PRÁCTICA 1 YASMIN

Roger Fernández Enríquez

Junio de 2024

Índice

1. Solución ejercicios **1**
1.1. Ejercicio 1 1
1.2. Ejercicio 2 2
1.3. Ejercicio 3 5
1.4. Ejercicio 4 6

1. Solución ejercicios

1.1. Ejercicio 1

Enunciado: Revisa y transcribe el ejemplo básico describiendo cada uno de los elementos principales con tus palabras.

El script yasmin_demo.py implementa la construcción de una máquina de estados finitos en la que existen sólo los estados BAR y FOO.

Se definen los estados de manera abstracta (clases BarState y FooState) heredando de la clase State. Cada clase de estado contiene una función execute que define la acción a realizar cuando se está en dicho estado. Dicho de otro modo, execute devuelve un resultado que indica las transición al siguiente estado o el fin de la máquina de estados (en el caso del script el fin de la máquina de estados es outcome4)

En la clase abstracta DemoNode se crea la máquina de estados finita, se instancian los estados de la misma (se instancian objetos de las clases BarState y FooState) y se definen las transiciones existentes entre los estados. También se visualiza la máquina de estados

en el visualizador de Yasmin,

Finalmente en el main se instancia un objeto de la clase DemoNode creando así una máquina de estados finita, en este caso con los estados BAR y FOO.

1.2. Ejercicio 2

Enunciado: Modifica el ejemplo anterior para que tenga un estado más.

El código propuesto es el siguiente (el script tiene el nombre modificacion_yasmin_demo.py):

```
import time
import rclpy
from simple_node import Node
from yasmin import State
from yasmin import Blackboard
from yasmin import StateMachine
from yasmin_viewer import YasminViewerPub


# define state Foo
class FooState(State):
    def __init__(self) -> None:
        super().__init__(["outcome1", "outcome2"])
        self.counter = 0
    def execute(self, blackboard: Blackboard) -> str:
        print("Executing state FOO")
        time.sleep(3)

        if self.counter >0:
            print(blackboard.bar_str)

        if self.counter < 3:
            self.counter += 1
            blackboard.foo_str = f"Counter: {self.counter}"
            return "outcome1"
        else:
            return "outcome2"

# define state Bar
class BarState(State):
    def __init__(self) -> None:
        super().__init__(outcomes=["outcome3","outcome4"])
        self.counter = 0
        self.iteraciones = 0
```

```
35          self.blackboard = ""
36
37      def execute(self, blackboard: Blackboard) -> str:
38          print("Executing state BAR")
39          time.sleep(3)
40
41          if self.blackboard == "" or self.blackboard != blackboard.foo_str:
42              self.counter=0
43              self.blackboard = blackboard.foo_str
44
45          self.iteraciones = self.iteraciones + 1
46
47          blackboard.bar_str = f"Se ha ejecutado {self.iteraciones} veces el
                estado BAR"
48
49          if self.counter <2:
50              self.counter = self.counter +1
51              #print(blackboard.foo_str)
52              return "outcome4"
53          else:
54              print(blackboard.foo_str)
55              return "outcome3"
56
57  # define state Nuevo
58  class NuevoState(State):
59      def __init__(self) -> None:
60          super().__init__(outcomes=["outcome5"])
61
62      def execute(self, blackboard: Blackboard) -> str:
63          print("Executing state NUEVO")
64          time.sleep(3)
65
66          #print(blackboard.foo_str)
67          return "outcome5"
68
69  class DemoNode(Node):
70
71      def __init__(self) -> None:
72          super().__init__("yasmin_node")
73
74          # create a state machine
75          sm = StateMachine(outcomes=["outcome6"])
76
77          # add states
78          sm.add_state("FOO", FooState(),
79                       transitions={"outcome1": "BAR",
```

```
80                                          "outcome2": "outcome6"})
81          sm.add_state("BAR", BarState(),
82                       transitions={"outcome3": "FOO",
83                                    "outcome4": "NUEVO"})
84          sm.add_state("NUEVO",NuevoState(),
85                       transitions={"outcome5": "BAR"})
86
87          # pub
88          YasminViewerPub(self, "YASMIN_DEMO_NUEVO", sm)
89
90          # execute
91          outcome = sm()
92          print(outcome)
93
94 # main
95 def main(args=None):
96
97      print("yasmin con estado NUEVO añadido")
98      rclpy.init(args=args)
99      node = DemoNode()
100     node.join_spin()
101     rclpy.shutdown()
102
103 if __name__ == "__main__":
104     main()
```

