INTERNATIONAL SUMMER SCHOOL ON INDUSTRIAL AGENTS 2024

STANDARDIZATION OF I4.0 SYSTEMS

Multi-agent programming with SPADE
Part I. Fundamentals

Oskar Casquero, Aintzane Armentia, Julen Cuadra, Ekaitz Hurtado

Bilbao, 24th June 2024

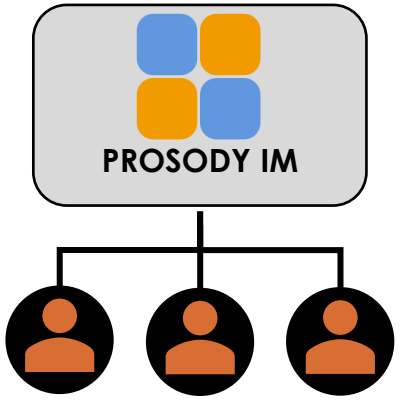
TABLE OF CONTENTS

RUNTIME ENVIRONMENT

OVERVIEW

- **What software resources are needed?**

- Multi-agent system platform: **SPADE**
  - Written in Python (benefits of its large ecosystem of IA libraries)
  - Message Transport System based on XMPP.
    - Agents do not talk directly to one another.
    - Not compliant with FIPA Message Transport standards
      http://www.fipa.org/repository/standardspecs.html



- XMPP server: Prosody IM
  - Hosted in VMware virtual machine with Ubuntu as guest

- Every agent in SPADE has a unique XMPP address
  <JID or Jabber ID>@<domain_name> → myAgent@ubuntu.min.vm

- Local XMPP server bound to domain name using **hosts** file
  - Linux: **/etc/hosts**
  - Windows: **C:\Windows\System32\drivers\etc\hosts**

```
# localhost name resolution is har
#   127.0.0.1       localhost
#   ::1             localhost
192.168.0.200       ubuntu.min.vm
```

MY FIRST SPADE AGENT

OVERVIEW+EXERCISE

- A SPADE agent is an instance of a Python class
  - The class is inherited from a SPADE class: **spade.agent.Agent**
- Agent common methods
  - **setup()**: to add initialization code
- SPADE runs in an asynchronous loop: **spade.run(main())**
  - SPADE is based on nonblocking methods called **courutines**
  - SPADE methods defined with "async def": **async def setup()**
  - SPADE methods are called with "await": **await agent.start()**

- Let's create our first SPADE agent!
- Run *1_my_first_agent.py*
- Agent that simply exists
  - Its only task is to initialization

COURUTINES

OVERVIEW+EXERCISE

- **Courutine**: nonblocking function that let other functions run while it waits for some call to complete.
  - Courutines deal with concurrency, not parallelism.
- How will Python know that some call is awaited? The courutine notifies the **event loop** with the **await** keyword so it can execute another function.
- WARNING! A blocking call in an async function will still block other functions from running, because they all share the same thread.
  - Use async.sleep() instead of time.sleep() since the async version is non-blocking.

- Run *main_courutine.py*
- **asyncio.create_task** submits a coroutine to the event loop.
- Thus, when the event loop starts running on the first await, both task1 and task2 are ready to run.

AGENT WITH BEHAVIOUR

OVERVIEW

- Behaviour: pattern-based task executed by an agent
- Different type of behaviours in SPADE
  - Cyclic and Periodic behaviours: repetitive tasks
  - One-Shot and Time-Out behaviours: casual tasks
  - Finite State Machine behaviour
- A Behaviour is an instance of a Python class inherited from a SPADE class:
  - **spade.behaviour.CyclicBehaviour**
  - **spade.behaviour.OneShotBehaviour**
  - **spade.behaviour.FSMBehaviour**

AGENT WITH BEHAVIOUR

EXERCISE

- Behaviour common methods
  - *on_start()*: executed when the behaviour begins
  - *run()*: where the core of the logic is executed
  - *on_end()*: executed when the behaviour is done or killed

- How to add a behaviour to an agent?
  - Create an instance

    mb = MyBehaviour()
  - Add the behaviour to the queue of behaviours of the agent

    agent.add_behaviour(mb)

- Run *2_agent_with_behaviour.py*

AGENT WITH SEVERAL BEHAVIOURS

OVERVIEW+EXERCISE

- Now let's create a more functional agent
  - 2nd behaviour added to the agent
  - This behaviour initializes an attribute of the agent.
  - The first behaviour is able to use this attribute.

