Karlsruhe Institute of Technology

The CoCoME Platform for Collaborative Empirical Research on Information System Evolution

Technical Report

Robert Heinrich, Niko Benkler, Tobias Haßberg, Ralf Reussner

May 16, 2018

Contents

List of Figures

1 Introduction

2 Evolution Scenarios

We implemented distinct evolution scenarios covering the categories adaptive and perfective evolution. Corrective evolution is not considered in the scenarios as this merely refers to fixing design or implementation issues.

2.1 Evolution Scenarios of the Hybrid Cloud-based Variant

This section introduces the two evolution scenarios of the hybrid cloud-based variant of CoCoME.

2.1.1 Setting up a Docker environment

The CoCoME company must reduce IT administration costs but frequent updates to the enterprise and store software are necessary to continuously improve the entire system. As a consequence, IT staff need to update the system components as soon as a new software version is released. An Operations Team member has to get access to the actual server in order to undeploy the old version and replace it with the new one. This is time consuming and expensive as the updates have to be done manually.
Therefore, a Docker version is elaborated to simplify the administration process. As soon as a new software version of CoCoME is ready for delivery, the Development Team wrap it into a Docker Image. This Image can be automatically deployed to the destination server according to the principle of Continuous Deployment (CD) [2].

2.1.2 Adding a Mobile App

After successfully adding a Pick-up Shop, the CoCoME company stays competitive with other online shop vendors (such as Amazon). In times of smartphones, customer do not only want to buy exclusively goods from their home computers. Purchasing goods 'on the way' comes more and more into fashion. This raises the idea to create a second sales channel next to the existing Pick-up Shop in the CoCoME system. As a consequence, more customers can be attracted to gain a larger share of the market.
The customer can order and pay by using the app. The delivery process is similar to the Pick-up Shop: The goods are delivered to a pick-up place (i.e. a store) of her/his choice, for example in the neighbourhood or the way to work. By introducing the Mobile App as a multi OS

application, the CoCoME system has to face various quality issues such as privacy, security and reliability. Also the performance of the whole application can be affected if many customers order via the app.

2.2 Evolution Scenarios of the Microservice-based Variant

This section introduces the evolution scenario of the Microservice-based variant of CoCoME.

2.2.1 Defining different Microservices

After a year of economical stagnation, the CoCoME company decided to restructure its infrastructure. Global players like Amazon or Netflix demonstrated that using a Microservice Architecture makes them more flexible regarding new functionality. When adding the Pick-up Shop, the CoCoME company realized that they have to break open the existing system. It was necessary to modify the *WebService::Inventory* and the *TradingSystem::Inventory* component [1]. Furthermore, adding a *MobileAppClient* demonstrated that the SOAP/WS*-based web services provided by CoCoME are not compatible with REST-based App development.
Inspired by the flexibility and reusability of Microservices, the CoCoME company decided to invest money into a restructuring process. The current system is divided into a collection of loosely coupled services. Each of them cover a specific part of the former CoCoME system. The aim is to preserve the functionality of the current system and solely change its architecture.
This enables the company to develop new markets much easier and therefore secures the future competitiveness. For example, the CoCoME company wants to extend their product range by offering movie streaming. This requires a vastly different system. Nevertheless, Customer Management like login and means of payment are identical to the former CoCoME system. Those components already exists as a Microservice a therefore can be taken over. The management is certain that this will soon result in economical growth.

3 Design Details for Evolution Scenarios

In this chapter we provide the detailed design documentation for each of the evolution scenarios introduced in the prior section. Sec. 3.1 sketches the design decision for the Mobile App that provides a second sales channel next to the existing Pick-up Shop. Sec. 3.2 describes the adaptive changes of setting up a Docker environment to simplify the update process. They are both based on, or at least use the Hybrid Cloud-based Variant of CoCoME [1]. In contrast, Sec. 3.3 provides a detailed design documentation of a new architectural version of CoCoME. This perfective evolution scenario is realized based on the Microservice idea.

3.1 Adding a Mobile App Client

Developing the Mobile App Client as an extension of CoCoME requires additional use cases. These are described in Sec. 3.1.1. Sec. 3.1.2 describes extensions on design level. The Mobile App Client is using the hybrid cloud-based variant of CoCoME but does not require any changes of CoCoME itself. The content of this chapter mainly originates from [3].

3.1.1 Use Cases of the Mobile App

UC 14 - ProcessAppSale
Brief Description A Customer selects the product items s/he wants to buy and the payment by credit card is performed.

Involved Actors AppCustomer, Bank

Precondition The App is ready to process a new sale and the Customer already has an account registered in the System.

Trigger The Customer opens the app and wants to buy product items.

Postcondition The Customer has paid and the sale is registered in the inventory.

Standard Process

1. The AppCustomer searches products provided by the App.
2. The AppCustomer can see details for each product on a separate site.
3. The AppCustomer adds the product items s/he wants to purchase to the Shopping Cart by clicking the Add to Cart button. Step 1-3 is repeated until all items are added to the cart.

