University of St.Gallen

Institute of Computer Science

Web-based Autonomous Systems, FS2023
Danai Vachtsevanou, Andrei Ciortea
{firstname}.{lastname}@unisg.ch

Exercise 6: Interacting Agents on the Web

Deadline: **April 4, 2023; 23:59 CET**

In this exercise, you will gain hands-on experience in programming the environment of a Multi-Agent System, and the interactions of BDI agents with each other and with the environment. You will implement a Multi-Agent System as a JaCaMo application where:

1.) agents interact with their environment based on W3C Web of Things (WoT) Thing Descriptions[1];

2.) agents coordinate with each other by communicating with KQML performatives and based on the Contract Net Protocol.

① Task 1 (10 points) Agent-to-Environment and Agent-to-Agent Interaction

Your task is to is to complete a JaCaMo application where autonomous agents are capable of coordinating with each other by interacting with the environment and with one another. The design objective of the agents is to assist a user through their daily activities, in a similar manner the personal assistant agent did during Exercise 5: The personal assistant will strive for waking up the user in the case of an upcoming event, but this time in a multi-agent setting, and using devices and services that expose Web APIs.

Overall, the MAS features the following agents:

- A personal assistant agent (`personal_assistant.asl`[2]) whose design objective is to assist its user through their daily activities. As last time, such is the case when the assistant believes that the user is asleep, and that the user has an upcoming event to attend. Considering its options, the assistant strives to wake up the user, this time, by *coordinating with other agents* for raising the blinds, turning on the lights, or asking for help from the user's friends through a message service.
- A calendar manager agent (`calendar_manager.asl`[3]) whose objective is to manage the events of a user by *interacting with calendar service WoT Thing*, and *inform other agents* in case of an upcoming event.
- A wristband manager agent (`wristband_manager.asl`[4]) whose objective is to monitor the activities of a user by *interacting with a wristband WoT Thing*, and *inform other agents*, e.g. about whether the user is asleep or awake.
- A blinds controller agent (`blinds_controller.asl`[5]) whose objective is to monitor and control the state of the blinds in a room *by interacting with a set of blinds WoT Thing.*

[1]Web of Things (WoT) Thing Description 1.1: https://www.w3.org/TR/wot-thing-description11/
[2]Personal assistant agent: https://github.com/HSG-WAS-SS23/exercise-6/blob/main/src/agt/personal_assistant.asl
[3]Calendar manager agent: https://github.com/HSG-WAS-SS23/exercise-6/blob/main/src/agt/calendar_manager.asl
[4]Wristband manager agent: https://github.com/HSG-WAS-SS23/exercise-6/blob/main/src/agt/wristband_manager.asl
[5]Blinds controller agent: https://github.com/HSG-WAS-SS23/exercise-6/blob/main/src/agt/blinds_controller.asl

- A lights controller agent (`lights_controller.asl`[6]) whose objective is to monitor and control the state of the lights in a room *by interacting with a set of lights WoT Thing*.

Additionally, the MAS features artifacts for enabling the agents to interact with their environment. Specifically, agents can interact with any device or service that is modelled as W3C WoT Thing through a `ThingArtifact` (see `ThingArtifact.java`[7]). A `ThingArtifact` offers two CArtAgO operations, `readProperty()` and `invokeAction()` for enabling agents to exploit TD Property Affordances and TD Action Affordances based on a Thing's Thing Description. Agents have access to the following Thing Descriptions (TDs):

- a TD of a calendar service Thing, available here[8], that offers a TD Property Affordance of type `was:ReadUpcomingEvent` for reading the upcoming events of a user.
- a TD of a wristband Thing, available here[9], that offers a TD Property Affordance of type `was:ReadOwnerState` for reading the state of the user.
- a TD of a set of blinds Thing, available here[10], that offers a TD Action Affordance of type `was:SetState` for setting the state of the blinds to "raised" or "lowered".
- a TD of a set of lights Thing, available here[11], that offers a TD Action Affordance of type `was:SetState` for setting the state of the lights to "on" or "off".

Finally, the MAS features a `DweetArtifact` (see `DweetArtifact.java`[12]) that enables agents to send messages based on the dweet.io API as documented here[13].

Task 1.1 (1.5 points): Agent-to-Environment interaction with a CArtAgO artifact Complete the implementation of the `DweetArtifact` that enables agents to send messages to the user's friends via the `dweet.io` API. Then, update the implementation of the personal assistant agent, so that the agent has a plan for creating a `DweetArtifact`, and then using the artifact to send messages. You can visit the "CArtAgO By Example" guide (available online[14]) for studying how to program CArtAgO artifacts and their operations, and how to program agents that create artifacts and invoke artifact operations.

Task 1.2 (3 points): Agent-to-Environment interaction based on W3C WoT Thing Descriptions Study the agent program of the wristband manager agent. Once deployed, the wristband manager creates a `ThingArtifact` for monitoring the user's activities based on the WoT Thing Description of the user's wristband. Then, every few milliseconds, the wristband manager exploits the property affordance `was:ReadOwnerState` for reading whether the user is "asleep" or "awake", by invoking the operation `readProperty()` of the `ThingArtifact`.

