.py

IMPLEMENTACIÓN

Para el desarrollo del proyecto hicimos uso de CLIPS, una herramienta para la producción de sistemas expertos. Se basa en un conjunto de reglas (que especifican acciones a realizar cuando ciertas condiciones se cumplen) y una base de hechos (que almacena datos sobre el estado actual del sistema). CLIPS compara constantemente los hechos con las reglas y ejecuta las acciones correspondientes cuando se detecta una coincidencia.

En nuestro proyecto de control de un Centro de Datos, definimos las siguientes reglas y hechos:

HECHOS

-Usuarios: define los usuarios que pueden moverse por el Centro. Guardamos su nombre, nivel de acceso y ocupación.

-Zonas: son las salas en las que se divide nuestro Centro. Sus atributos serán su nombre, nivel de acceso necesario, ocupación máxima, ocupación actual y el tipo de contenido (sensible o no sensible, que determinará en caso de incendio, cómo actuar).

A parte de los hechos que están en la base, los hechos referentes a los sensores los insertamos al motor CLIPS desde el propio python para disparar las reglas. Para los distintos casos de ejemplo, tenemos:

-Incendio: se dispara el sensor de humo de una sala.

-Temperatura-humedad muy alta/baja: se activan el sensor de temperatura y humedad, respectivamente.

-Accesos a salas: se introducen los hechos de accesos y salidas de salas (con diferentes niveles de acceso).

-Ataque terrorista: se dispara el sensor de metales de la entrada.

REGLAS

Regla de temperatura: al detectar una temperatura superior o inferior al límite establecido (25 y 20 grados) se activan las medidas de regulación de temperatura oportunas para cada caso, cambiando la temperatura poco a poco hasta estar dentro del rango ideal.

Regla de humedad: similarmente al caso anterior, si se detecta una humedad anormal, se subirá o bajará hasta entrar en el rango ideal (40-60%).

Regla de incendios: al detectar humo en una sala se disparan las contramedidas: en caso de ser una sala catalogada como sensible se sofocará el incendio mediante gas y en caso contrario se utilizará agua para apagar el incendio. En cualquiera de los casos el edificio será desalojado, reduciendo la ocupación de todas las salas a 0.

Regla de detección de metales: cuando se activa el sensor de la entrada, se inicia un protocolo de evacuación del edificio, por un posible caso de amenaza terrorista.

Regla de control-acceso: cuando un usuario intenta entrar en una zona, primero comprueba que haya espacio en la sala y, en caso de haber espacio, comprueba que el usuario tenga al menos el nivel de acceso mínimo para acceder a dicha sala. Si se permite la entrada, aumenta en uno la ocupación actual para tenerlo en cuenta en otros intentos de acceso.

EJEMPLO DE FUNCIONAMIENTO

Para explicar el funcionamiento del sistema, explicaremos el script python paso a paso:

```python
from clips import Environment
from time import sleep

# Crear el entorno CLIPS
env = Environment()

# Cargar las reglas
env.load("templatesrules2.clp")

# Cargar los hechos
def cargar_hechos():
    env.load("hechos.clp")
    env.reset()
```

Comenzamos creando el entorno clips y cargando las reglas y plantillas, también creamos la función que usaremos para cargar los hechos base antes de cada caso de ejemplo.

```python
# Función para mostrar la ejecucion de las reglas de regulacion de temperatura
def demostracion_temperatura():
    cargar_hechos()
    print("Demostración de temperatura: ")
    env.assert_string('(sensor-temperatura (nombre "sensor-temp-1") (zona "servidores") (temperatura 27))')
    env.assert_string('(sensor-temperatura (nombre "sensor-temp-2") (zona "servidores") (temperatura 18))')
    env.run()
    print("\n")
    sleep(5)

# Función para mostrar la ejecucion de las reglas de regulacion de humedad
def demostracion_humedad():
    cargar_hechos()
    print("Demostración de humedad: ")
    env.assert_string('(sensor-humedad (nombre "sensor-hum-1") (zona "servidores") (humedad 20))')
    env.assert_string('(sensor-humedad (nombre "sensor-hum-2") (zona "servidores") (humedad 90))')
    env.run()
    print("\n")
    sleep(5)
```

```python
# Función para mostrar la ejecucion de las reglas de incendio
def demostracion_incendio():
    cargar_hechos()
    print("Demostración de incendio: ")
    env.assert_string('(sensor-humo (nombre "sensor-humo-1") (zona "servidores") (humo "Si"))')
    env.run()
    print("\n")
    sleep(5)

# Función para mostrar la ejecucion de las reglas de accesos
def demostracion_accesos():
    cargar_hechos()
    print("Demostración de accesos: ")
    env.assert_string('(sensor-acceso (nombre "sensor-acc-1") (zona "servidores") (acceso "Tomas"))')
    env.assert_string('(sensor-acceso (nombre "sensor-acc-2") (zona "servidores") (acceso "David"))')
    env.assert_string('(sensor-salida (nombre "sensor-sal-1") (zona "servidores"))')
    env.run()
    print("\n")
    sleep(5)

# Función para mostrar la ejecucion de las reglas de caso de bomba
def demostracion_bomba():
    cargar_hechos()
    print("Demostración de bomba: ")
    env.assert_string('(sensor-metales (nombre "sensor-metales-1") (zona "entrada"))')
    env.run()
    print("\n")
```

