Universidade do Minho
Escola de Engenharia
Departamento de Informática

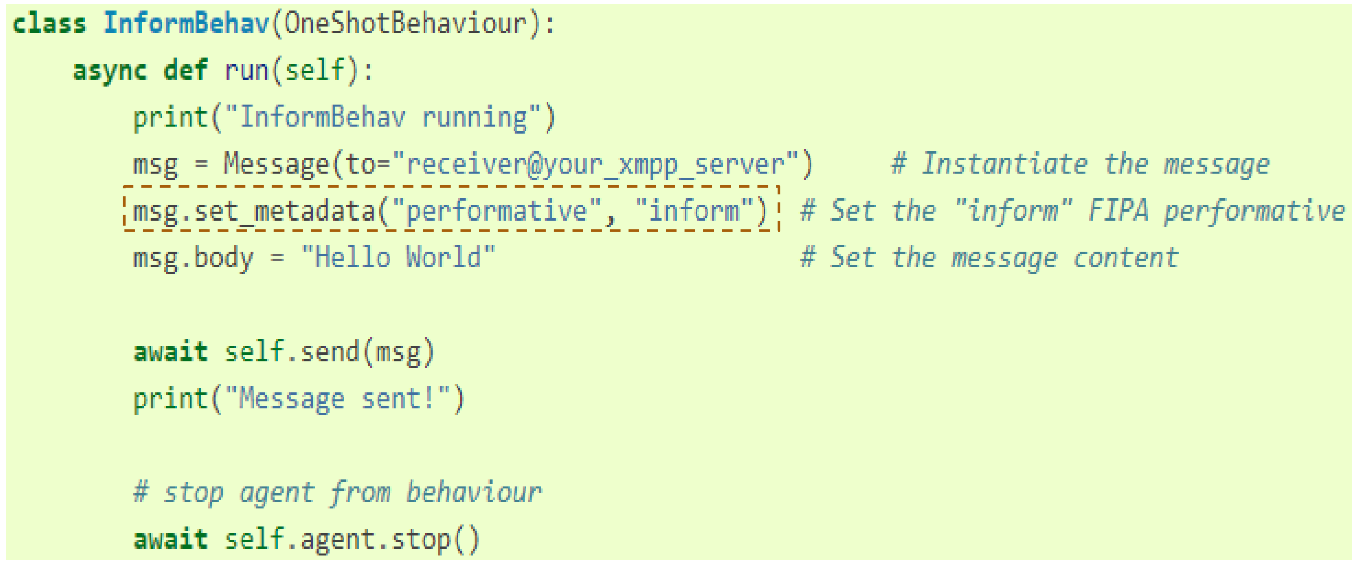
SPADE – Message Performatives and Serialized Message Content