Lanzamos el nodo y lo visualizamos con yasmin viewer:

```
ros2 run yasmin_demo modificacion_yasmin_demo.py
```

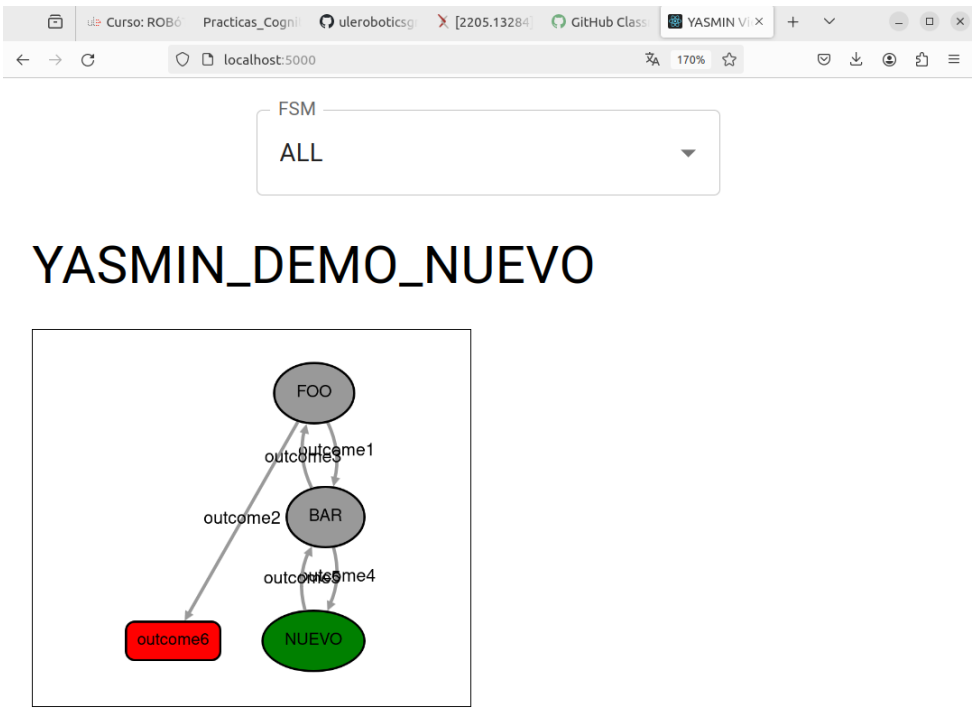

Figura 1: Visualización en Yasmin de la máquina de estados del ejercicio 2

1.3. Ejercicio 3

Enunciado: Define que es un Blackboard, para qué se utiliza en YASMIN. Indica puntos positivos y negativos.

Un Blackboard es es un patrón de diseño utilizado en la programación de máquinas de estados. Se trata de una estructura de datos compartida que permite la comunicación y el intercambio de información entre diferentes componentes del sistema. En el caso de Yasmin permite la comunicación entre estados. En el código en particular permite que los distintos estados compartan un string con información sobre el contador o el número de iteraciones.

Puntos positivos de usar Blackboards:

• La Blackboard permite que diferentes componentes (o estados, en el caso de máquinas de estados) compartan datos sin necesidad de establecer canales de comunicación

directos entre ellos. Esto simplifica la arquitectura del sistema.

• Es fácil agregar nuevos componentes o estados que utilicen el Blackboard sin necesidad de realizar cambios significativos en los componentes existentes.