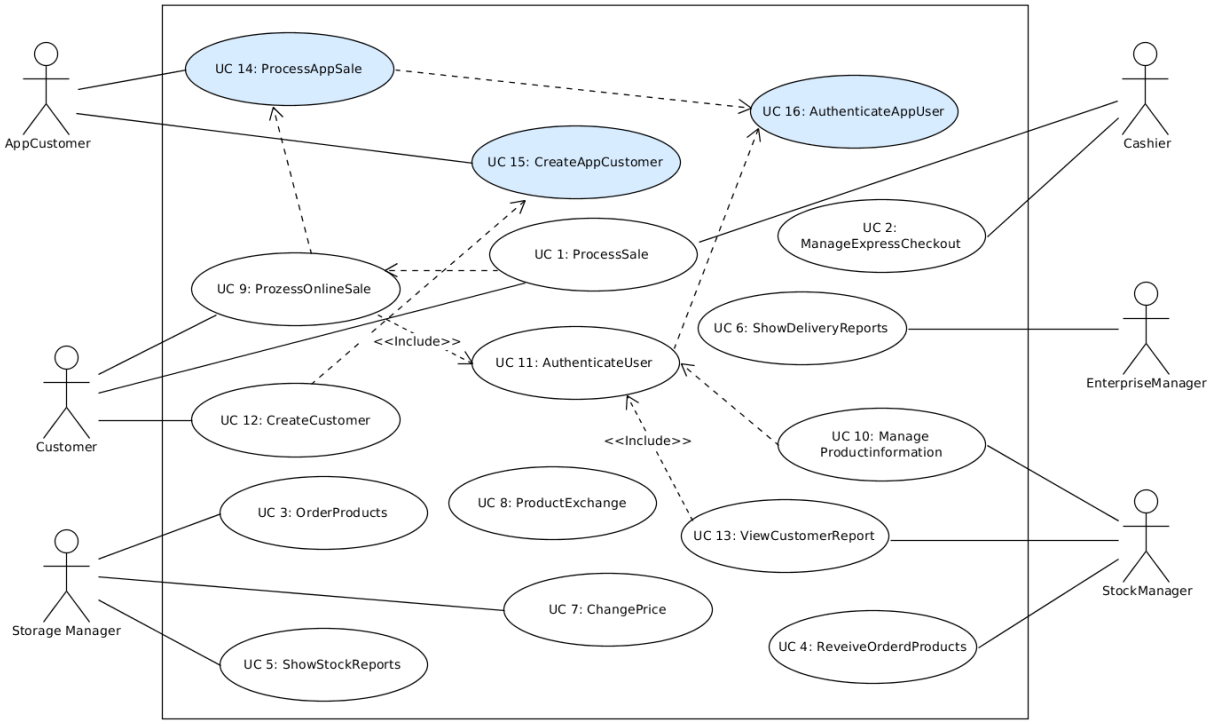

Figure 3.1: Use Case Diagram CoCoME Mobile App

4. The System presents the Customer with the item names, prices and the running total.
5. Denoting the end of adding items, the Customer presses the Proceed to checkout button.
6. The AppCustomer selects the Store where he wants to pick up his purchased product items. A mobile Application proof of concept 3
7. The AppCustomer is presented with a login form and is required to complete the "Authenticate user" use case.
8. In order to initiate card payment, the Customer selects a credit card used for the purchase.
9. The AppCustomer enters his PIN in the designated field presented by the System.
10. The System presents the Customer with an overview of the purchase, the AppCustomer confirms the purchase and waits for validation. Step 9 is repeated until the validation is successful or the Customer decides to cancel the purchase.
11. Completed sales are logged by the System and sale information are sent to the Inventory in order to update the stock.
12. A success message is presented to the AppCustomer and the product items are being prepared to be picked up by the customer.
13. The AppCustomer closes the app.

Alternative or Exceptional Processes

- In step 8: No Card available

1. In order to add a new credit card the Customer clicks the Add Card button.
2. The Customer enters the card number of the new credit card and saves the card.

- In step 10: Card validation fails
  1. The Customer tries again and again.
  2. Otherwise, the Customer can decide to cancel the purchase.

UC 15 - CreateAppCustomer

Brief Description The app provides the opportunity to create a new Customer account.

Involved Actors AppCustomer

Precondition The Customer does not have a Customer account yet and the app is started.

Trigger A new AppCustomer wants to create an account.

Postcondition The User is authenticated.

Standard Process

1. The app presents the Customer a 3 step wizard with forms to fill out, requesting all necessary information to create a new Customer account.
   (a) Form for name, email and password
   (b) Form for address
   (c) Summery of the information and a submit button
2. The Customer fills out the forms, checks the information and submits the information.
3. The app checks the provided information and creates a new Customer account in the Inventory.

Alternative or Exceptional Processes

- In step 3 : Provided information is incorrect or not valid. The Customer is notified of the problem and enters the information again until it passes the check.