Agentes e Sistemas Multiagente

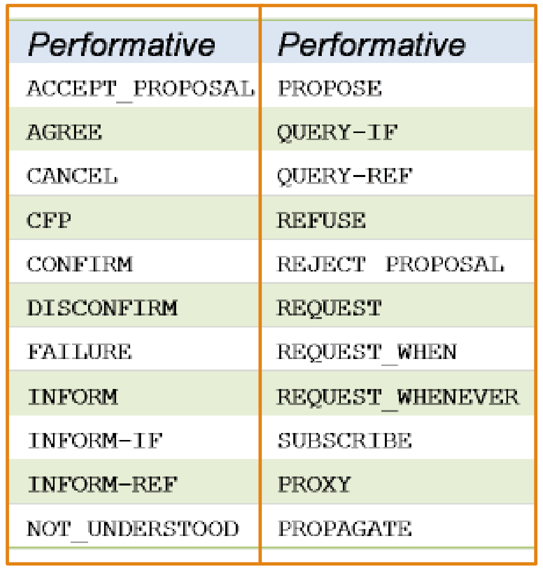
Pedro Oliveira, Paulo Novais

Synthetic Intelligence Lab

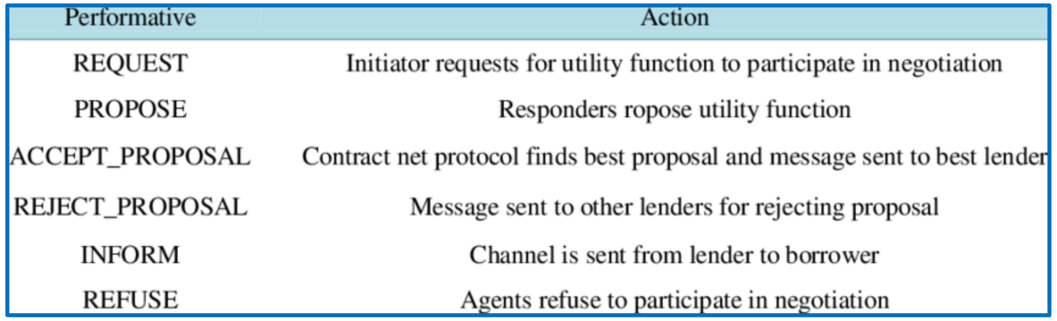
- Performatives:



```python
class InformBehav(OneShotBehaviour):
    async def run(self):
        print("InformBehav running")
        msg = Message(to="receiver@your_xmpp_server")     # Instantiate the message
        msg.set_metadata("performative", "inform")  # Set the "inform" FIPA performative
        msg.body = "Hello World"                    # Set the message content

        await self.send(msg)
        print("Message sent!")

        # stop agent from behaviour
        await self.agent.stop()
```



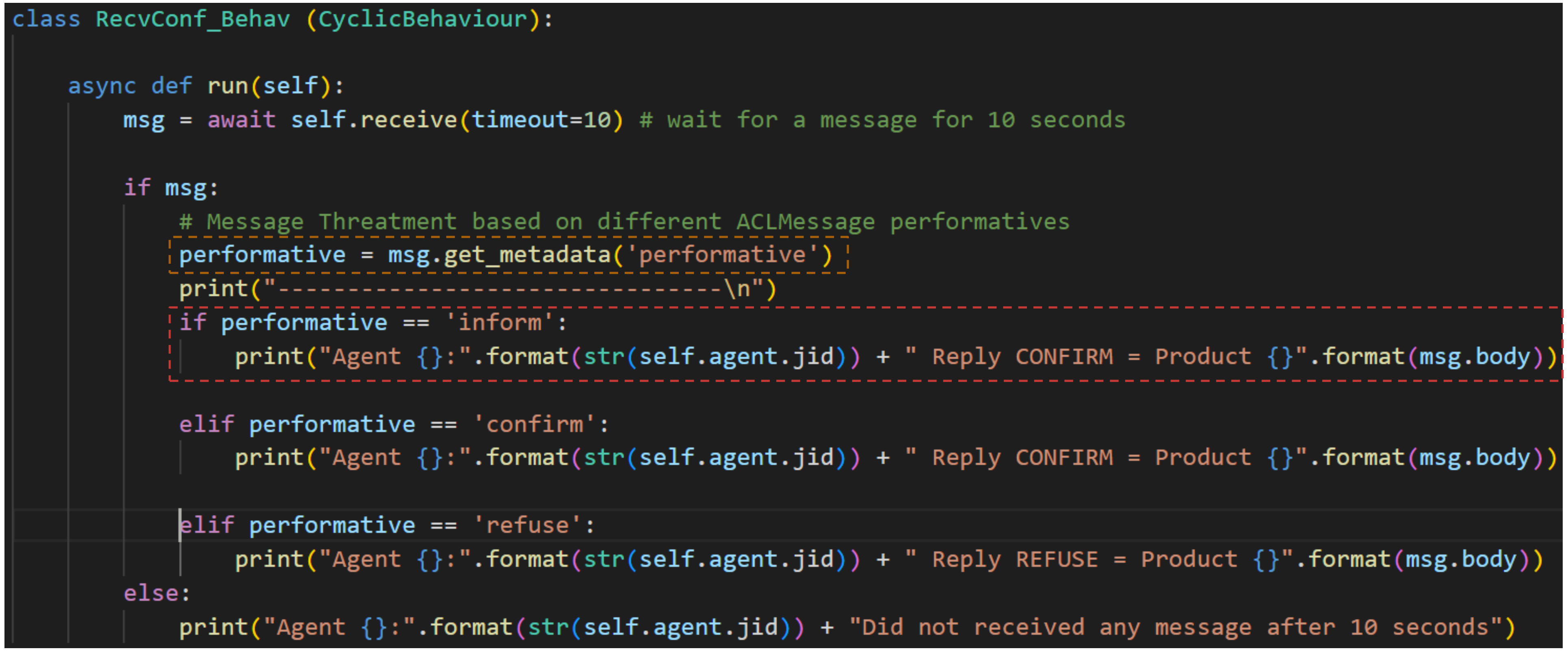
Performative	*Performative*
ACCEPT_PROPOSAL	PROPOSE
AGREE	QUERY-IF
CANCEL	QUERY-REF
CFP	REFUSE
CONFIRM	REJECT_PROPOSAL
DISCONFIRM	REQUEST
FAILURE	REQUEST_WHEN
INFORM	REQUEST_WHENEVER
INFORM-IF	SUBSCRIBE
INFORM-REF	PROXY
NOT_UNDERSTOOD	PROPAGATE



