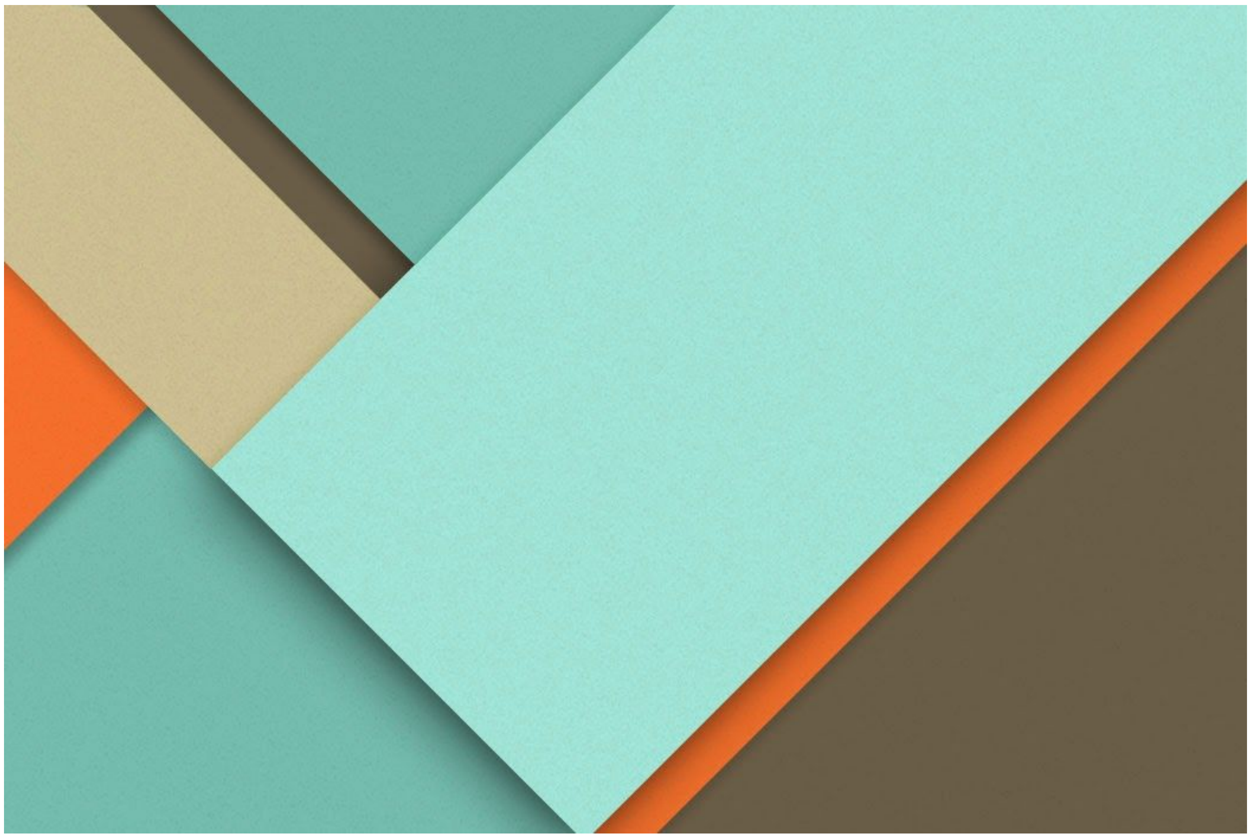


SPRINT 2

17.09.2020

Laura Mazzuca, Nicholas Antonio Carroll, Giuseppe Giorgio

Requirement Analysis

Addition to the Requirements

From Devs:

- The Waiterwalker should be divided into Waiterwalker and Walker.

From Commissioner:

- The Client should interact with the Waiter through an Interface.