Complete the implementation of the following agents so that they interact with their environment based on W3C WoT Thing Descriptions:

- Update the implementation of the calendar manager agent. Once deployed, the agent should create a `ThingArtifact` based on the WoT Thing Description of the calendar service. Then,

[6] Lights controller agent: https://github.com/HSG-WAS-SS23/exercise-6/blob/main/src/agt/lights_controller.asl

[7] The implementation of `ThingArtifact.java` is part of the `jacamo-hypermedia` library: https://github.com/HyperAgents/jacamo-hypermedia

[8] Calendar service TD: https://github.com/Interactions-HSG/example-tds/blob/was/tds/calendar-service.ttl

[9] Wristband TD: https://github.com/Interactions-HSG/example-tds/blob/was/tds/wristband-simu.ttl

[10] Blinds TD: https://github.com/Interactions-HSG/example-tds/blob/was/tds/blinds.ttl

[11] Lights TD: https://github.com/Interactions-HSG/example-tds/blob/was/tds/lights.ttl

[12] DweetArtifact.java: https://github.com/HSG-WAS-SS23/exercise-6/blob/main/src/env/room/DweetArtifact.java

[13] dweet.iot API: https://dweet.io/

[14] CArtAgO By Example: https://www.emse.fr/~boissier/enseignement/maop13/courses/cartagoByExamples.pdf

the agent should exploit the property affordance `was:ReadUpcomingEvent` of the service for reading when there is an upcoming event (e.g. “now”).

- Update the implementation of the blinds controller agent. Once deployed, the agent should create a `ThingArtifact` based on the WoT Thing Description of the blinds in the room. Additionally, write the necessary plans so that the agent can achieve the goal of raising the blinds and the goal of lowering the blinds, by exploiting the action affordance `was:SetState` of the blinds with the appropriate input (“raised” or “lowered”).
- Update the implementation of the lights controller agent. Once deployed, the agent should create a `ThingArtifact` based on the WoT Thing Description of the blinds in the room. Additionally, write the necessary plans so that the agent can achieve the goal of turning on the lights and the goal of turning off the lights, by exploiting the action affordance `was:SetState` of the blinds with the appropriate input (“on” or “off”4).

Task 1.3 (1.5 points): Agent-to-Agent interaction with performatives Now the wristband manager, the calendar manager, the blinds controller, and the lights controller, successfully interact with the devices and services they manage. However, these agents do not yet share their knowledge about the user and the environment with the user’s personal assistant agent. Update the implementation of the agents and use the necessary performatives (see Lecture 6), so that the agents inform the personal assistant about changes on the states of the user and the environment.

Task 1.4 (4 points): Agent-to-Agent interaction based on the Contract Net Protocol Now the personal assistant agent receives information about the states of the user and the environment. Complete the implementation of the personal assistant, the blinds controller, and the lights controller so that the agents can coordinate for waking up the user in case there is any upcoming event.

1.) Update the implementation of the personal assistant agent so that it reacts to the addition of the belief that there is an upcoming event “now”. If the user is awake, the assistant should print “Enjoy your event”. If the user is asleep, the assistant should print “Starting wake-up routine”.

2.) Update the implementation of the personal assistant agent so that, when initialized, the agent holds beliefs about how the user prefers to be woken up (the most preferred method is assigned the lowest rank):

- the user ranks waking up with artificial light with a 1.
- the user ranks waking up with natural light with a 0.

3.) Update the programs of the personal assistant, the blinds controller, and the lights controller so that, when there is an upcoming event “now” and the user is asleep, the agents coordinate with each other to wake up the user. The assistant should attempt to achieve this goal by interacting with the blinds controller and the lights controller based on the Contract Net Protocol (specified here[15]). Initially, the assistant should *broadcast* a Call for Proposals for the task of increasing the illuminance in the room. Both controller agents should react to the call by *sending messages to the assistant*: If the blinds are lowered, the blinds controller should propose to raise the blinds – otherwise, it should refuse the call. Respectively, if the lights are off, the lights controller should propose to turn on the lights – otherwise, it should refuse the call.

The assistant should always accept the available proposal taking into consideration how the user prefers to be woken up, and reject any other received proposals. If any of the controller agents’ proposal has been accepted, the relevant controller agent should react by acting on the environment according to its proposal, i.e. by raising the blinds or turning on the lights. The

[15]FIPA Contract Net Interaction Protocol Specification: http://www.fipa.org/specs/fipa00029/SC00029H.html

expected outcome is that the assistant, first, attempts to wake up the user by accepting the proposal of the blinds controller, and then, by accepting the proposal of the lights controller. However, the user remains asleep.

4.) Update the implementation of the personal assistant agent, so that, if the agent initiates the Contract Net Protocol but no proposals are received, it strives to wake up the user by asking one of the user's friends to wake them up (i.e. it delegates the goal to the user's friend). Use the plan that you implemented in Task 1.1 so that the assistant sends such a message using the `DweetArtifact`. Regardless, our user in this assignment is really a heavy sleeper — so, despite all the assistant's attempts, the user will remain asleep.

② **Hand-in Instructions** By the deadline, hand in via Canvas a **zipfile** that contains:

1.) a PDF file that includes the link to the GitHub fork containing **code for Task 1**;
2.) any additional instructions for how to run your code (if non-obvious).