Performative	Action
REQUEST	Initiator requests for utility function to participate in negotiation
PROPOSE	Responders ropose utility function
ACCEPT_PROPOSAL	Contract net protocol finds best proposal and message sent to best lender
REJECT_PROPOSAL	Message sent to other lenders for rejecting proposal
INFORM	Channel is sent from lender to borrower
REFUSE	Agents refuse to participate in negotiation

ISLab
Synthetic Intelligence Lab

- Performatives:



```
class RecvConf_Behav (CyclicBehaviour):

    async def run(self):
        msg = await self.receive(timeout=10) # wait for a message for 10 seconds

        if msg:
            # Message Threatment based on different ACLMessage performatives
            performative = msg.get_metadata('performative')
            print("-------------------------------\n")
            if performative == 'inform':
                print("Agent {}:".format(str(self.agent.jid)) + " Reply CONFIRM = Product {}".format(msg.body))

            elif performative == 'confirm':
                print("Agent {}:".format(str(self.agent.jid)) + " Reply CONFIRM = Product {}".format(msg.body))

            elif performative == 'refuse':
                print("Agent {}:".format(str(self.agent.jid)) + " Reply REFUSE = Product {}".format(msg.body))
        else:
            print("Agent {}:".format(str(self.agent.jid)) + "Did not received any message after 10 seconds")
```

ISLab
Synthetic Intelligence Lab

- <u>Alternative</u>: Filter Messages from Message Pool

```python
class ReceiverAgent(Agent):
    class RecvBehav(OneShotBehaviour):
        async def run(self):
            print("RecvBehav running")

            msg = await self.receive(timeout=10) # wait for a message for 10 seconds
            if msg:
                print("Message received with content: {}".format(msg.body))
            else:
                print("Did not received any message after 10 seconds")

            # stop agent from behaviour
            await self.agent.stop()

    async def setup(self):
        print("ReceiverAgent started")
        b = self.RecvBehav()
        template = Template()
        template.set_metadata("performative", "inform")
        self.add_behaviour(b, template)
```

<u>RecvBehav only regards Messages with performative INFORM</u>

<u>Defined by Template() class incremented in add behaviou()</u>

ISLab
Synthetic Intelligence Lab

- If agents are to communicate in a way that makes sense for them, they must share the same language, vocabulary and protocols
- SPADE already supports a certain degree of commonality
- However, it is required to define your own vocabulary and semantics for the content of the messages exchanged between agents