UC 16 - AuthenticateAppUser

Brief Description The app provides the opportunity to authenticate a User.

Involved Actors AppCustomer

Precondition The app is started.

Trigger A User wants to authenticate himself.

Postcondition The User is authenticated.

Standard Process

1. The AppCustomer is presented with a login form and enters his email and password.
2. The app checks the provided credentials and logs the User in.

Alternative or Exceptional Processes

- In step 2: Wrong credentials
    1. The User is presented with an error message.
    2. The User may try again until the authentication succeeds.

3.1.2 Design of the Mobile App

In Fig. 3.2, the component diagram for this evolution scenario is shown. For adding the Mobile App client, the hybrid cloud-based variant of CoCoME did not have to be modified. Therefore, CoCoME is encapsulated in a single component. Simply the three web services *WebService::Inventory::LoginManager, WebService::Inventory::Store and WebService::Inventory::Enterprise* used by the App Client are emphasized. The entire component diagram for the hybrid cloud-based variant is available in the Technical Report [1].
Fig. 3.2 indicates that the AppShop requires an adapter to access the web services provided by CoCoME. This is because CoCoME uses SOAP/WS*-based web services which are not compatible with the technology used to implement the AppShop Client. A more detailed introduction about the technology used to implement the Mobile App Client can be found in [3].

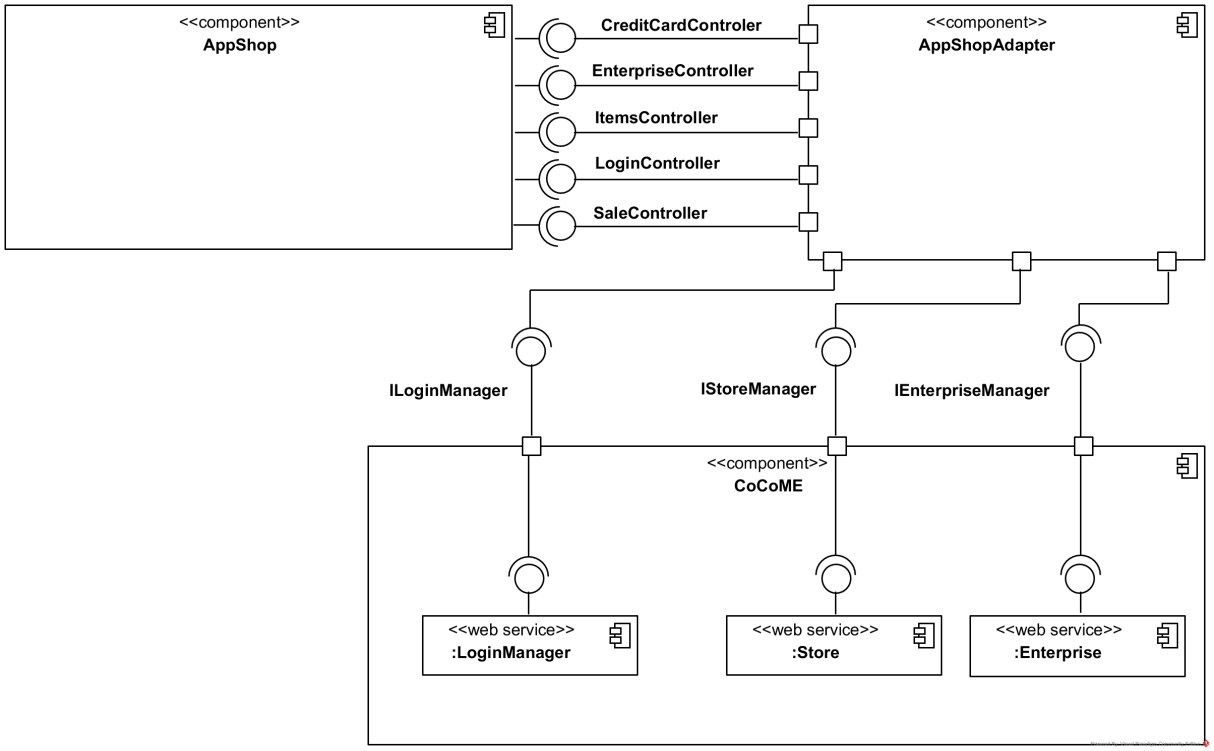

Figure 3.2: Component Diagram of the CoCoME Ecosystem After Adding the Mobile App Client

The *AppShopAdapter* consumes the three web services *WebService::Inventory::LoginManager, WebService::Inventory::Store* and *WebService::Inventory::Enterprise* and provides a Rest Api which is used by the actual *AppShop*. This Rest Api contains endpoints to retrieve and process Credit Card, Enterprise and StockItem information. To implement *UC14-16*, the Api also provides endpoints for user management and processing sales.