- Run *3_agent_with_several_behaviours.py*

AGENT WITH CONCURRENT BEHAVIOURS

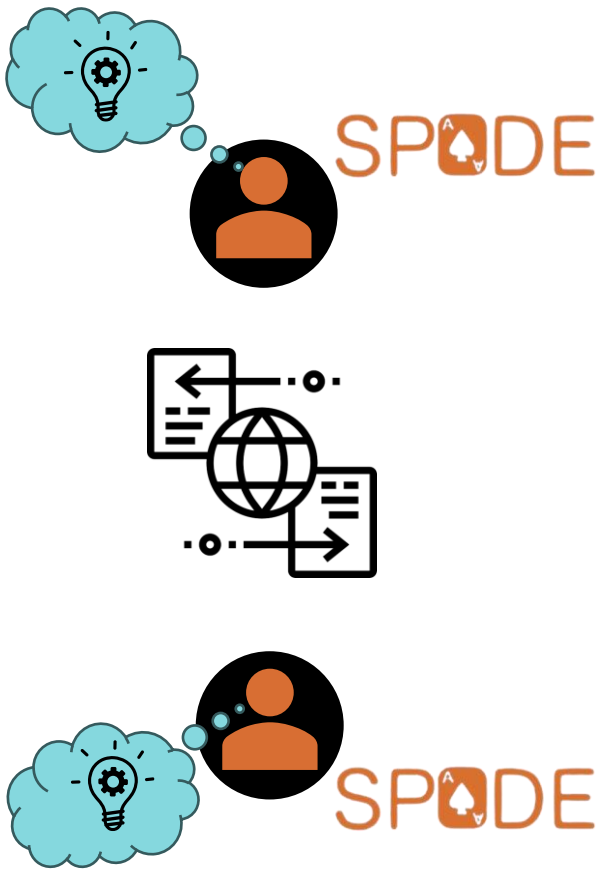
OVERVIEW+EXERCISE

- Agent with multiple, concurrent behaviours.
- The BootingBehaviour is replaced by a another RunningBehaviour that waits 5s in every cycle.

- Run *4_agent_with_concurrent_behaviours.py*
- Is the result as expected?
- Why?
- How can it be solved?

COMMUNICATION BETWEEN AGENTS

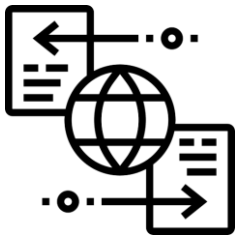
OVERVIEW



- SPADE agents can communicate with each other by exchanging messages
- Structure of a message (not FIPA ACL standard compliant)
  - **to**: receiver agent jid
  - **sender:** sender agent jid
  - **body**: content of the message.
  - **thread**: identifier of the conversation (equivalent to FIPA ACL "conversation-id" message parameter)
  - **metadata**: a (key, value) dictionary of strings for additional data, such as FIPA ACL message parameters (performative, ontology…)
- sending and receiving message functions are asynchronous
  - **await** self.send()
  - **await** self.receive()

COMMUNICATION BETWEEN AGENTS

EXERCISE



- Example: communication between two agents.
- Sender agent sends message in behaviour's **on_start()**
- FIPA ACL attributes are added to the message.
  - performative: "inform"
  - ontology: "myOntology"
- Receiver agent waits for messages in behaviour's **run()**

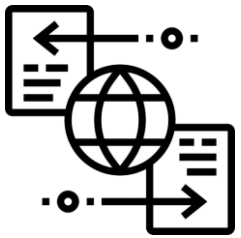
- First, the receiver agent should be executed
  5_agent_comms_recver.py 1
- Then, the sender agent can be executed
  5_agent_comms_sender.py 2

- What is the suffix added to the senders JID?
  - https://wiki.xmpp.org/web/XMPP_Resources

DISPATCHING MESSAGES TO BEHAVIOURS

OVERVIEW+EXERCISE

- **Templates** are used to automatically dispatch received messages to the behaviours that are waiting for them.
- Templates support boolean operators to create complex templates.
- Individual messages can be manually matched against templates.
- WARNING! XMPP resources must be considered in templates



- Manually match the received msg against a template in receiver.
- Send 2 msgs (accept and reject performatives) from the receiver.
- Apply a template in the senders behaviour to match accept msg.
- Let's see this in action, run:

