VRIJE
UNIVERSITEIT
BRUSSEL

ACTOR-BASED DESIGN PATTERNS

Assignment 3 - Software Architectures

Lennert Bontinck

Second session, 2020-2021

Student number: 568702

Computer Science: AI

Contents

General remarks

1.1 Notes on the assignment

None of the assignments for the Software Architectures course were submitted in the first examination period due to personal reasons. Because of this, all of the code written for this assignment is written specifically for the second examination period. All code was written by using the course material and the online Akka documentation[1].

1.2 Important design assumptions

Whilst the assignment gave great details on which components should be created, there were some gaps in connecting these components. It was opted to create extra non-specified functionality of adding stock houses and products so that these gaps are filled realistically compared to hard-coding `ActorRef` lists and `ProductsWithQuantity` in stock. Sadly, this decision left no time to implement extra patterns such as a persistent business handshake pattern.

1.3 Important files

All code written is available on the GitHub repository for this assignment (Bontinck, 2021). Rights to this private GitHub repository can be granted upon request. A copy of this GitHub repository is accompanied by this report. An overview of important files is given below:

- `README.md`
  - General information of the GitHub repository as well as some technical details about the used environment and how to run the project.
  - A section called `Validated output` where the main functionality is discussed informally by discussing some screenshots.
- `assignment.pdf`, `Lennert-Bontinck-SA3.pdf` and `report/`
  - The assignment PDF, this report and the source files of the report. A VUB themed LaTeX template by De Smet (2020) was used to create this report.
- `code/Lennert-Bontinck-SA3/`
  - The folder containing the code of the assignment solution.
  - Instructions on how to run the code are available in the `README.md` file.

[1] https://doc.akka.io/docs/akka/2.5.32/

Project defence

In what follows a high-level discussion is held on the created code. The `README.md` file contains a more informal discussion on the functionality as well. Multiple figures and screenshots were made to illustrate the different parts of the project. Some of these figures are provided in the extra figures list at the end of this report. For more technical details the reader is invited to read the documentation and comments in the code.

2.1 Used patterns

The main used patterns are: `Forward Flow`, `Aggregator` and `Domain Object`. A custom `Priority Queue Mailbox` was implemented as well. The `Domain Object` pattern makes for a clear separation of business logic and communication logic as requested by the assignment. A type of `Aggregator` pattern is used to collect all requested items of a purchase. The `Forward Flow` pattern is not so much of a pattern as it is a *mind set*. Special care was taken to eliminate non-necessary *man in the middle* by specifying `replyTo` addresses. The custom `priority queue mailbox` that is used as default `Mailbox` for all `Actors` gives a higher priority to messages from `Prime` members.

2.2 Managing stock houses

Whilst not specified in the assignment, it is possible to add stock houses and inventory through messages such as `AddStockHouse` and `AddProductToStockHouse`. Adding `StockHouse` objects is done in business logic, inside a `StockHouseManager`. This manager will keep a list of stock houses it created. The `StockHouseManagerService`, responsible for the communication logic, also stores the `ActorRef` and `Name` for each of the added stock house. It does this by maintaining a list of `NamedActor` objects, which have a unique name based on the stock house address and the Actor's `ActorRef`. The clear separation of business logic is noticeable here and everywhere in the project, as the `Domain Object` pattern is used. The `Main` loop initialises some stock houses and inventory, as displayed in figure 2.3

2.3 Starting a purchase process and ephemeral children

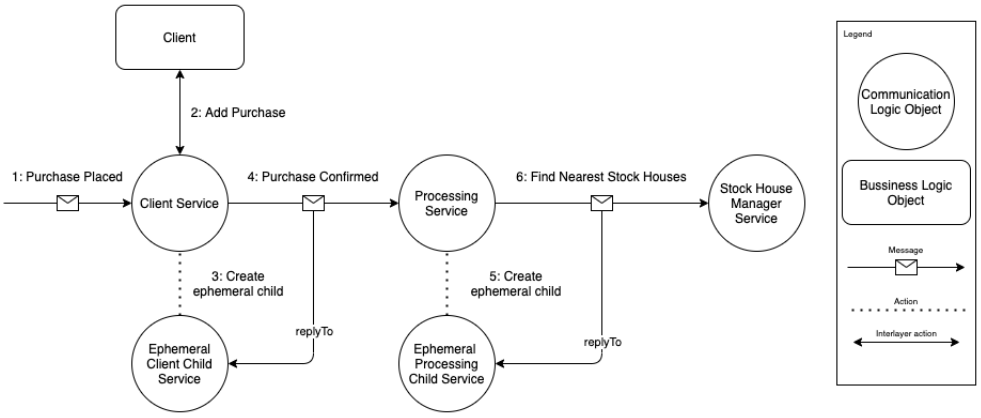

Figure 2.1: Simplified representation of first phase of purchase process.

