




Multi Agent Restaurant

AIAD 2019/2020

4MIEIC03, Group 35



Problem Description

Our project replicates a restaurant environment with a multi-agent system. In this system, we have three different types of agents: Kitchen, Waiter and Customer. There is only one Kitchen, but multiple Waiters and Customers.

As a regular restaurant would work, the customers interact with the waiters to get their order done and the waiters interact with the kitchen and with other waiters as a way to get information quickly. The kitchen verifies the orders' availability and takes care of the order preparation. Further details about these interactions will be presented soon.

Project Global Schema

Customer:

- Orders a meal.
- When meal is delivered, it gives the waiter a tip based on its overall mood.

Waiter:

- Takes customers orders.
- Can suggest other dishes to customers.
- Can provide information (reliable or not) to other waiters about other dishes.
- Tells the kitchen staff to actually prepare the meals.
- Delivers food to customer and receives tip.

Kitchen:

- Provides accurate dish information to waiters.
- Has final say on dish preparation.

Communication- Interaction and Protocols

FIPA Request:

- Customer first contact with a waiter, to know if he's available.
- Waiter requests for dish information, either to another waiter or the kitchen staff.
- Waiter final check with kitchen to start dish preparation (if actually possible).
- Meal delivery to customer by waiter and tip receipt.

FIPA Contract Net:

- Waiter conversation with customer, where its order is taken and evaluated, with other dishes being suggested depending on its current mood and initial request.

Used Strategies

When a waiter doesn't have information about a particular dish, it can choose to ask another waiter or the kitchen staff.

The first way is "quicker", however, the information received might not be entirely accurate (either on purpose or not).

On the other hand, when asking the kitchen staff the information relayed is always up to date but "it takes longer" do this, dropping the customer's mood.

Used Strategies

There are two main strategies waiters can apply regarding information sharing:

1. Full cooperation - When asked about a certain dish, a waiter will, whenever possible, provide the information he believes to be most accurate.
2. Lying - In this strategy, whether a waiter actually possesses information about the requested dish or not, he will reply with possibly false information about it, making it look like the dish is worse than it actually is in order to make the other waiter suggest another one, thus not decreasing the availability of that dish.

Other Mechanisms

- “Yellow Pages”:

We use the Directory Facilitator (DF) agent as “Yellow Pages”, where other agents can register to “advertise” their services.

In our system, the waiters and the kitchen register themselves in the Yellow Pages, by adding their service details (*ServiceDescription*) to a *DFAgentDescription*, later on registering the agent description and themselves (*this*) to the *DFService*, so that other agents can find the waiters by searching in that service.

Used Software Specifications

For this project we used the **JAVA Agent DEvelopment Framework - JADE**, which is an open source platform for peer-to-peer agent based applications.

JADE is particularly useful when it comes to implementing agent communication based on FIPA Interaction Protocols since it complies with FIPA specifications.

Experiments Made

- One customer and one waiter
- One customer and multiple waiters
- Multiple customers and one waiter
- Multiple customers and waiters

Result Analysis

In the short term, lying seems to be favorable as it:

- allows waiters to manipulate information to their benefit.
- harms other waiters' tips thus giving advantage to the "liar waiter" when it comes to comparing the total tips between waiters.

However, the factor with the biggest impact on the amount that the customer tips is his initial mood, so depending on their customers' mood, each waiter already stands in disadvantage / advantage.

Conclusions

Overall, this system tries to mimic reality in such a way that emotional factors are taken into account and that interactions with other agents have effects on those factors.

For future work, it would be interesting to have the customers choose a meal based on their preferences (for instance cow meat, vegetarian, salad...) instead of choosing randomly.

Detailed Execution Examples

Let's see analyse an execution example.

First, we'll launch the application with the following program arguments:

```
-agents  
John:agents.Waiter(true);Alex:agents.Waiter(false);Paul:agents.Customer;  
Sarah:agents.Customer;Seth:agents.Kitchen
```

This means that we'll have two Waiter agents (John and Alex), two Customer agents (Paul and Sarah) and one Kitchen agent (Seth).

John has a full cooperation strategy while Alex has a lying strategy.

Detailed Execution Examples: setup

These are the first messages we see:

```
(Customer Sarah) Hello! Customer Sarah is ready.  
(Customer Sarah) My mood: 1  
(Customer Paul) Hello! Customer Paul is ready.  
(Customer Paul) My mood: 7  
(Waiter Alex) Checking in.  
(Waiter John) Checking in.  
(kitchen) Kitchen Seth at your service.
```

These are printed in the `setup()` method of each agent. The customer's mood is randomly generated during setup.

