Universidad Nacional de la Patagonia “San Juan Bosco”

Facultad de Ingeniería – Sede Trelew

Fundamentos teóricos de la informática

Trabajo Práctico Final

Simulación de ascensores con redes de Petri

Alumna:

- Moyano María Ivana

Índice

Introducción 3

Problemática 3

Estructura y funcionamiento del modelo de red de Petri propuesto 4

Implementación del software 5

Funcionamiento 6

Manual del usuario 8

Conclusión 9

Bibliografía 9

Introducción

Una Red de Petri es una herramienta gráfica que se adapta bien a un gran número de aplicaciones en la que el conocimiento de eventos y evoluciones simultáneas son importantes. Esta teoría fue Inventada por el alemán Karl Adam Petri en 1962. En su tesis doctoral “kommunication mit automaten” (Comunicación con autómatas), utilizada como método de descripción visual en la ayuda a la descripción del comportamiento de sistemas complejos.

Las Redes de Petri, son un formalismo para modelar, analizar, simular, controlar y evaluar el comportamiento de sistemas concurrentes. Han jugado un papel importante en el desarrollo de estos sistemas, ya que entre otras posibilidades pueden considerarse como un lenguaje formal y gráfico para su modelización. Además, las Redes de Petri tienen una sólida base matemática que permite el análisis cualitativo de las propiedades de tales sitemas como el interbloqueo y la acotabilidad. Sin embargo, el formalismo de las redes de Petri no ofrece una manera directa de representar características tales como cambios dinámicos de actividades, migración de tareas, modos de operación múltiples, etc., que son importantes en el diseño de sistemas concurrentes.

Por eso, para modelar sistemas concurrentes y distribuidos reales se utilizan redes de Petri.

Problemática

El comportamiento de un ascensor puede ser modelado como un sistema a eventos discretos (SED), en el que la evolución de sus estados en el espacio temporal depende de las relaciones estado-evento. Las Redes de Petri constituyen una de las herramientas matemáticas más sencilla e intuitiva para modelar un SED; gracias a su capacidad de representación gráfica, estas permiten visualizar y establecer a través de grafos las relaciones existentes entre los diferentes elementos que integran un sistema . Es por ello que se han escogido para representar el modelo dinámico del ascensor,

asegurando una operación eficiente del mismo.

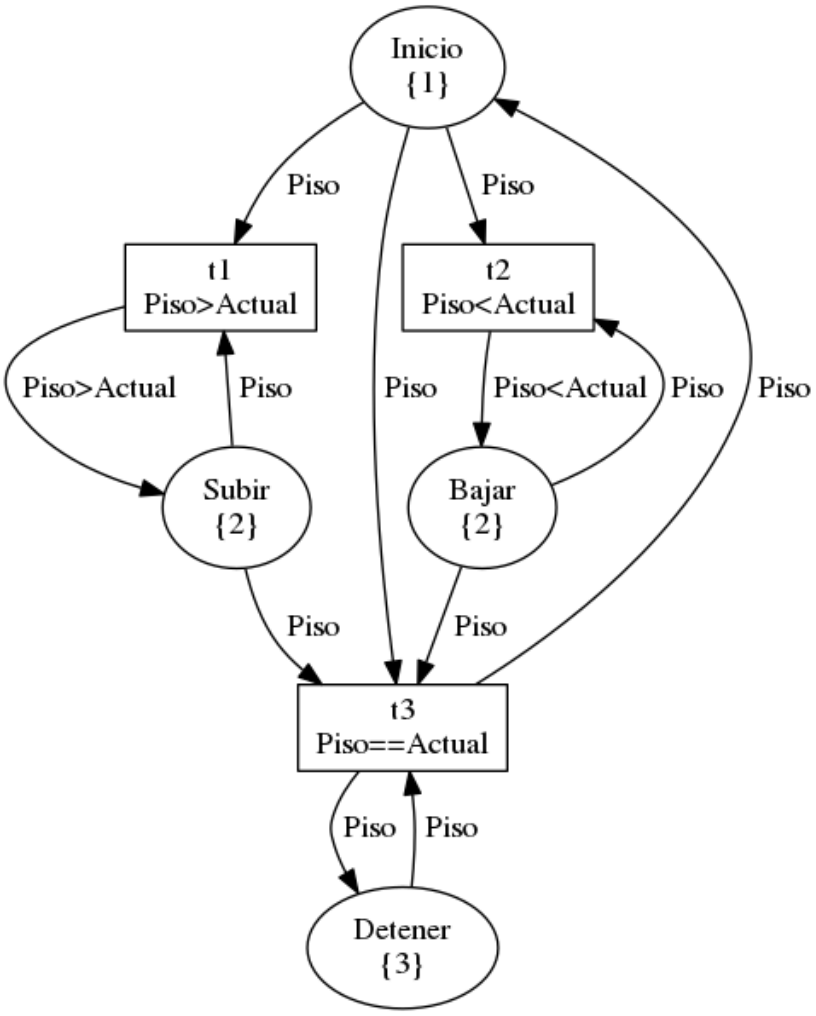
Debido a que las redes de Petri son adecuadas para capturar la concurrencia y la asincronía, se convierten en un modelo que puede ser adaptado para la simulación de sistemas de ascensores.