The first phase of the purchase process is shown in figure 2.1. To start the process, the `ClientService`, from the `client` who placed the purchase, should receive a `PurchasePlaced` message. This will first, in the *domain logic* object of class `Client`, update the purchases list. It will then in *communication logic* create an ephemeral `ClientChildService` responsible for further processing this purchase. It then sends out a `PurchaseConfirmed` message to the `ProcessingService`. It is noted here that the `Forward Flow` pattern is in place by supplying the `replyTo` address of the ephemeral child. It is also noted here that, as is the case for more messages in this project, a prime alternative message `PurchaseConfirmedPrime` can also be sent out depending on if the requesting client is a prime member. Why a different message is used for `Prime` members will become clear when discussing the priority mailbox.

Once the `ProcessingService` receives a `PurchaseConfirmed` message, it also will create an ephemeral `ProcessingChildService` responsible for further handling this purchase. It will also message the `StockHouseManagerService` a `FindNearestStockHouses` request (prime alternative available). This request also contains the `replyTo` address for the just created ephemeral child, keeping in mind the `Forward Flow` pattern.

2.4 Aggregating items to fulfill purchases

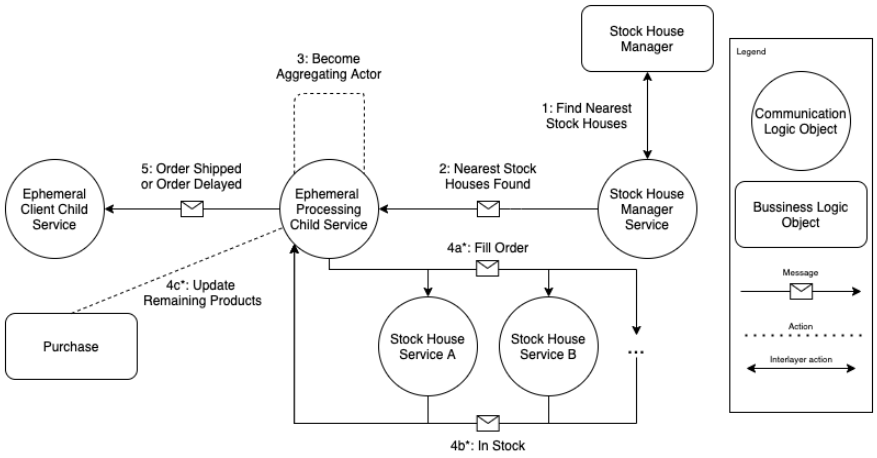

Figure 2.2: Simplified representation of aggregating phase of purchase process.

The aggregating phase of the purchase process is shown in figure 2.2. Once the nearest stock houses are found in *business logic*, the ephemeral `ProcessingChildService` receives a `Nearest-StockHousesFound` message. Upon receiving this message, the ephemeral child will change its behaviour by becoming an aggregating actor. This aggregating happens by sending out `FillOrder` messages (prime alternative available) to the found nearest stock houses until either:

- All items of the purchase are received through `InStock` messages. The ephemeral client child will receive a `OrderShipped` message.
- There are no stock houses to contact left and the order is not yet complete. The ephemeral client child will receive a `OrderDelayed` message.

It is noted that the sending out of the `FillOrder` messages happens incrementally rather than all at once. The latter is often the aggregator pattern approach but for simplification reasons, the incremental variant seemed fit. Some screenshots of this process are shown in figure 2.4.

2.5 Handling prime membership

Handling the requests of prime members with priority is done by using a custom made priority mailbox[1]: `PrimePriorityMailbox`. This mailbox simply ensures prime messages have a higher priority than other messages. It's noted that only messages which can be in an inbox with both prime and non-prime member messages require special attention. Indeed, prioritizing `InStock` messages, and others, for prime members has no benefit. They are sent to an ephemeral child of the `ClientService` which only receives messages for one `Client` and thus only prime or non-prime messages. Figure 2.5 shows the implementation of this mailbox.

[1] https://doc.akka.io/docs/akka/current/mailboxes.html

Extra figures

To make the report more readable some figures are not provided directly in the text. These figures are provided here.