(New) Assumptions

- we **will** consider the presence of **only one client**;
- The client, the barman and the smartbell will be simulated and grouped together with the waiter context-wise;
- we **will not** consider the existence of the manager and the web application he would use to monitor the state of the tearoom;
- we **will not** take into account optimization of tasks;
- The waiter **won't** be able to interrupt a task;
- We **won't** take into account the MaxStayTime and MaxWaitTime timeouts;
- The Waiterwalker **will** be divided into Waiterwalker and Walker;
- The Client **will** be able to interact with the Waiter through an Interface.

Waiterwalker -> Waiterwalker + Walker

The division is meant to bring to light the separation between the actor that handles the interaction with the basicrobot and the actor that is meant to find the best path to the destination that the waiter has to reach, depending on the task the Waiter is executing.

The Client Interface

The Clients need a way to communicate with both the Waiter and the Smartbell. An easy way to do so would be having a User Interface that enables the Client to make requests such as:

- Ring the smartbell
- Ask to be taken to the table
- Order
- Pay
- Ask to be escorted to the Exitdoor

All of the above actions have already been analyzed during the first SPRINT.

Problem Analysis

Refactoring Waiterwalker: improving our Microservices

As it was noticed in the Project Implementation during SPRINT1, the need to divide the Waiterwalker actor from a more simple Walker has risen to ensure **reusability of components** developed and the **Single Responsibility Principle**, which will help greatly with the support for the system if the need for new requirements arises.

Also, doing so will help decoupling the Actors, which enables us to much more easily test and deploy them on different independently.

The system should then behave as the following diagram:

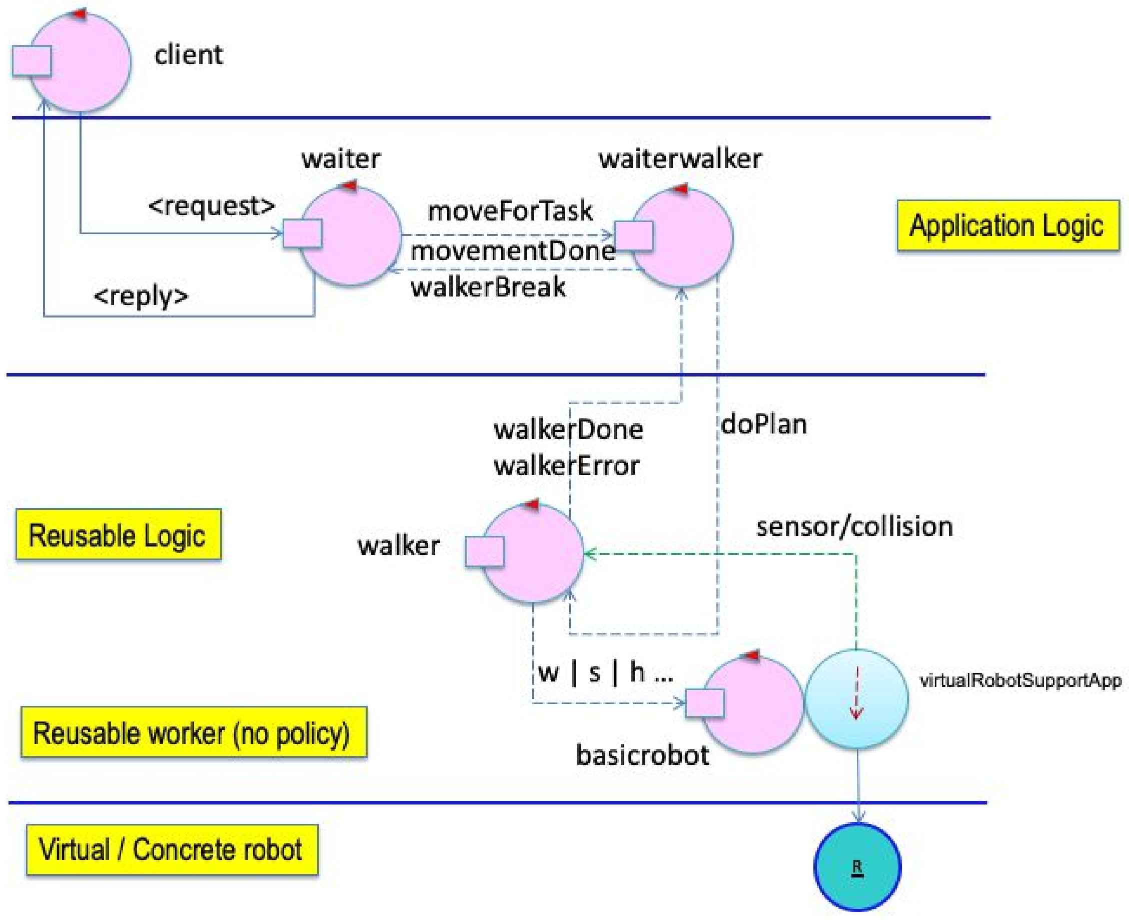


Fig. 1: Architecture after refactoring

Client-Waiter interaction

Another issue that has come to light is the necessity to separate and make completely independent the Client from the Waiter.

As a matter of fact, as of SPRINT1 we had a Client deeply bound to the waiter's system through a direct exchange of messages from the Waiter to the Client, meaning that the Waiter would have had to know directly the Client.

Since we want to build clients that are not necessarily QActors, but may be any kind of *alien*, we need to change the logic of the interaction. In particular, **what makes for a better exchange of messages is a series of Request-Reply, where the Client requests a service and the Waiter, once he executes it, replies with the proper payload**.

The Client can also send Dispatch messages to share request parameters that can only be shared when the Waiter is actually ready (i.e. the tea ordered or the amount paid).

The following image represents the new interaction diagram:

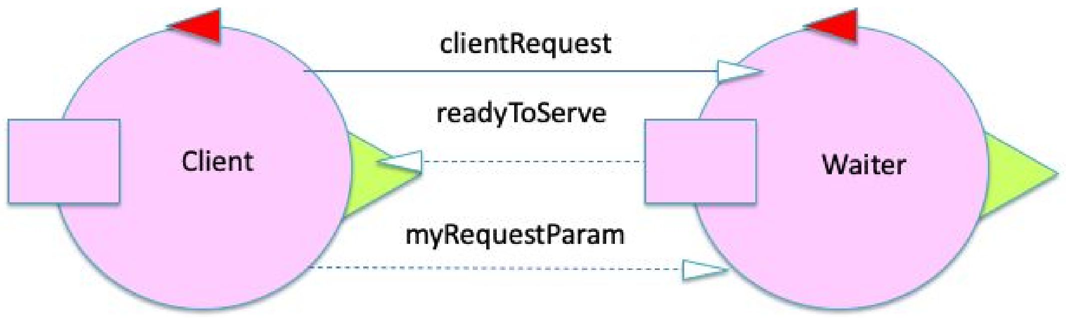


Fig. 2: Client-Waiter interaction

Client User Interface

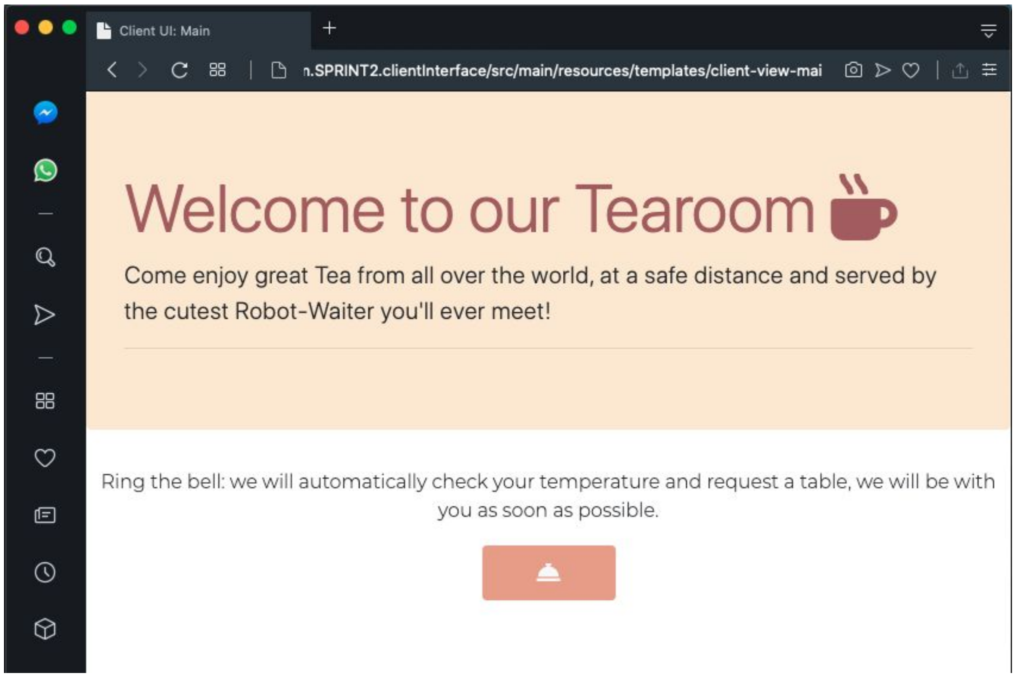
Lastly, we discuss the Client User Interface. Keeping in mind that we also have to develop an Interface for the Manager, we want to build a reusable Interface for the Model (the Tearoom System) and the Views. This Interface will need to enable the View to:

- **Specify if the View is for interaction or monitoring**. For example, the Client would need to interact with the system while the Manager only needs to monitor it;
- **Handle the communication with the Model**;
- **Update the View accordingly**.

An Architecture that can easily handle all this and that can last through any change in Requirements should provide a way to configure the Controller depending on needs, leaving it as independent as possible from both the Model and the View.

MOCKUPS

The following images are intended as Mockups of a Web Application for the Client.

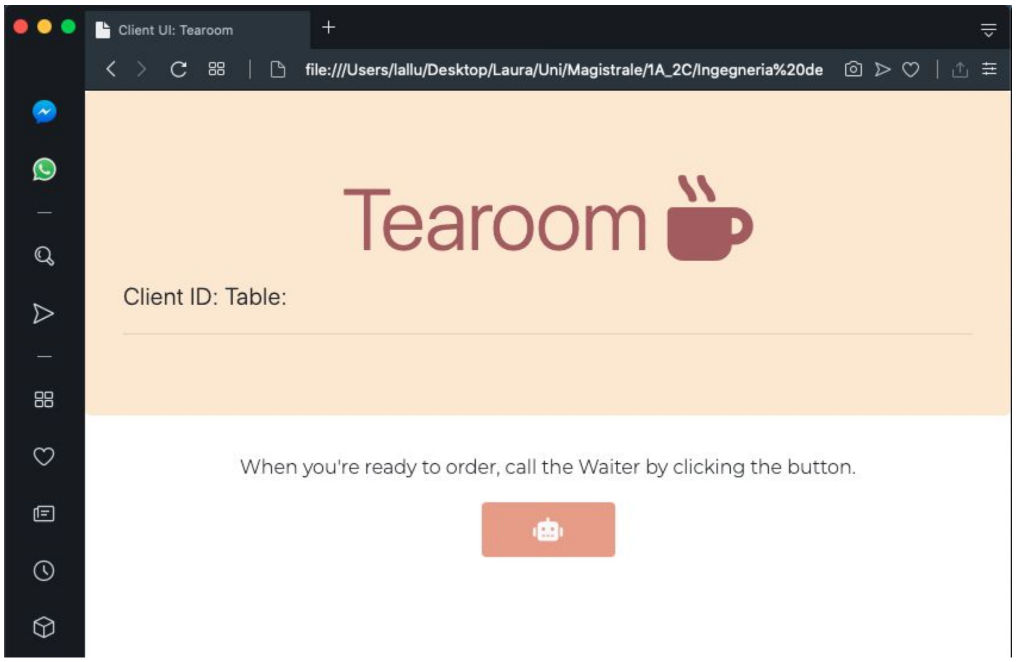

Client UI: Main
n.SPRINT2.clientInterface/src/main/resources/templates/client-view-mai

Welcome to our Tearoom

Come enjoy great Tea from all over the world, at a safe distance and served by the cutest Robot-Waiter you'll ever meet!