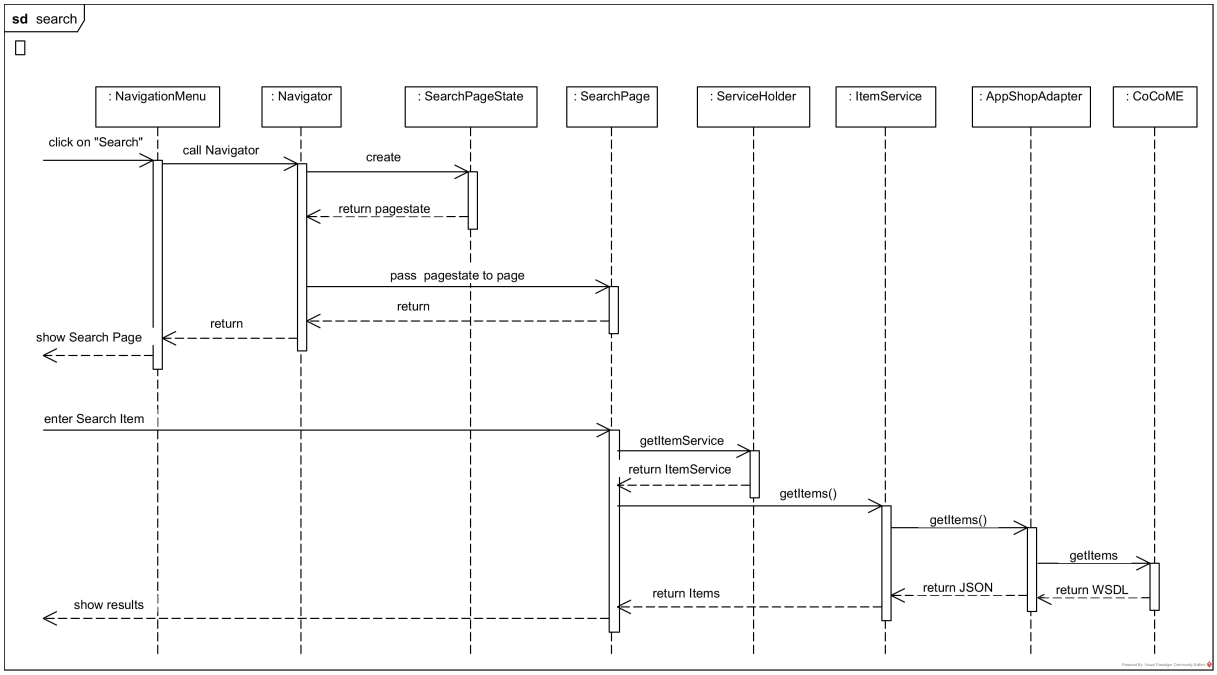

Figure 3.3: Sequence Diagram of Searching an Item in the Mobile App Client

Fig. 3.3 shows the process of opening a page to search for an Item. The customer opens the *WebShopClient* and clicks on the "Search" button to search for an item. To open the page, the *NavigatorMenu* must call the *Navigator* which creates a pagestate object and passes the object to the page. This HTML page is now presented to the customer. To fill the page with information, i.e when searching for a *ProductItem*, the page uses services provided by the *ServiceHolder*. In this case, the *ItemService* calls the responsible REST-Service of *AppShopAdapter* which in turn retrieves the necessary information from the WSDL services provided by CoCoME.

Fig. 3.4 demonstrates how the Mobile App Client processes sales. First, the customer searches for items (according to Fig. 3.3). By clicking on the desired Item, the according *ItemPage* is shown. This page carries information about the Item. Here, the customer decides whether the Item should be added to the Shopping Cart or not. The last steps are repeated until the customer decides to proceed to the checkout. If not logged in, the customer gets forwarded to the *LoginPage*. When successfully logged in, the customer clicks the *BuyNow*-Button. The Sale process is finished as soon as the backend (CoCoME) has processed the sale.