Managing stock houses

```
StockHouse-X_1.1-Y_1.1 added!
StockHouse-X_2.2-Y_2.2 added!
StockHouse-X_3.3-Y_3.3 added!
StockHouse-X_4.4-Y_4.4 added!
StockHouse-X_5.5-Y_5.5 added!
StockHouse-X_6.6-Y_6.6 added!
StockHouse-X_7.7-Y_7.7 added!
StockHouse-X_8.8-Y_8.8 added!
StockHouse-X_8.8-Y_8.8 not added, the stock house most likely already existed
```

(a) The main loop adding stock houses.

```
StockHouse-X_1.1-Y_1.1: Added 3x NVIDIA GTX 3080
StockHouse-X_2.2-Y_2.2: Added 3x Apple iMac
StockHouse-X_3.3-Y_3.3: Added 3x HP Omen 15
StockHouse-X_2.2-Y_2.2: Added 4x Apple iPhone
StockHouse-X_3.3-Y_3.3: Added 4x NVIDIA GTX 3080
StockHouse-X_2.2-Y_2.2: Added 5x Samsung Galaxy S21 Ultra
StockHouse-X_3.3-Y_3.3: Added 5x Sony PS5
StockHouse-X_4.4-Y_4.4: Added 3x Apple MacBook Pro 15
StockHouse-X_4.4-Y_4.4: Added 4x Apple iMac
StockHouse-X_1.1-Y_1.1: Added 4x Sony PS5
StockHouse-X_4.4-Y_4.4: Added 5x Apple iPhone
StockHouse-X_1.1-Y_1.1: Added 5x Apple MacBook Pro 15
StockHouse-X_5.5-Y_5.5: Added 3x Samsung Galaxy S21 Ultra
StockHouse-X_5.5-Y_5.5: Added 4x HP Omen 15
```

(b) The main loop adding products to the stock houses.

Figure 2.3: The main loop managing the stock houses.

Purchase process

```
/user/Lennert] ClientService: received PurchasePlaced, registered purchase, created ephemeral child and sent PurchaseConfirmed.
/user/processingService] ProcessingService: received purchaseConfirmed, created ephemeral child and sent FindNearestStockHouses.
/user/processingService/$a] ProcessingChildService 9d93539c-768a-442b-837a-711c4cba97c6: Got nearest stock houses, became aggregating receiver and sent first FillOrder message.
/user/StockHouseManagerService/StockHouse-X_2.2-Y_2.2] StockHouse-X_2.2-Y_2.2: Checked available items in stock and provided them for client.
/user/processingService/$a] ProcessingChildService 9d93539c-768a-442b-837a-711c4cba97c6: got more InStock messages, updated information.
/user/StockHouseManagerService/StockHouse-X_4.4-Y_4.4] StockHouse-X_4.4-Y_4.4: Checked available items in stock and provided them for client.
/user/processingService/$a] ProcessingChildService 9d93539c-768a-442b-837a-711c4cba97c6: got more InStock messages, updated information.
/user/StockHouseManagerService/StockHouse-X_5.5-Y_5.5] StockHouse-X_5.5-Y_5.5: Checked available items in stock and provided them for client.
/user/processingService/$a] ProcessingChildService 9d93539c-768a-442b-837a-711c4cba97c6: got more InStock messages, updated information.
/user/StockHouseManagerService/StockHouse-X_1.1-Y_1.1] StockHouse-X_1.1-Y_1.1: Checked available items in stock and provided them for client.
/user/processingService/$a] ProcessingChildService 9d93539c-768a-442b-837a-711c4cba97c6: got more InStock messages, updated information.
/user/processingService/$a] ProcessingChildService 9d93539c-768a-442b-837a-711c4cba97c6: got all items, sending OrderShipped and terminating myself.
/user/Lennert/$a] ClientChildService 9d93539c-768a-442b-837a-711c4cba97c6: got OrderShipped message, I'm happy and terminate myself.
```