Detailed Execution Examples: ordering the meal

After the setup messages, we get this:

```
(Customer Sarah) Are you available, John?  
(Customer Paul) Are you available, John?  
(Waiter John) I'll gladly be your waiter this evening, Sarah.  
(Waiter John) Go ahead, what can I get you?  
(Waiter John) I'm sorry, I'm a bit busy at the moment, Paul.  
(Customer Sarah) I would like to eat escargots.  
(Waiter John) Hold on a minute, let me ask my colleague.  
(Waiter Alex) Hmm, let me think...  
(Waiter Alex) *Lies* Yes, here you go: escargots - 5 - 9 - 0  
(Waiter John) Excellent choice!  
(Customer Sarah) Perfect, just what I wanted!  
(Waiter John) Right away!  
(Kitchen Seth) Here you go: escargots - 2 - 3 - 4  
(Waiter John) *Thinking* Alex was lying, I'll take note of that...  
(Waiter John) Your meal is being prepared.
```

Notice that John asks information from Alex, and he lies. Later, John discovers he lied when he actually goes to the kitchen to order the meal.

Detailed Execution Examples: receiving the meal

```
(Waiter John) A dose of escargots, just like you ordered, Sarah.  
(Customer Sarah) Thank you!  
(Customer Sarah) I'll pay now.  
(Customer Sarah) Here's your tip: 0.82€.  
(Waiter John) Thank you very much Sarah for the 0.82€!  
(Customer Sarah) My mood: 1  
(Waiter John) I have collected 0.82€ in tips so far!  
(Customer Sarah) Going home
```

```
(Waiter Alex) A dose of king crab, just like you ordered, Paul.  
(Customer Paul) Thank you!  
(Customer Paul) I'll pay now.  
(Customer Paul) Here's your tip: 3.06€.  
(Customer Paul) My mood: 6  
(Customer Paul) Going home  
(Waiter Alex) Thank you very much Paul for the 3.06€!  
(Waiter Alex) I have collected 3.06€ in tips so far!
```

Customer's mood is generated randomly during the setup and has the biggest influence on the tip. However, it increases or decreases depending on some factors (e.g. if the meal is well prepared or takes little time). Note that Paul's mood actually dropped (initially it was 7).

Implemented Classes

- Agents

- **RestaurantAgent** - abstract “mother” class to other agents.
- **Kitchen** - agent that prepares dishes and responds to waiters enquiries.
- **Waiter** - agent that takes care of customer orders and interacts with kitchen or other waiters to get information in order to aid the customer interaction.
- **Customer** - agent that requests the waiters’ service, its mood influences the tip given.

- Behaviours

- **OrderPerformer** - customer behaviour, when a waiter agent is found through *ServiceSearch*, this behaviour is initiated to start the order.
- **ReceiveMeal** - customer behaviour, receives meal from waiter and gives the tip.
- **ReplyToWaiter** - waiter behaviour, describes interaction between waiters when one asks another details about a certain dish.
- **ServeMeal** - waiter behaviour, initiates a protocol request between the waiter and customer, for the waiter to deliver the meal and receive the customer tip.

Implemented Classes

- Behaviours
 - **ServiceSearch** - customer and waiter behaviour, used to find other waiters.
 - **TakeOrder** - waiter behaviour, describes various courses of action depending on the stage that the waiter agent is: take an order from the customer, get dish details from another waiter or kitchen agent, get feedback from the customer or from the kitchen.
 - **TakeRequest** - kitchen behaviour, receives requests from the waiter agent to either send a certain dish details or to start preparing a requested meal.
 - **AttendCustomer** - waiter behaviour, receives requests from customers that are unattended and replies affirmatively if he is not currently engaged in a conversation with another customer.
- Utils
 - Dish - object that saves details of a certain dish.
 - Pair - basic data structure to group two values/objects.

Other Observations

In regards to the lying mechanic between waiters, the motivation for this action is, to make the dish look bad to the other waiter so as to not decrease its availability, incentivizing the requesting waiter to choose another one. In addition to this, the customer would then give him a smaller tip since its mood also depends on this factor, contributing to the lying waiter's goal of accumulating the biggest tip amount.

However, this strategy can be a double edged sword. If the requesting waiter still finds the dish appealing and requests it, the kitchen will eventually relay the real details to him. The waiter will then brand the other waiter as a liar, refusing any requests for information from his part in the future. As such, a lying waiter can become "isolated" and be forced to ask a dish's details always from the kitchen, dropping its customer mood in the process.