• Los datos en el Blackboard permanecen disponibles a lo largo del ciclo de vida del sistema, permitiendo su uso por múltiples componentes o estados a lo largo del tiempo.

Puntos negativos de usar Blackboards:

• Complejidad en la gestión de los datos a medida que el número de los mismos aumenta.

• Problemas de concurrencia, puesto que el acceso al Blackboard puede dar lugar a condiciones de carrera y a inconsistencias de los datos.

• Complica la depuración de programas, ya que los errores pueden venir de cualquier componente que lea o escriba en el Blackboard.

1.4. Ejercicio 4

Enunciado: Modifica el ejemplo anterior para que cada estado publique un mensaje diferente sobre un topic de tu elección.

El código propuesto es el siguiente:

```
import time
import rclpy
from simple_node import Node
from yasmin import State
from yasmin import Blackboard
from yasmin import StateMachine
from yasmin_viewer import YasminViewerPub
from rclpy.node import Node as Node2
from std_msgs.msg import String

# define state Foo
class FooState(State):
    def __init__(self) -> None:
        super().__init__(["outcome1", "outcome2"])
        self.counter = 0
```

```
16      def execute(self, blackboard: Blackboard) -> str:
17          print("Executing state FOO")
18          time.sleep(3)
19          if self.counter == 0:
20              blackboard.publisher = MinimalPublisher()
21              blackboard.publisher.publish_message("Dentro del estado FOO")
22          if self.counter >0:
23              print(blackboard.bar_str)
24              blackboard.publisher.publish_message("Dentro del estado FOO")
25              #minimal_publisher.publish_message("Ejecutandose FOO")
26
27          if self.counter < 3:
28              self.counter += 1
29              blackboard.foo_str = f"Counter: {self.counter}"
30              return "outcome1"
31          else:
32              return "outcome2"
33
34  # define state Bar
35  class BarState(State):
36      def __init__(self) -> None:
37          super().__init__(outcomes=["outcome3","outcome4"])
38          self.counter = 0
39          self.iteraciones = 0
40          self.blackboard = ""
41
42      def execute(self, blackboard: Blackboard) -> str:
43          print("Executing state BAR")
44          blackboard.publisher.publish_message("Dentro del estado BAR")
45          time.sleep(3)
46
47          if self.blackboard == "" or self.blackboard != blackboard.foo_str:
48              self.counter=0
49              self.blackboard = blackboard.foo_str
50
51          self.iteraciones = self.iteraciones + 1
52
53          blackboard.bar_str = f"Se ha ejecutado {self.iteraciones} veces el
                estado BAR"
54
55          if self.counter <2:
56              self.counter = self.counter +1
57              #print(blackboard.foo_str)
58              return "outcome4"
59          else:
60              print(blackboard.foo_str)
```

```
61              return "outcome3"
62
63  # define state Nuevo
64  class NuevoState(State):
65      def __init__(self) -> None:
66          super().__init__(outcomes=["outcome5"])
67
68      def execute(self, blackboard: Blackboard) -> str:
69          print("Executing state NUEVO")
70          blackboard.publisher.publish_message("Dentro del estado NUEVO")
71          time.sleep(3)
72          #print(blackboard.foo_str)
73          return "outcome5"
74
75  class MinimalPublisher(Node2):
76      def __init__(self):
77          super().__init__('nodo_topico_yasmin')
78          self.publisher_ = self.create_publisher(String, 'topico_yasmin',
                10)
79          timer_period = 0.5  # seconds
80          self.timer = self.create_timer(timer_period, self.timer_callback)
81          self.i = 0
82
83      def timer_callback(self):
84          msg = String()
85          msg.data = 'Hello World: %d' % self.i
86          self.publisher_.publish(msg)
87          self.get_logger().info('Publishing: "%s"' % msg.data)
88          self.i += 1
89
90      def publish_message(self, message):
91          msg = String()
92          msg.data = message
93          self.publisher_.publish(msg)
94          #self.get_logger().info('Publicando: "%s"' % msg.data)
95
96  class DemoNode(Node):
97      def __init__(self) -> None:
98          super().__init__("yasmin_node")
99
100
101         #minimal_publisher = MinimalPublisher()
102
103         # create a state machine
104         sm = StateMachine(outcomes=["outcome6"])
105
```

```
106         # add states
107         sm.add_state("FOO", FooState(),
108                      transitions={"outcome1": "BAR",
109                                   "outcome2": "outcome6"})
110         sm.add_state("BAR", BarState(),
111                      transitions={"outcome3": "FOO",
112                                   "outcome4": "NUEVO"})
113         sm.add_state("NUEVO",NuevoState(),
114                      transitions={"outcome5": "BAR"})
115         # pub
116         YasminViewerPub(self, "YASMIN_DEMO_NUEVO", sm)
117
118         # execute
119         outcome = sm()
120         print(outcome)
121
122 # main
123 def main(args=None):
124
125     #minimal_publisher = MinimalPublisher()
126     print("yasmin con estado NUEVO añadido")
127     rclpy.init(args=args)
128     node = DemoNode()
129     node.join_spin()
130     rclpy.shutdown()
131
132 if __name__ == "__main__":
133     main()
```