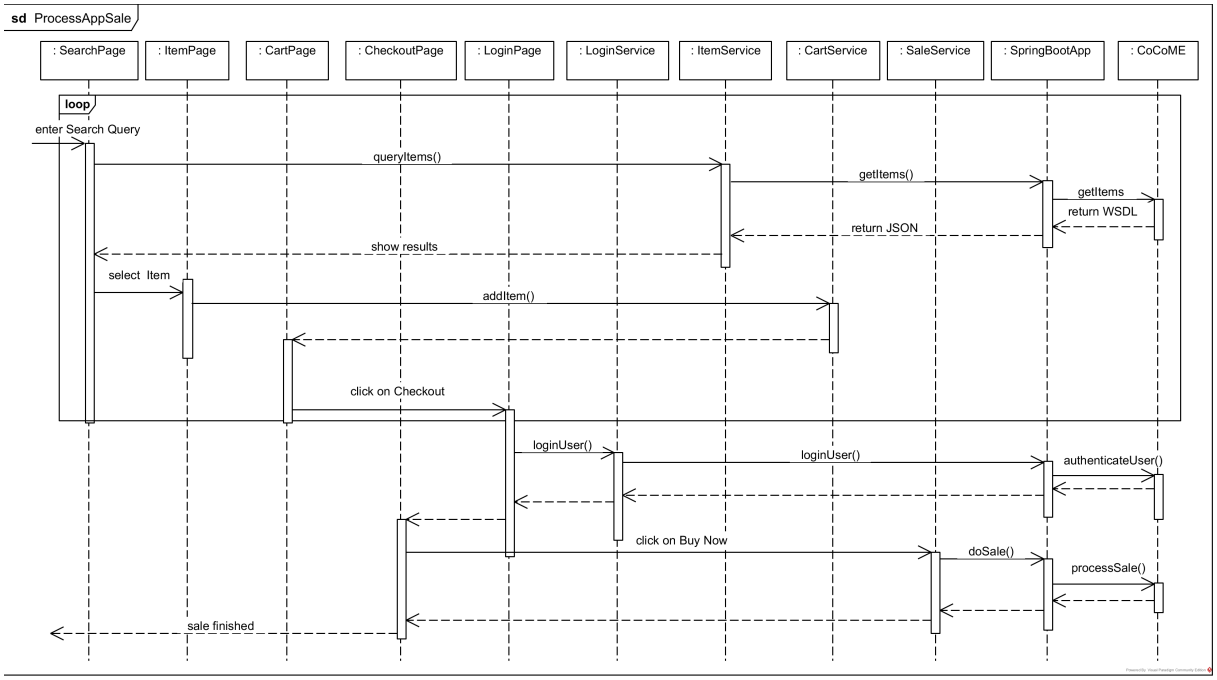

Figure 3.4: Sequence Diagram of Processing a Sale

3.2 Setting up a Docker environment

As shown in Fig. 3.5, the changes carried out when setting up a Docker environment are affecting the technology stack by adding additional layers. More detailed, the given CoCoME Stack is moved into the Docker Deamon, which runs a Linux distribution.The original parts of the stack, like Glassfish and the Java Virtual Machine, are still a part of the stack.

The Dockerfile defines an environment based on the latest version of Ubuntu 16:04. Onto it there is installed Maven, Git and Java by using the Ubuntu package manager.
Git has two purposes: On the one hand it is used to download the most recent version of CoCoME. On the other hand, it is used to download a prefabricated version of Glassfish that already includes domains and other adjustments required for CoCoME. Java is required by Glassfish and CoCoME as they need the Java Virtual Machine. Maven is needed to deploy the latest version of CoCoME onto the provided Glassfish servers.

During the development, it was decided to implement and provide two different versions. The first version always pulls the most recent CoCoME source code from GitHub, downloads the entire dependencies with maven, compiles and builds the project and finally, deploys CoCoME on the Glassfish servers. . As a consequence. creating and starting a Docker Container takes about one hour.
In contrast, the second version only pulls a prefabricated version of CoCoME from GitHub. Therefore, pulling the source code up to building the project is skipped. As a consequence, Maven does not have to be included in the technology stack. Solely, deploying CoCoME on the

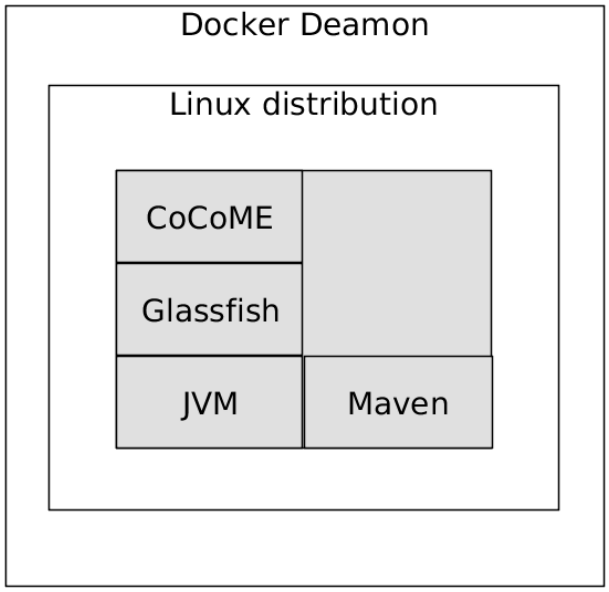

Figure 3.5: Extended technology stack CoCoME

glassfish server is necessary.
This reduces the deployment time to a few minutes but has a disadvantage: The prefabricated version is updated manually. Therefore, it is sometime not the most recent version.
By providing both, a fast deploying version and a current version, the user can choose what's the best for its situation.

3.3 Using Microservices Technology

Info teils aus [4]

- je microservice absatz mit entsprechenden Sequenzendiagram
- frontend? muss dazu auch das gemacht werden?
- ein blocktext zu mehereren diagrammen oder diagramme zwischen text?
- je die einzelnen module und szenarien erlaeutern in dem diese sinnvoll sind

3.3.1 Products

abstrahieren der Produktinformationen

3.3.2 Stores

einzelne Laeden alleinstehend abbilden um nach bedarf neue microservices alias laeden starten zu können

3.3.3 Enterprise

aehnlich zu stroes

3.3.4 Reports

stellt alleinigen aufgaben bereich dar, entsprechen undabhaengig darzustellen von anderem.