Definimos las funciones que usaremos para lanzar los casos de demostración. En cada regla se cargan de nuevo los hechos base para evitar conflictos entre los ejemplos, después se inserta el hecho(s) que disparan las reglas cargadas previamente y activamos el motor de inferencia.

```python
#Lanzar demostraciones
demostracion_temperatura()
demostracion_humedad()
demostracion_incendio()
demostracion_accesos()
demostracion_bomba()
```

Posteriormente lanzamos las funciones.

Ejemplo de temperatura:
En el caso de la temperatura, insertamos el siguiente hecho:

```python
env.assert_string('(sensor-temperatura (nombre "sensor-temp-1") (zona "servidores") (temperatura 27))')
```

Esto hace que se dispare la regla:

```
(defrule ajustar-temperatura-alta
   ?sensor <- (sensor-temperatura (nombre ?nombre) (zona ?zona) (temperatura ?temp))
   (test (>= ?temp 25))
   =>
   (printout t "Temperatura alta ("?temp") en el sensor " ?nombre ", de la zona " ?zona ". Bajando 1 grado." crlf)
   (modify ?sensor (temperatura (- ?temp 1))))
```

Ya que detecta que hay un sensor-temperatura con una temperatura mayor a 25, después se modifica en un grado dicho atributo. Este cambio hace que se siga cumpliendo la regla, por lo tanto se seguirá ejecutando hasta que sea menor que 25. Así se le muestra al usuario dicha acción:

```
Temperatura alta (27) en el sensor sensor-temp-1, de la zona servidores. Bajando 1 grado.
Temperatura alta (26) en el sensor sensor-temp-1, de la zona servidores. Bajando 1 grado.
Temperatura alta (25) en el sensor sensor-temp-1, de la zona servidores. Bajando 1 grado.
```

Ejemplo de acceso:
Para un acceso insertamos el hecho:

```
env.assert_string('(sensor-acceso (nombre "sensor-acc-1") (zona "servidores") (acceso "Tomas"))')
env.assert_string('(sensor-acceso (nombre "sensor-acc-2") (zona "servidores") (acceso "David"))')
```

Esto dispara la regla:

```
(defrule control-acceso
   ?sensor <- (sensor-acceso (nombre ?nombre) (zona ?zona) (acceso ?per))
   ?sala <- (zona (nombre ?zona) (acceso ?acc) (ocupacion-max ?ocpmax) (ocupacion-actual ?ocpact))
   ?usuario <- (usuario (nombre ?per) (acceso ?niv))
   =>
   (if (< ?ocpact ?ocpmax)
      then
      (if (>= ?niv ?acc)
         then
         (printout t ?per " ha entrado en " ?zona "." crlf)
         (modify ?sala (ocupacion-actual (+ ?ocpact 1)))
      else
         (printout t ?per " no tiene permiso para entrar en " ?zona "." crlf))
   else
      (printout t "La zona " ?zona " esta llena." crlf))
   (retract ?sensor))
```

Al detectar que una persona intenta entrar en una sala comprueba: primero, que haya espacio y, si lo hay, verifica que la persona tiene permiso para entrar, comparando el nivel de acceso de la zona en la base de hechos con el nivel que tiene la persona. En caso de permitir el acceso, aumenta en uno la ocupación-actual de dicha zona.

```
Demostración de accesos:
David ha entrado en servidores.
La zona servidores esta llena.
```

Aquí se puede ver cómo al entrar una persona en la sala de servidores que solo tenía espacio para una persona más no permite que entre nadie más.

CONCLUSIONES Y PROBLEMAS

Podemos concluir que CLIPS es una herramienta muy útil para el desarrollo de sistemas de control, gracias a su velocidad al aplicar las reglas y a su escalabilidad, ya que se pueden agregar nuevas reglas muy fácilmente, sin comprometer el funcionamiento del programa.

Para un caso real, una versión más avanzada de este sistema podría tener una aplicación útil, en caso de que los valores introducidos al CLIPS fuesen de un sensor real y si se dispusiese de actuadores para poder llevar a cabo las tareas como reducir la temperatura o la humedad, evacuar el edificio, etc.

A la hora de desarrollar el sistema, nos encontramos con varios problemas. Principalmente, había varias de nuestras reglas que generaban conflicto y hacían que el motor CLIPS entrase en un bucle. Entre otros problemas menores, nos costó que se disparasen varias reglas que usan varios condicionales a la vez .

Como posibles mejoras, pensamos que se podrían incluir los racks al sistema y, entre otras cosas, añadir más hechos de zonas y usuarios y que los sucesos ocurran de forma aleatoria para dar algo más de variedad y realismo.

BIBLIOGRAFÍA

-https://clipspy.readthedocs.io/en/latest/

-http://www.clipsrules.net/