Lanzamos el nodo y lo visualizamos con yasmin viewer:

```
ros2 run yasmin_demo topico_yasmin_demo.py
```

En otra terminal hacemos un ros2 topic echo al tópico de nombre /topico_yasmin, en el cual se están publicando mensajes:

```
ros2 echo /topico_yasmin
```

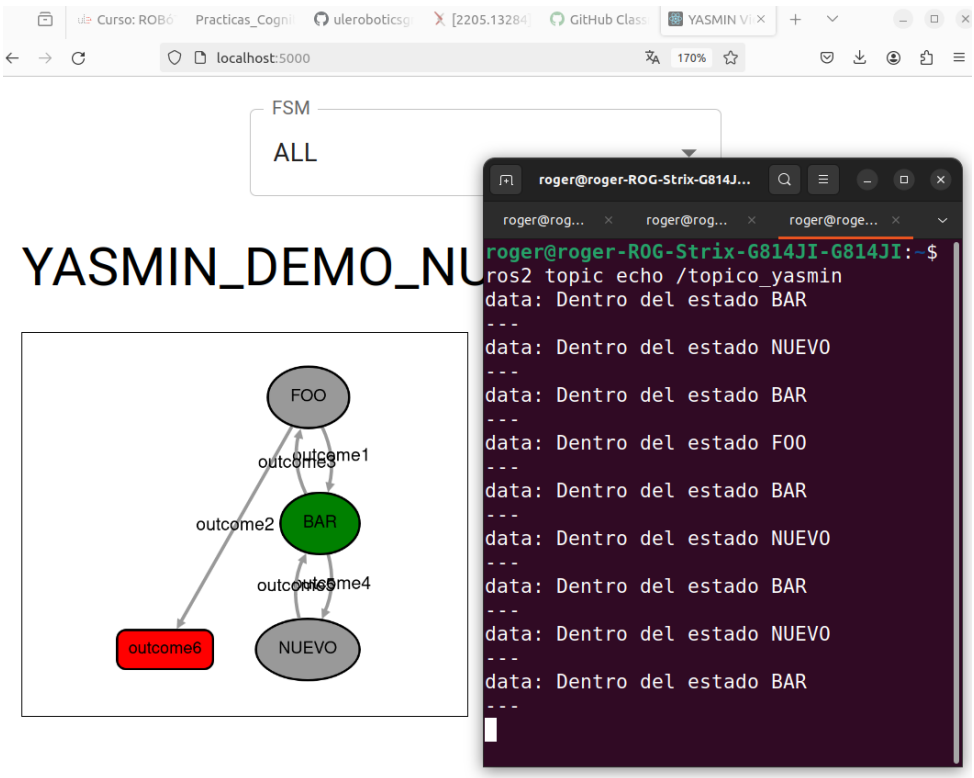

Figura 2: Visualización en Yasmin de la máquina de estados del ejercicio 4 y publicación en tópico topico_yasmin