Homework 1 **Kim Hammar**
ID2209 Due Date: 14 November 2016
Distributed Artifical Intelligence and Intelligent Agents

Problem Statement

The given task was to implement a very basic MultiAgentSystem (M.A.S) in the JAVA Agent Development Framework (JADE) [2], with the purpose of getting hands on experience with agent platforms and in particular the JADE agent platform. The assignment were also designed in order to give experience programming agents in the context of a *practical scenario*.

Main problems and solutions

- *Connecting agents through a platform that can be used for interaction and service discovery*
  The JADE framework provides among other things a runtime environemnt, where JADE agents can "live". The agents in the M.A.S for this assignment are distributed on different JADE containers that all are connected to a single platform (it is possible to have multiple platforms but this is not used in this assignment), which enable the agents to utilize the JADE runtime environment to find and interact with each other.
- *Agent design (micro perspective): designing agents that can act autonomously in a given environment and make decisions.*
  Agents in this M.A.S perform tasks and perceive the environment to achieve their goals. Using JADE, the main mechanisms for designing and implementing agent tasks is through use of behaviours.
- *Society design (macro perspective): designing interactions for cooperation, coordination and negotiation between agents in a M.A.S*
  The Foundation for Intelligent Physical Agents (FIPA) is used as the main Agent Communication Language (ACL) between agents in this M.A.S. FIPA was convenient as ACL since JADE is to a large extent an implementation of FIPA specifications and provides good support for that particular ACL.

Connecting Agents

We refer to each running instance of JADE runtime environment as a Container, which can contain zero or more agents. The set of active containers is called a platform and each platform have one container that is a special *main container* (effectively the first started container becomes a main container and the "normal" containers need to specify which main container to connect to) [1]. The most important aspect of the main container apart from connecting other containers is to provide two important services: **AMS and DF**, AMS (Agent Management System) is a naming service that ensures that each agent in the platform has a unique name. DF (Directory Facilitator) aka "The Yellow Pages" is a service where agents can *register* as providers of certain services and where agents can *search* for providers of specific services.

When running the M.A.S for this assignment the typical setup is to use 4 containers:

I **Main container**: required container for connecting the other containers. Does not run any agents directly in this setup.

II **Container 1**: Container where one ore more *curator* agents live.

III **Container 2**: Container where one ore more *tourguide* agents live.

IV **Container 3**: Container where one ore more *profiler* agents live.

Agent Design

There are three different types of agents in this system,

- **CuratorAgent**: Agent with the goal of monitoring an artgallery and respond to requests for art information from TourGuideAgents and ProfilerAgents. The CuratorAgent registers at the DF as a provider of the `artgallery-information` service, this agent perceives its environment mainly through the main container and its message-mailbox. The tasks/JADE behaviours of this agent are:
  - A `ParallelBehaviour` consisting of three `SubBehaviours`:
    * `GenreRequestServer` - A `CyclicBehaviour` that receives requests for a list of all genres of the monitored art gallery and responds to it.
    * `TourRequestServer` - A `CyclicBehaviour` that receives requests for a list of artifacts in the artgallery that matches a certain genre/interest and responds to it.
    * `ArtifactRequestServer` - A `CyclicBehaviour` that receives requests for details about a certain artifact in the artallery and responds to it.
- **TourGuideAgent**: Agent with the goal to build virtual tours of art galleries upon requests from ProfilerAgents. The TourGuideAgent interacts with CuratorAgents to retrieve information for tours. The TourGuideAgent registers at the DF as providing the `virtualtour` service, and perceives its environment mainly through the main container and its message-mailbox. The tasks/JADE behaviours of this agent are:
  - A `ParallelBehaviour` consisting of three `SubBehaviours`:
    * `CuratorSubscriber` - A `SubscriptionInitiator` behaviour that subscripes to the DF service to receive notifications when new agents that provide the `artgallery-information` service (a service that is provided by CuratorAgents) registers.
    * `ProfilerMatcher` - An `AchieveREResponder` that receives requests from ProfilerAgents asking what kind of genres it can build virtual tours for. This behaviour is linked to the `FindSupportedInterest` behaviour (which is an `AchieveREInitiator`) that will be invoked when a request is received in order to: *(i)* ask discovered curator agents about their genres, *(ii)* build a list of all genres, *(iii)* respond to the requester with the list of genres.
    * `VirtualTourServer` - An `AchieveREResponder` that receives requests for virtual tours for specific interests/genres from profilers. When a request is received it will cause the `BuildVirtualTour` behaviour to be invoked which is a `AchieveREInitiator` that will send requests to all discovered curators and

build a list of $\langle Artifact, Curator \rangle$ pairs that matches the given interest and finally respond it to the requester.