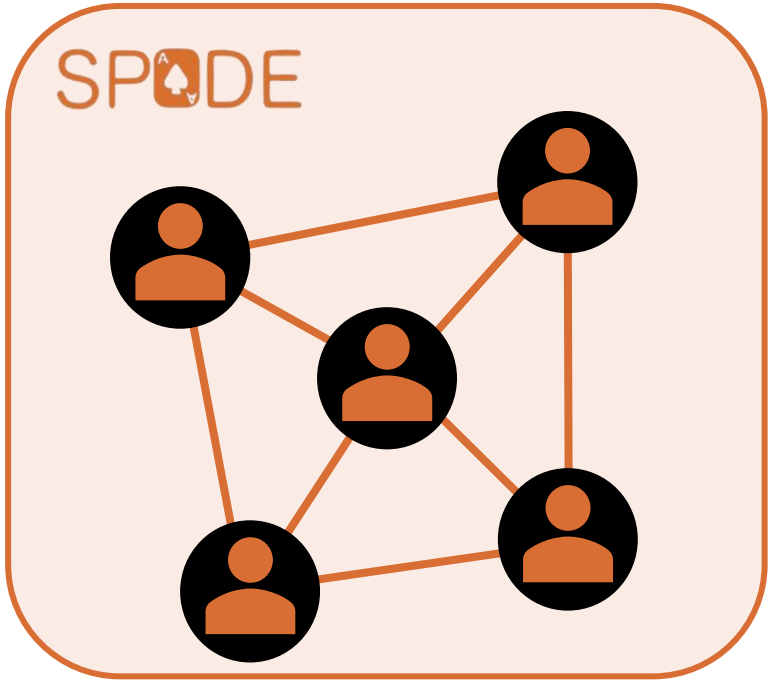
  6_agent_comms_recver_templates.py 1

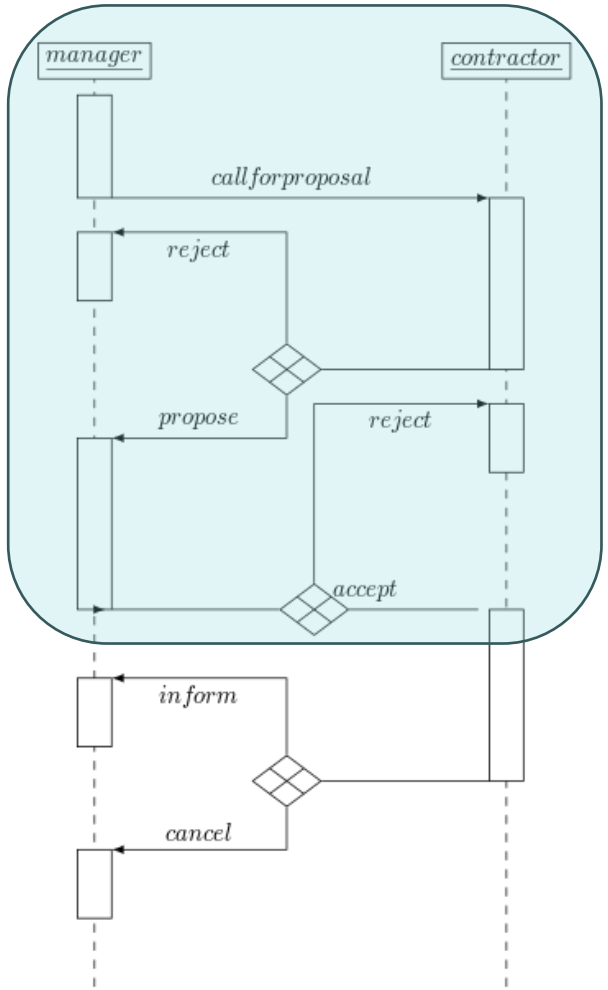
  6_agent_comms_sender_templates.py 2

- Does the receiver agent get the two messages?

CONTRACT NET PROTOCOL

EXERCISE





https://en.wikipedia.org/wiki/Contract_Net_Protocol

PROGRAM THIS PART:

1. The protocol is initialized by the manager, who sends a *call-for-proposals* to the contractors

2. The contractors can send either a *proposal* if they are interested or a *reject* if they are not.

3. The manager chooses among the proposals the one that suits it best and sends to the corresponding contractor an *accept*. It sends a *reject* to the other contractors to inform them of its decision.