- SPADE provides two ways to implement communication between agents:
  1. (Used until now) Using Strings to represent the message’s content
     - Convenient when the content of messages is atomic data, but not in the case of abstract concepts, objects or structured data
     - String needs to be parsed for the agent to understand
  2. Transmit serialized objects directly as the content of messages
     - Serialization: process of converting a data object into a series of bytes that saves the state of the object in an easily transmittable form
     - Convenient method for a local application where all agents are implemented in Python
     - **Require *jsonpickle* library to encode/decode objects**
     - **To install:** pip install –U jsonpickle

The Bank example:

- Two agents are created which implement the client and server roles for a bank with savings accounts

- The **BankServerAgent** class acts as a server and the **BankClientAgent** class acts as client

ISLab
Synthetic Intelligence Lab

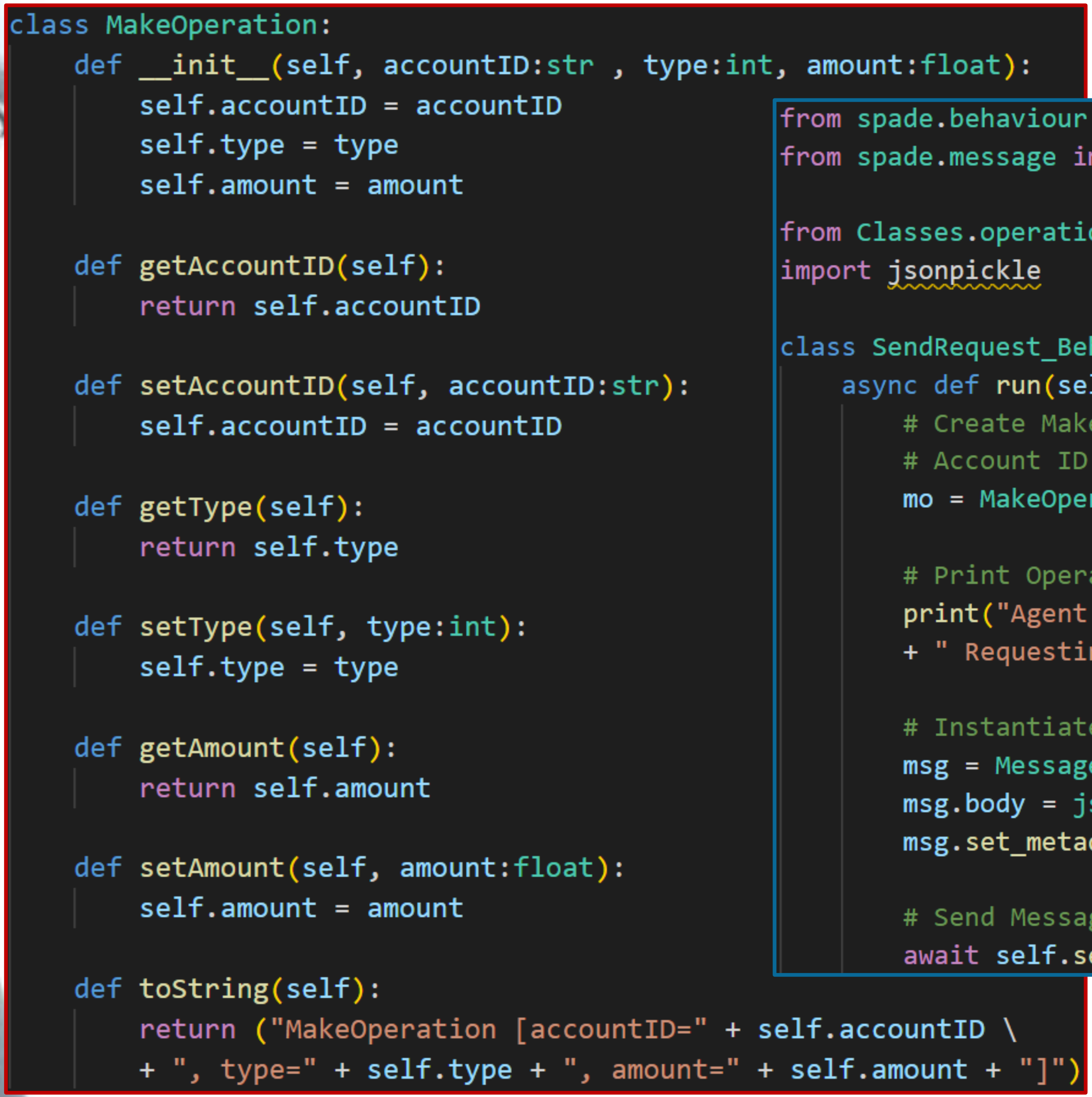
The Bank example:

- Client agent sends a REQUEST message to the server agent, following a simple protocol:
    - Make an Operation (**MakeOperation** class: action of making an operation such as deposit or withdrawl)

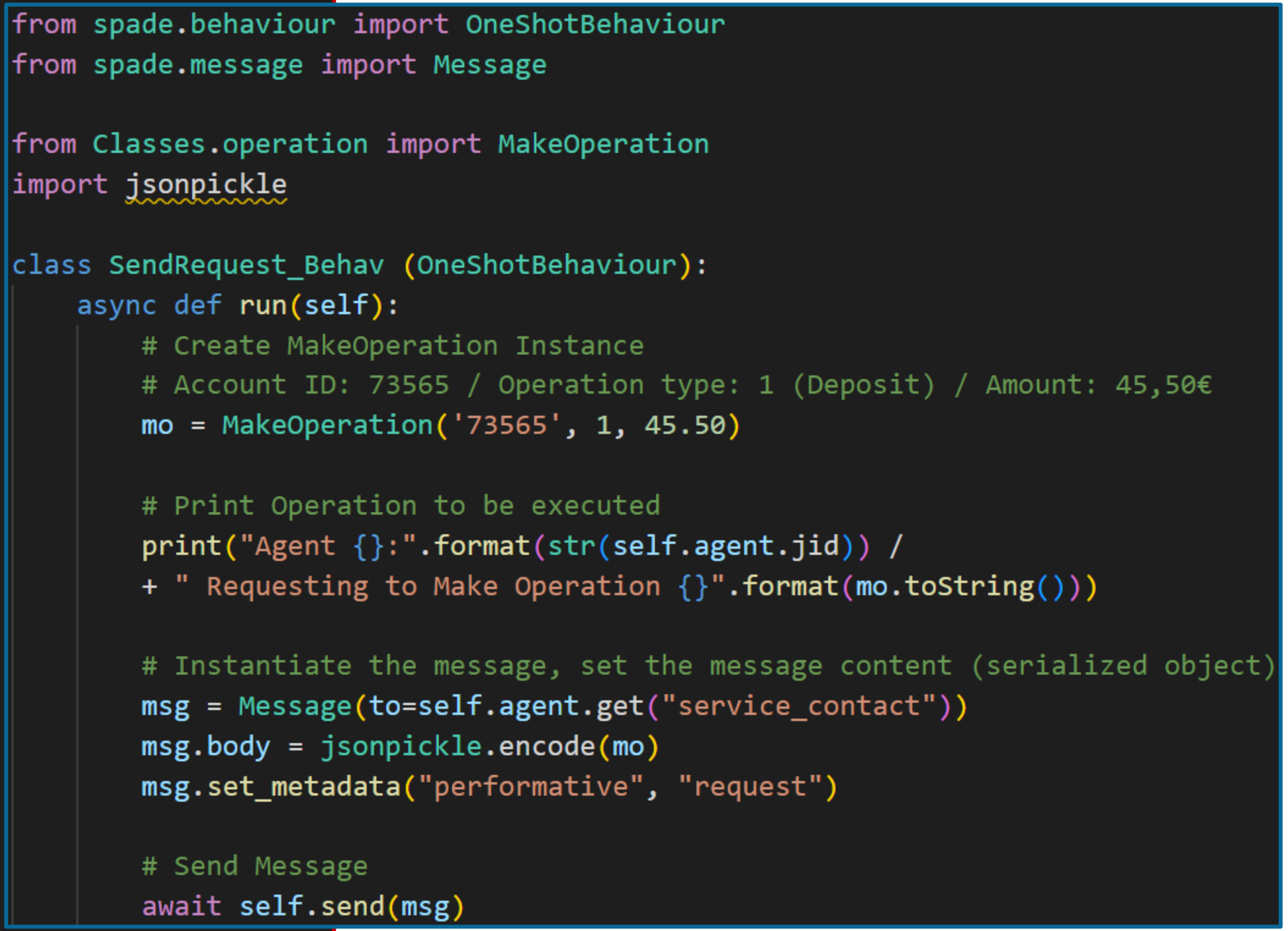
- The server agent responds either with:
    - an INFORM after processing the request
    - a NOT_UNDERSTOOD if it cannot decode the content of the message