Ring the bell: we will automatically check your temperature and request a table, we will be with you as soon as possible.

Fig. 3: Greetings Client page


Client UI: Tearoom
file:///Users/lallu/Desktop/Laura/Uni/Magistrale/1A_2C/Ingegneria%20de
Tearoom

Client ID: Table:

When you're ready to order, call the Waiter by clicking the button.

Fig. 4: Main Client page, once admitted inside the Tearoom and escorted to the table

Test

The **waiterwalker** should be tested for:

- correct handling of moveForTask messages.

The **barman** and smartbell actors should be tested for:

- ability to answer unexpected messages.

The **waiter** should be tested for:

- its behaviour when he receives Requests that are different from the ones in the main schedule.

Project

Waiterwalker-Walker division

To implement the separation, we have left only the Knowledge Base to the Waiterwalker, while the Walker handles the planning of the route and the communication with the Basicrobot.

The interaction between the two Actors is handled through an exchange of Request-Reply messages.

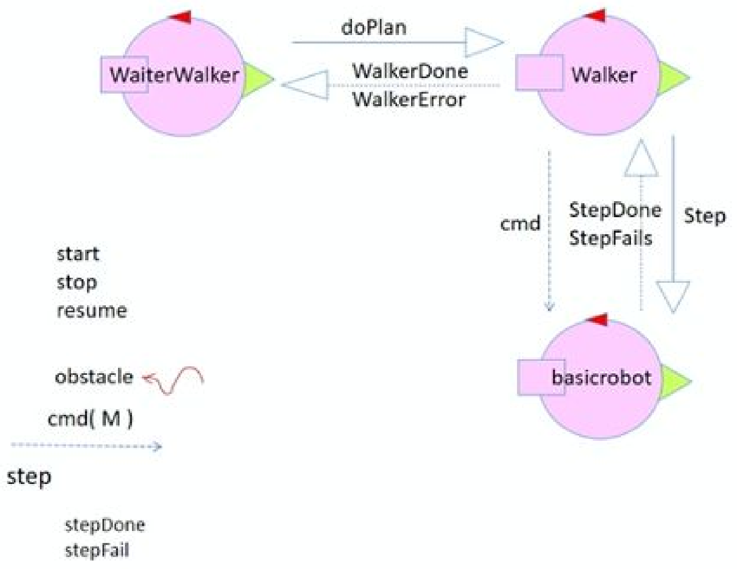


Fig. 5: Interaction after division between Waiterwalker and Walker

Client - Waiter Interaction

The Client, that was previously part of the Tearoom Context and System, is now independently defined in the `simclient.qak`, which grants the possibility to simulate the Client's behavior (only the "correct" one) without using a Graphic User Interface.

The communication between the Client and the Waiter has been radically changed. **Now, the Client always sends either a Request, to ask for a task to be executed, or a Dispatch, to send ulterior information, if required, while the Waiter sends a Reply when the Task is completed.**

This separation has enabled a clearer implementation of tests and will greatly help with ***the Client User Interface, which will be developed in the next SPRINT***.

Test

The tests have been implemented in the two classes:

- `SPRINTS/tearoom.SPRINT2/test/TestRobotPosition.kt`
- `SPRINTS/tearoom.SPRINT2/test/TestTearoomSys.kt`

Backlog Update

- Implement the Client Interface and the Server on which it should work on.