4 Implementation of Evolution Scenarios

This chapter describes implementation details for realizing the Docker environment 4.1, the Mobile App Client for the existing hybrid cloud-based variant of CoCoME (4.2) and the Microservice-based variant (4.3).

4.1 Docker

As shown in figure 4.1 the docker Container contains five different Glassfish servers. In particular they are called *WEB*, *ENTERPRISE*, *STORE*, *REGISTRY* and *ADAPTER* and correspond to the given by the CoCoME deplyoment setup. By default, Glassfish provides a Derby DB that is connected to the server Adapter using Java Database Conectivity (JDBS) interface.

CoCoME is deployed inside the docker container on the same way it is usually deployed. This means the maven generated archive files *cloud-web-frontend*,*enterprise-logic-ear*,*store-logic-earcloud-registry-sevice* and *service-adapter-ear* are deployed to the servers with the following assignment:

Server	Deployment file
WEB	cloud-web-frontend
ENTERPRISE	enterprise-logic-ear
STORE	store-logic-ear
REGISTRY	cloud-registry-service
ADAPTER	service-adapter-ear

Figure 4.2: Assignment of archive files to Servers

4.2 demonstrates the assignment between the archive files and the servers as it is implemented and also recommended by the CoCoME deployment guide. This information is also represented in Fig. 2. As mentioned earlier, there are two versions of this Docker project. Both deploy the CoCoME main program with this assignment.

In addition, the fast version can be extends by the pickup shop[1]. This pickup-shop runs inside a separate container which is shown in figure 4.3. As shown in figure 4.3, this container

[1] https://github.com/cocome-community-case-study/cocome-cloud-jee-web-shop

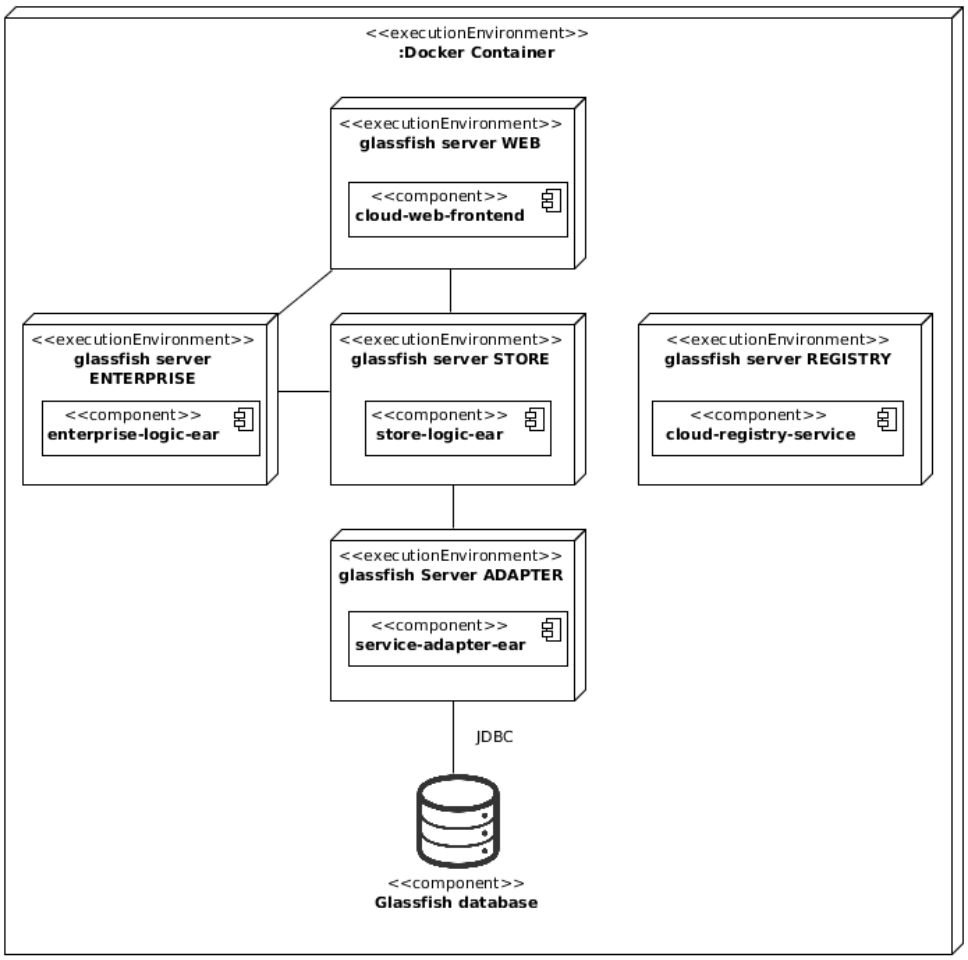

Figure 4.1: Deployment diagram CoCoME

provides only one Glassfish server.