ISLab
Synthetic Intelligence Lab

Sending Messages



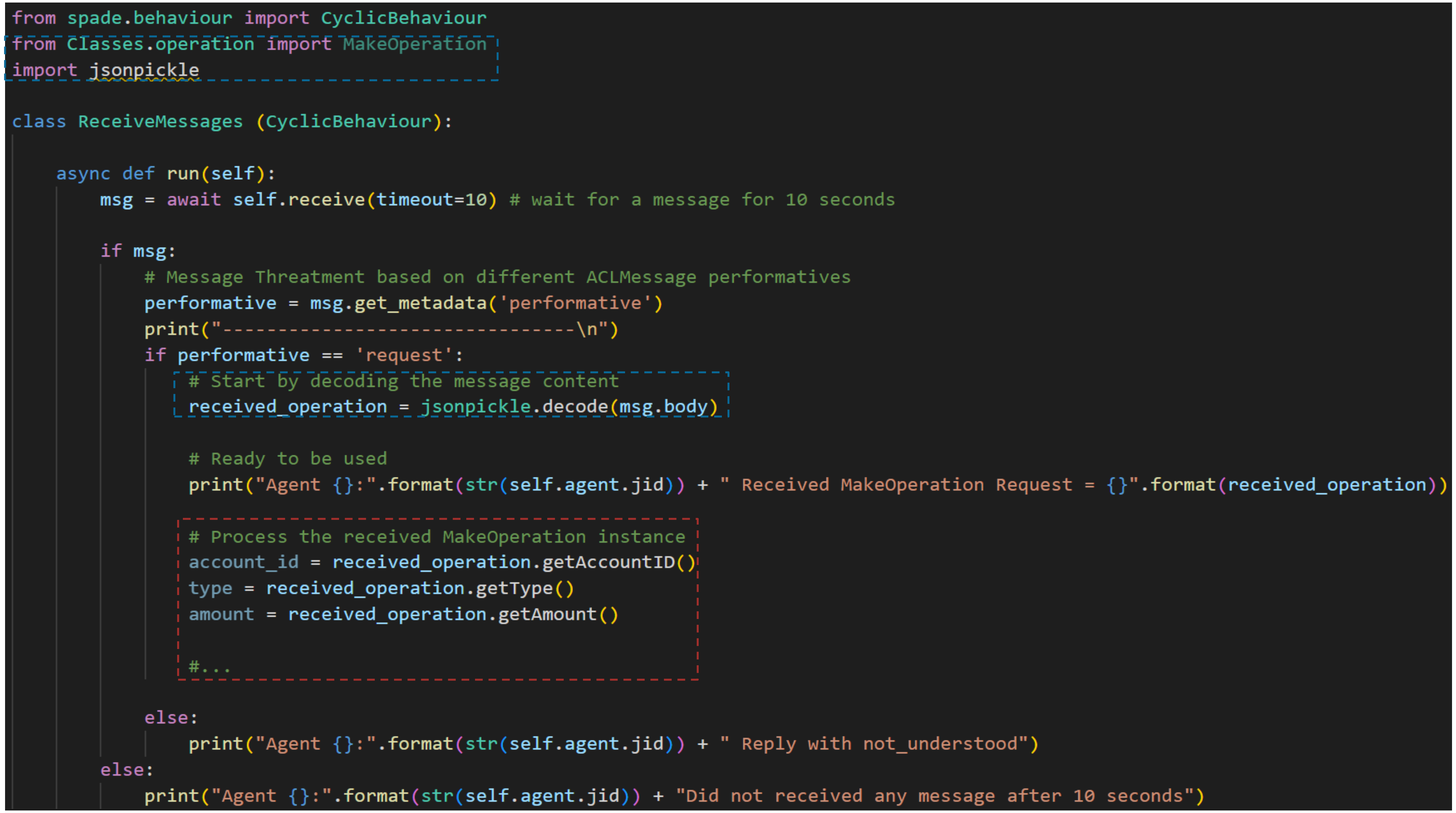
```python
class MakeOperation:
    def __init__(self, accountID:str , type:int, amount:float):
        self.accountID = accountID
        self.type = type
        self.amount = amount

    def getAccountID(self):
        return self.accountID

    def setAccountID(self, accountID:str):
        self.accountID = accountID

    def getType(self):
        return self.type

    def setType(self, type:int):
        self.type = type

    def getAmount(self):
        return self.amount

    def setAmount(self, amount:float):
        self.amount = amount

    def toString(self):
        return ("MakeOperation [accountID=" + self.accountID \
        + ", type=" + self.type + ", amount=" + self.amount + "]")
```



```python
from spade.behaviour import OneShotBehaviour
from spade.message import Message

from Classes.operation import MakeOperation
import jsonpickle

class SendRequest_Behav (OneShotBehaviour):
    async def run(self):
        # Create MakeOperation Instance
        # Account ID: 73565 / Operation type: 1 (Deposit) / Amount: 45,50€
        mo = MakeOperation('73565', 1, 45.50)

        # Print Operation to be executed
        print("Agent {}:".format(str(self.agent.jid)) /
        + " Requesting to Make Operation {}".format(mo.toString()))

        # Instantiate the message, set the message content (serialized object)
        msg = Message(to=self.agent.get("service_contact"))
        msg.body = jsonpickle.encode(mo)
        msg.set_metadata("performative", "request")

        # Send Message
        await self.send(msg)
```

