VRIJE
UNIVERSITEIT
BRUSSEL

SOFTWARE ARCHITECTURES

Report assignment 3

Christophe Van den Eede

December 28, 2021

Sciences and Bio-Engineering Sciences

1 Introduction

In this assignment we were instructed to implement a vacation booking service using Akka Actors. This report will describe what design choices I made in terms of Actor patterns.

2 Program flow

In this part i will describe how I interpreted the given specification.

- A Client is created and given a reference to the SystemService they will be allowed to use, their personal information and a query to be sent there. I assumed the Client in the actor system will be something invisible to the user, that is called by for example a desktop application.
- The Client will turn their Query into a Search by adding their own ActorRef and send it to the SystemService.
- The SystemService will use the given Search values to filter through its cache of existing properties. Followed by filtering the results with the reservation cache to make sure no properties that are booked on the requested day are returned. A list of available properties is sent back to the Client. An ask pattern would not make sense here because the Client has no other tasks while waiting for a result.
- In this example system the client will choose a random property from the list and send a MakeReservation message to the SystemService. If none were returned: halt.
- The SystemService will, upon receiving this message, create an ephemeral child and forward the MakeReservation message to it.
- Here we come to the first point of the specification i found somewhat unclear. The child should now confirm that the property is available on the given date and *then* ask the ReservationService to confirm it. This seems like it just moves the concurrency problem further along instead of solving it. Instead i chose to immediately send the MakeReservation message to the ReservationService and expect a Confirm or Deny message back. Again, an ask pattern does not make sense for this request, since the child has to tasks to perform while waiting for a response.
- The ReservationService, upon receiving the MakeReservationMessage, will look into its reservations map to see if the requested property is available on the requested day and respond with Confirm or Deny appropriately.
- When the child actor receives a message stating that its reservation was accepted, it will tell both the SystemService (to update its cache), and tell the client the good news. If the reservation was instead denied, this means the SystemService's cache was invalid. As such the child will send a CacheInvalidated message to the SystemService and tell the client the bad news.
- The client, upon receiving a message saying their reservation was accepted, will acknowledge to the child actor that the message was well received. should the message say that the reservation was refused, the client will acknowledge reception and again ask the SystemService for all available properties, to restart the process again.
- When the child actor receives acknowledgment of reception, it self terminates.
- When the SystemService gets told that its cache has been invalidated, it will request both the existing properties and the reservations from the ReservationService. For this request it does make sense to implement an ask pattern, because the SystemService will still have to manage many requests as it waits for the requested data.

3 Pattern choices

When looking into the ReservationService, it is an actor that in accordance to received messages, can manage both an object containing all properties (static in this assignment but cerntainly possible) and an object containing the reservations for these properties. This means i have implemented the Domain Object pattern.

When the SystemService updates their cache, this is done by using the ask patter, through the ask operator (?). However besides this case, the emphemeral child described in the specification also mandates the implementation of the Ask pattern. The child waits for and manages the response from the ReservationService, sending a result to the client instead of passing back through the SystemService.

4 Interesting Implementations

I have one piece of code that i wanted to touch upon more in detail, as I believe this is quite an interesting piece of code. Though this has less to do with Actor patterns and more with Scala itself. To implement the ”optional” properties in the Search message, I defined them as Option values with a default value of None. This way, the Client can choose to leave these parameters empty if they don't care about them. To manage this when deciding what properties fulfil the demands, I was able to filter the properties step by step for every parameter. By using *.getOrElse* i was able to take these values or apply a default value. For example for the name of the property, the default value for the string could be ””. This enables us to apply *.contains* on the string of the to-filter property. As *”abcde”.contains(”bc”)* results in true, using the empty string to filter will always result in true and return all values. For the category this was more of a problem, if true can only be the result if the category matches exactly, there would be no appropriate default value to return all options. Instead i chose to implement the filter such that the given value for category is the *lowest* value the user will accept. Using this I set the default value to be low, allowing all options to return.