Server	Deployment file
PICKUP_SHOP	cocome-pickup-war

Figure 4.4: Assignment archive files to Servers

To control the start of both containers, precisely the CoCoME and the Pick Up Shop, another specific file is needed: the Docker Compose file. It ensures that the CoCoME Container is active, before the pickup-shop container is starting. This is necessary as the Pickup Shop requires a running instance of CoCoME to register itself.
Also CoCoME runs without the pickup-shop, the pickup-shop does not work without an running instance of CoCoME.
Both containers need to communicate with each other. By default, docker prohibits any outgoing and ingoing communication from an in a container. This is solved by opening specific

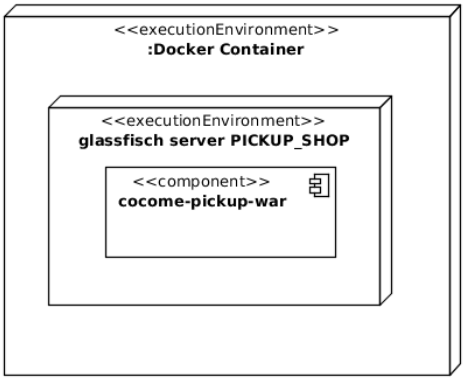

Figure 4.3: Deployment diagram CoCoME Pickup Shop

ports through which the communication is possible. Which ports the containers can use is specified in the Docker Compose file as well.

4.2 Adding a Mobile App Client

Adding a Mobile App Client did not require a modification within the hybrid cloud-based variant of CoCoME [1]. The implementation was done using the Cordova framework and OnsenUI to provide a multi OS compatible backend and UI [3]. The App itself is written in Typescript/Javascript. Fig. 4.5 shows the principal classes and their relationships.
The *Navigator* is the primary class that manages the pages. The pages consist of two components: The *Page* itself and its *PageState*. The *PageState* is used to store and transfer the current status of a page. There are currently six different pages available: *IndexPage*, *SearchPage*, *ItemPage*, *CheckoutPage*, *CartPage* and *LoginPage*. For the sake of clarity, they are subsumed under the generic terms *ConcretePage* and *ConcretePageState*.
Pages use components. Such components are i.e. the *Navbar* or the *Searchbar*. These components are abstract descriptions of UI elements that are connected to the actual *HTML-elements* via Knockout.js. By using Knockout.js, changing values of a component results in an immediate change of the UI. Besides, the App Client retrieves information of the CoCoME system. As mentioned in 3.1.2, the Client is not able to access the CoCoME system directly. Therefore, the pages use *Services* provided by a *ServiceHolder* to call the *AppController'* Rest-API. The *AppController* is written in Java using the SpringBoot framework and converts the Rest-requests of the App Client to SOAP-Requests in order to match the CoCoME-API.