La problemática que será abordada como tema de investigación e implementación, será el modelado y simulación de personas que desean tomar un ascensor de la red de ascensores.

Estructura y funcionamiento del modelo de red de Petri propuesto

El modelo de la red de Petri de los ascensores está compuesto por los siguientes elementos:

1. **P**. Los lugares representan las celdas (o lugares) de conexión e intersección que pueden ser ocupados por los ascensores.
2. Los recursos (o tokens) representan a los ascensores en cada lugar o pisos del edificio.
3. **T**. Las transiciones modelan el flujo de los ascensores en el sistema entre celdas consecutivas. Tenemos 4 estados Inicio, Subir, Bajar y Detener, donde Inicio es cuando el elevador esta por usarse y nos envía la variable Piso según el piso elegido a una transición Piso > Actual ó Piso < Actual, el cual la primera irá hacía Subir y la segunda hacía Bajar todo esto hasta que Piso sea igual al Actual para entrar a el estado Detener.



Implementación del software

Para la implementación del modelo propuesto, se utilizó Python como lenguaje de programación y se utilizaron las siguientes librerías:

- **PyGame**. Es una librería para el desarrollo de videojuegos, y se utilizó para confeccionar la vista de la simulación.
- **Snakes**. Es una librería para el armado y modelado de redes de Petri, ofreciendo elementos necesarios para construir una red tales como: Lugares, transiciones, y el objeto PetriNet. Se utilizó como base para armar las redes de Petri usadas por los ascensores.

El software se encuentra estructurado en clases del modelo y de clases de la vista. Con respecto a las clases del modelo, se desarrollaron las siguientes clases principales:

- **Main.** La clase main se ejecuta al inicio de la simulación, y se encarga de crear la simulación, crear la vista, establecer relaciones entre ellos, esta clase también inicializa al reloj en pantalla.
- **Reloj.** Esta clase representa al reloj que indica el tiempo transcurrido desde el inicio de la simulación.
- **RedDeAscensores.** Esta clase extiende de Thread y mantiene una colección de ascensores.
- **Simulacion.** La clase Simulacion, contiene el comportamiento general relativo a la misma. Esta clase, es la responsable de inicializar la red de ascensores, crear a las personas y colocarlas en un piso determinado
- **Ascensor.** La clase Ascensor posee los métodos del ascensor.
- **Persona**. La clase Persona posee los métodos de la persona.

En cuanto a las clases que conforman la vista se encuentran:

- **Vista.** Esta clase es el canvas utilizado para el dibujado de los objetos en el escenario y utiliza PyGame para inicializar la vista y actualizar los elementos en la pantalla.
  Al ser instanciada, la simulación invoca al método actualizar() de vista, que mantiene a ésta dentro de un bucle que refresca constantemente las posiciones de los ascensores y de las personas.
- **SpritePersona.** Esta clase hereda de pygame.sprite.Sprite, que es una clase requerida por Pygame, y permite obtener la imagen que representa a la persona en pantalla, y utilizarlo para actualizar la posición del elemento.
- **SpriteAscensor.** Esta clase hereda de pygame.sprite.Sprite, que es una clase requerida por Pygame, y permite obtener la imagen que representa al ascensor en pantalla, y utilizarlo para actualizar la posición del elemento.