ISLab
Synthetic Intelligence Lab

Receiving Messages



```python
from spade.behaviour import CyclicBehaviour
from Classes.operation import MakeOperation
import jsonpickle

class ReceiveMessages (CyclicBehaviour):

    async def run(self):
        msg = await self.receive(timeout=10) # wait for a message for 10 seconds

        if msg:
            # Message Threatment based on different ACLMessage performatives
            performative = msg.get_metadata('performative')
            print("--------------------------------\n")
            if performative == 'request':
                # Start by decoding the message content
                received_operation = jsonpickle.decode(msg.body)

                # Ready to be used
                print("Agent {}:".format(str(self.agent.jid)) + " Received MakeOperation Request = {}".format(received_operation))

                # Process the received MakeOperation instance
                account_id = received_operation.getAccountID()
                type = received_operation.getType()
                amount = received_operation.getAmount()

                #...

            else:
                print("Agent {}:".format(str(self.agent.jid)) + " Reply with not_understood")
        else:
            print("Agent {}:".format(str(self.agent.jid)) + "Did not received any message after 10 seconds")
```

Synthetic Intelligence Lab

References and Electronic Resources

- J. Palanca, A. Terrasa, V. Julian and C. Carrascosa, "SPADE 3: Supporting the New Generation of Multi-Agent Systems," in IEEE Access, vol. 8, pp. 182537-182549, 2020

Universidade do Minho
Escola de Engenharia
Departamento de Informática

SPADE – Message Performatives and Serialized Message Content

Agentes e Sistemas Multiagente

Pedro Oliveira, Paulo Novais