6.4.3

Program a FlexBE State

Carlos Hernandez

Design your State implementation

Think the state design before start coding:

design principle: node = capability -> state (client)

```
# Conveyor belt control

# desired conveyor belt state
ConveyorBeltState state

---
bool success
```

```
# Conveyor belt state message
float64 power    # power of the belt (percentage, in +Y direction of belt frame)
```

Design your State implementation

Think the state design before start coding.

• What happens when the states becomes active?	set conveyor belt speed
• Outcomes?	‘succeeded’, ‘failed’
• Input_keys?	speed (float)
• Output_keys?	*-none-*
• Parameters?	stop (boolean)

Start coding

set_conveyor_power_state.py

```
#!/usr/bin/env python
import rospy
from flexbe_core import EventState


class SetConveyorPowerState(EventState):
```

Document the interface of your state

set_conveyor_power_state.py

```
#!/usr/bin/env python
import rospy
from flexbe_core import EventState


class SetConveyorPowerState(EventState):
    ''' Updates the speed of the conveyor belt through a service call
    -- stop            bool            If 'true' the state instance stops the
                                       conveyor belt, ignoring the speed inputkey
    ># speed           float           Speed for the conveyor belt
    <= succeeded                       The speed was successfully updated
    <= failed                          There was a problem setting the speed
    '''
```

init

set_conveyor_power_state.py

```
def __init__(self, stop):
        # Declare outcomes, input_keys, and output_keys by calling the super
        # constructor with the corresponding arguments.
        super(SetConveyorPowerState, self).__init__(outcomes = [’succeeded',
                'failed'], input_keys = ['speed'])


        # Store state parameter for later use.
        self._stop = bool(stop)


        # initialize service proxy
```

FlexBE Proxies

set_conveyor_power_state.py

```
#!/usr/bin/env python
import rospy
from flexbe_core import EventState
from flexbe_core.proxy import ProxyServiceCaller

from hrwros_gazebo.srv import SetConveyorControl, SetConveyorControlRequest

class SetConveyorPowerState(EventState):
    ''' Updates the speed of the conveyor belt through a service call
    -- stop              bool            If 'true' the state instance stops the
                                         conveyor belt, ignoring the speed inputkey
    ># speed             float           speed for the conveyor belt
    <= succeeded                         The speed was successfully updated
```

init

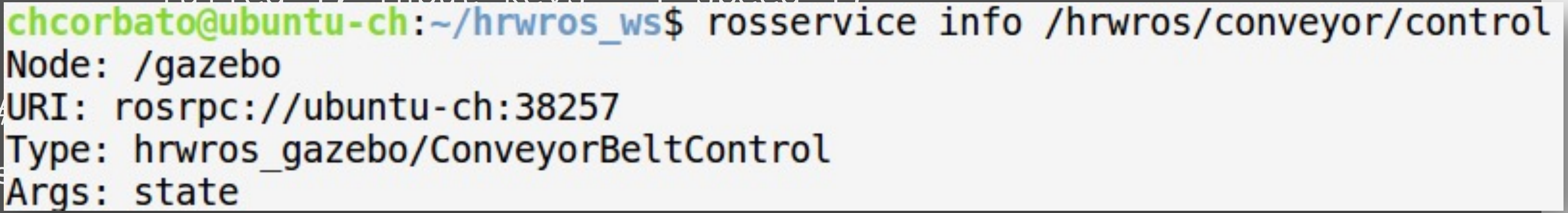
set_conveyor_power_state.py

```
def __init__(self, stop):
        # Declare outcomes, input_keys, and output_keys by calling the super
        # constructor with the corresponding arguments.
        super(SetConveyorPowerState, self).__init__(outcomes = ['continue',
              'failed'], input_keys = ['speed'])


        # Store state parameter for later use.
        self._stop = bool(stop)


        # initialize service proxy
        self._srv_topic = '/hrwros/conveyor/control’
        self._srv = ProxyServiceCaller({self._srv_topic: SetConveyorPower})
```

init

set_conveyor_power_state.py

```
def __init__(self, stop):
        # Declare outcomes, input_keys, and output_keys by calling the super
        # constructor with the corresponding arguments.

        super(SetConveyorPowerState, self).__init__(outcomes = ['continue',
                'failed'], input_keys = ['speed'])
```



```
chcorbato@ubuntu-ch:~/hrwros_ws$ rosservice info /hrwros/conveyor/control
Node: /gazebo
URI: rosrpc://ubuntu-ch:38257
Type: hrwros_gazebo/ConveyorBeltControl
Args: state
```

```
        #
        s

        # initialize service proxy
        self._srv_topic = '/hrwros/conveyor/control'
        self._srv = ProxyServiceCaller({self._srv_topic: SetConveyorPower})
```

on_enter

set_conveyor_power_state.py

```
def on_enter(self, userdata):
        self.speed = userdata.speed
        # create service request depending on activation parameter and userdata
        self._srv_req = ConveyorBeltControlRequest()

        if self._stop is True:
                self._srv_req.state.power = 0
        else:
                self._srv_req.state.power = self.speed
        try:
                self._srv_result = self._srv.call(self._srv_topic,self._srv_req)
                self._failed = False
        except Exception as e:
                rospy.logwarn(str(e))
                self._failed = True
```

execute

set_conveyor_power_state.py

```
def execute(self, userdata):
        # If no outcome is returned, the state will stay active.
        if self._failed:
                return 'failed'

        if self._srv_result.success is True:
                return 'succeeded'
        else:
                return 'failed'
```

on start, on_exit, on_stop

set_conveyor_power_state.py

```
        else:
                return 'failed'

def on_start(self):
        pass

def on_exit(self, userdata):
        pass

def on_stop(self):
        pass
```