- **ProfilerAgent**: Agent that maintains the profile of a user and that has the goal of travelling around the network and collecting interesting (from the user's point of view) information about art. The agent perceives its environment mainly through the main container and input from the user. The tasks/JADE behaviours of this agent are:
  - A `FSMBehaviour` consisting of 6 states and 9 different state transitions. The states are:
    * `INITIALIZE_USER_PROFILE` - A `OneShotBehaviour` that interacts with the user for initializing the user profile (only invoked if command-line aguments were not supplied)
    * `SEARCH_TOURGUIDES_STATE` - A `OneShotBehaviour` that queries the DF for a list of all agents that provide the `virtualtour` service.
    * `FIND_MATCHING_TOUR_GUIDES_STATE` - An `AchieveREInitiator` that sends request-queries to all found TourGuideAgents asking what type of tours they provide.
    * `SELECT_TOURGUIDE_STATE` - A `OneShotBehaviour` for presenting the found tour-guides and the types of tours they offer to the user and letting the user choose a tourguide.
    * `FIND_VIRTUAL_TOUR_STATE` - An `AchieveREInitiator` that sends a request to a chosen tourguide, requesting a virtual tour matching the interest of the user.
    * `SELECT_ARTIFACT_STATE` - A `OneShotBehaviour` for presenting the virtual tour to the user and letting the user pick artifacts to visit.
    * `RETRIEVE_ARTIFACT_STATE` - An `AchieveREInitiator` that sends a request to the curator of the artifact in the virtual tour that was selected, for details about the artifact. When the details are retrieved they are presented to the user.

Society Design

From the presentation of the agent designs it should be clear that they need to interact in order to complete their individual goals. As mentioned, FIPA ACL is used for communication to gurarantee a consistent syntax of messages. The central concept for communication is that agents explicitly state a *performative-verb* together with the content of each message, the performative verb decides how the agent at the receiving side will interpret the message. FIPA defines a large set of performative verbs, the main ones used in this M.A.S are:

query-ref Used by one agent to determine the specific value for an expression [3], for example when a profiler agent queries tour guides for the types of virtual tours they support.

request Allows an agent to request another agent to perform some action [3], for example when a profiler agent requests a tourguide agent to build a virtual tour for a specific interest.

agree Used to indicate that the agent has agreed a request made by another agent [3], for example when a tourguide receivs a request to build a virtual tour it responds with a message with this performative before actually building the tour.

inform Basic performative for communicating information [3], used for example by an tourguide agent to inform a profiler after having successfully built a virtual tour upon request.

failure Indicates that an attempt to perform some action failed [3], used by tourguide agents to respond to profilers if they fail to build a virtual tour (for example if curators are not responding).

Conclusions

Building a M.A.S consisting of autonomous agents that interact with each other is different to building "regular" distributed systems. In this assignment the agents were *benevolent* towards each other which eased the development quite abit, since there were no need to be concerned about tricky negotiations and agent-strategies. The main hurdle to overcome when developing this M.A.S as opposed to regular distributed systems was to understand how the agents interact through performative verbs, JADE libraries were very useful in this aspect.

Attachments

Documented source code can be found in the attached zipfile. See README.MD in the root directory for instruction on how to execute the program.

References

[1] Giovanni Caire. Jade turorial. `http://jade.tilab.com/doc/tutorials/JADEProgramming-Tutorial-for-beginners.pdf`, 2009. [Online; accessed 12-Nov-2016].

[2] Telecom Italia. Java agent development framework. `http://jade.tilab.com/`, 2016. [Online; accessed 11-Nov-2016].

[3] Michael Woolridge and Michael J. Wooldridge. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.