(a) A single purchase being processed successfully.

```
r/processingService] ProcessingService: received purchaseConfirmed, created ephemeral child and sent FindNearestStockHouses.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: Got nearest stock houses, became aggregating receiver and sent first FillOrder message.
r/StockHouseManagerService/StockHouse-X_1.1-Y_1.1] StockHouse-X_1.1-Y_1.1: Checked available items in stock and provided them for client.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: got more InStock messages, updated information.
r/StockHouseManagerService/StockHouse-X_3.3-Y_3.3] StockHouse-X_3.3-Y_3.3: Checked available items in stock and provided them for client.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: got more InStock messages, updated information.
r/StockHouseManagerService/StockHouse-X_4.4-Y_4.4] StockHouse-X_4.4-Y_4.4: Checked available items in stock and provided them for client.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: got more InStock messages, updated information.
r/StockHouseManagerService/StockHouse-X_2.2-Y_2.2] StockHouse-X_2.2-Y_2.2: Checked available items in stock and provided them for client.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: got more InStock messages, updated information.
r/StockHouseManagerService/StockHouse-X_5.5-Y_5.5] StockHouse-X_5.5-Y_5.5: Checked available items in stock and provided them for client.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: got more InStock messages, updated information.
r/processingService/$a] ProcessingChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: couldn't collect all items, sending OrderDelayed and terminating myself.
r/Lennert/$a] ClientChildService 7d7b0568-bc22-4f38-8dfe-60be36447eaf: got OrderDelayed message, I'm sad and terminate myself.
```

(b) A single purchase failing to be fulfilled.

```
/user/processingService/$e] ProcessingChildService cae27286-c398-4ea1-b6a7-93dd8c2c25f0: got more InStock messages, updated information.
/user/StockHouseManagerService/StockHouse-X_6.6-Y_6.6] StockHouse-X_6.6-Y_6.6: Checked available items in stock and provided them for client.
m/user/Bontinck/$a] ClientChildService f12cfac0-7491-47ff-bda8-9d299f0b4eb1: got OrderShipped message, I'm happy and terminate myself.
/user/processingService/$e] ProcessingChildService cae27286-c398-4ea1-b6a7-93dd8c2c25f0: got all items, sending OrderShipped and terminating myself.
/user/Coen/$a] ClientChildService cae27286-c398-4ea1-b6a7-93dd8c2c25f0: got OrderShipped message, I'm happy and terminate myself.
/user/processingService/$d] ProcessingChildService b3c41f57-38ad-45e8-adf9-bf5e7cc807b8: got more InStock messages, updated information.
/user/processingService/$c] ProcessingChildService f12cfac0-7491-47ff-bda8-9d299f0b4eb1: got all items, sending OrderShipped and terminating myself.
/user/StockHouseManagerService/StockHouse-X_5.5-Y_5.5] StockHouse-X_5.5-Y_5.5: Checked available items in stock and provided them for client.
m/user/processingService/$a] ProcessingChildService 82c2e86d-86ea-4591-a75c-060a8e32dce7: got all items, sending OrderShipped and terminating myself.
m/user/processingService/$d] ProcessingChildService b3c41f57-38ad-45e8-adf9-bf5e7cc807b8: got more InStock messages, updated information.
m/user/processingService/$d] ProcessingChildService b3c41f57-38ad-45e8-adf9-bf5e7cc807b8: got all items, sending OrderShipped and terminating myself.
m/user/Kris/$a] ClientChildService b3c41f57-38ad-45e8-adf9-bf5e7cc807b8: got OrderShipped message, I'm happy and terminate myself.
/user/Joske/$a] ClientChildService 82c2e86d-86ea-4591-a75c-060a8e32dce7: got OrderShipped message, I'm happy and terminate myself.
```

(c) Many purchases with many different products being processed concurrently.

Figure 2.4: The multiple variants of the demo behaviour available in the main loop.

Custom priority mailbox

```scala
/** Custom mailbox to use so that prime messages are prioritized.
 * Inspiration from: https://doc.akka.io/docs/akka/current/mailboxes.html */
class PrimePriorityMailbox(settings: ActorSystem.Settings, config: Config) extends UnboundedPriorityMailbox(
  // Create a new PriorityGenerator, lower priority means more important
  PriorityGenerator {
    // Prime messages have highest priority (priority 0)
    case msg: FillOrderPrime => 0
    case msg: FindNearestStockHousesPrime => 0
    case msg: PurchaseConfirmedPrime => 0
    // All other messages are treated equally but slower (priority 1)
    case _ => 1
  })
```

Figure 2.5: Implementation of `PrimePriorityMailbox`.

References

Bontinck, L. (2021). *Assignment 3 of software architecture course* [GitHub commit: XXX]. Retrieved August 6, 2021, from https://github.com/pikawika/VUB-SA-assignment-3

De Smet, R. (2020). *Vub latex huisstijl* [GitHub commit: d91f55...]. Retrieved November 2, 2020, from https://gitlab.com/rubdos/texlive-vub