Funcionamiento

El funcionamiento general de la simulación es el siguiente: Cuando la simulación comienza, se instancia la clase Main, la que crea la simulación, la vista, el reloj, establece

relaciones entre ellas.

```
def iniciar(self):
    """ Este metodo instancia la simulacion, la v
    # Se inicializan todas las constantes de pyga
    pygame.init()
    self.imprimirFecha()

    simul=Simulacion()
    #simul.comenzar()
    reloj= Reloj()
    simul.setReloj(reloj)

    reloj.setDaemon(True)
    reloj.start()
    v=Vista(simul)
    v.actualizar(simul)
    v.setReloj(reloj)
```

La simulación crea a las personas y a la red de Ascensores. Los cuales comienzan a ejecutarse.

```
def actualizar(self,simul):
    logging.debug("ACTUALIZO LA VISTA")
    background_image = load_image('data/fondo2.jpg')
    self.rect = background_image.get_rect()
    self.rect.centerx = WIDTH
    self.rect.centery = HEIGHT
    clock = pygame.time.Clock()



    while True:
        tiempo = self.clock.tick(60)
        #keys = pygame.key.get_pressed()
        for eventos in pygame.event.get():
            if (eventos.type == QUIT)  :
                print "Saliendo de la simulacion!"
                print ""
                self.simulacion.cancelar()
                sys.exit(0)

        #debo matar o detener al hilo si temrina la simulacion
        self.screen.blit(background_image, (0, 0))
        simul.correr(tiempo)
        self.actualizarSprites(simul, tiempo)
        time.sleep(0.4)
        pygame.display.flip()
    return 0
```

Cuando una persona aparece en un piso del edificio, el ascensor más cercano y el

que no se encuentra ocupado por otra persona, o que se encuentre en moviemiento será el que atenderá a la persona y la trasladará a un piso detemrinado.

```
class Simulacion(pyglet.event.EventDispatcher):
    """ """
    def __init__(self):
        super(Simulacion,self).__init__()
        logging.debug("creo la simulacion")
        self.finalizoSimulacion=False
        self.redAscensores=RedDeAscensores(self)
        self.threads=[]
        self.reloj=None
        self.personas=[]

        piso=random.randint(2,4)
        piso2=random.randint(2,4)
        imagen="data/persona.jpg"
        p=Personas(piso,imagen)
        imagen2="data/persona2.jpg"
        p2=Personas(piso2,imagen2)
        self.personas.append(p)
        self.personas.append(p2)
```

Manual del usuario

Para iniciar la simulación se debe acceder a la carpeta y ejecutar el siguiente comando:

$ python final.py

Conclusión

Se ha comprobado la utilidad de las Redes de Petri como herramienta para modelar sistemas a eventos discretos, que pueden ser materializados en forma sencilla por medio de un lenguaje de programación.

Podemos concluir diciendo de que las Redes de Petri son una alternativa de modelado de sistemas, aplicados principalmente hacia el control y proceso, por su facilidad de manejo en el problema de la sincronización de procesos.

Bibliografía

- https://www.pygame.org/docs/
- http://erevistas.saber.ula.ve/index.php/cienciaeingenieria/article/view/4574/4350
- https://www.pygame.org/docs/
- http://repositorio.utp.edu.co/dspace/bitstream/handle/11059/1083/6298M912.pdf;jsessionid=6A650AE4AA7C4A2FFCB0B0DC7FB71561?sequence=1
- https://es.wikipedia.org/wiki/Red_de_Petri
- http://www.monografias.com/trabajos14/redesdepetri/redesdepetri.shtml
- https://www.uaeh.edu.mx/scige/boletin/icbi/n1/e4.html