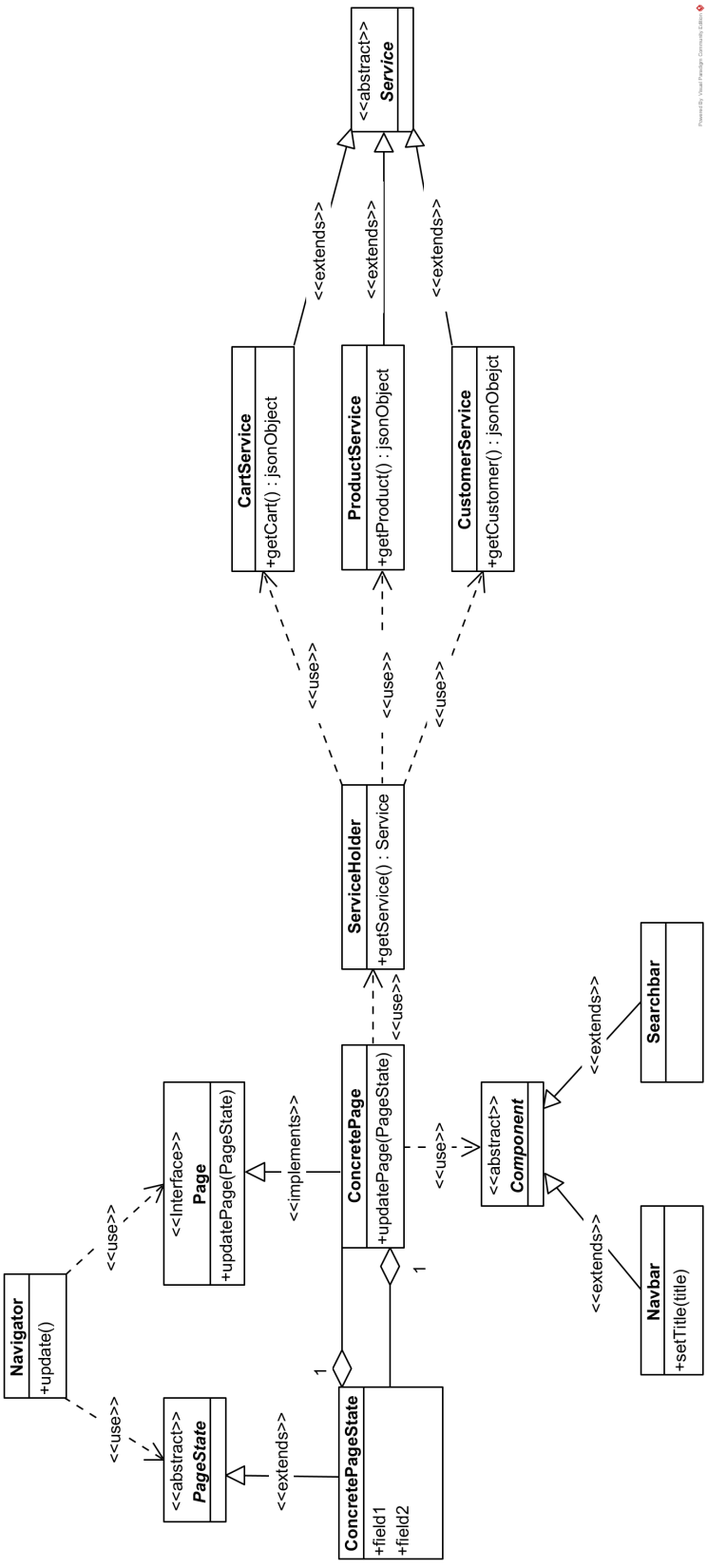

Figure 4.5: Primary Classes of the App

4.3 Using Microservice Technology

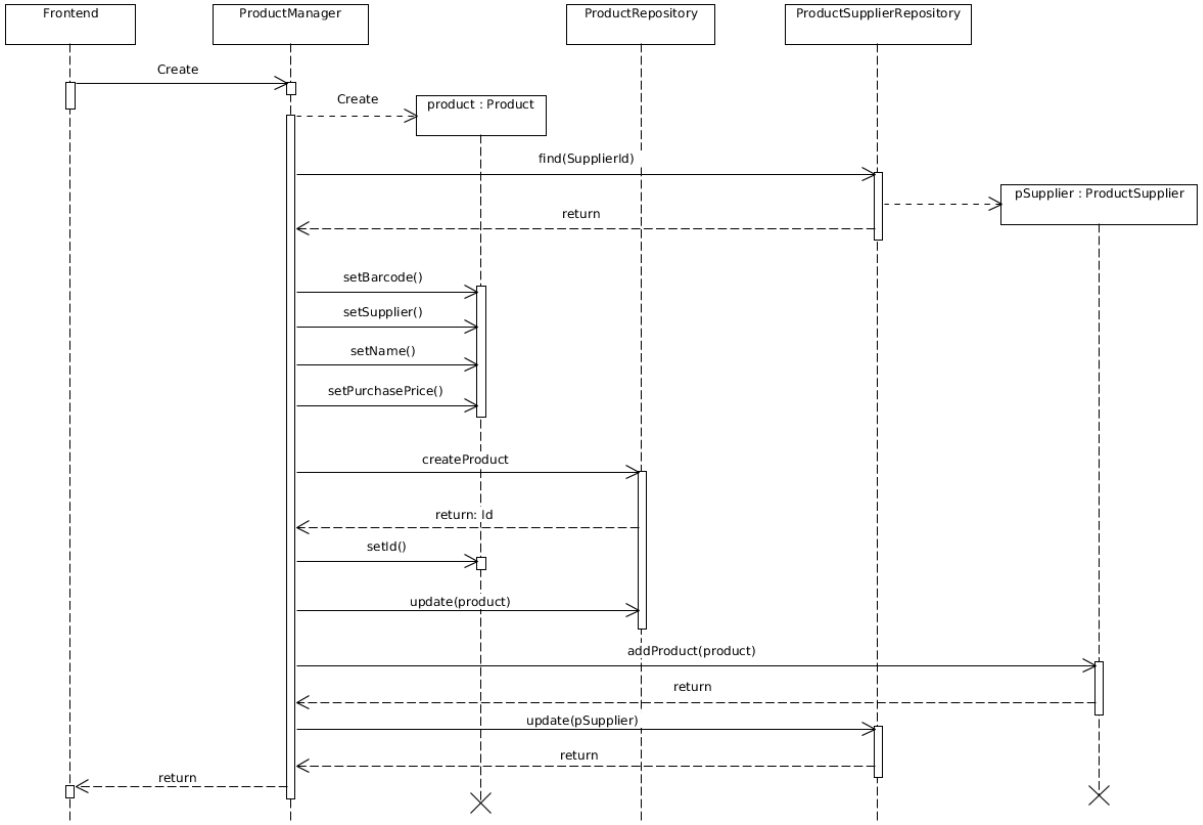

Figure 4.6: Sequenze Diagram for creation of new Products

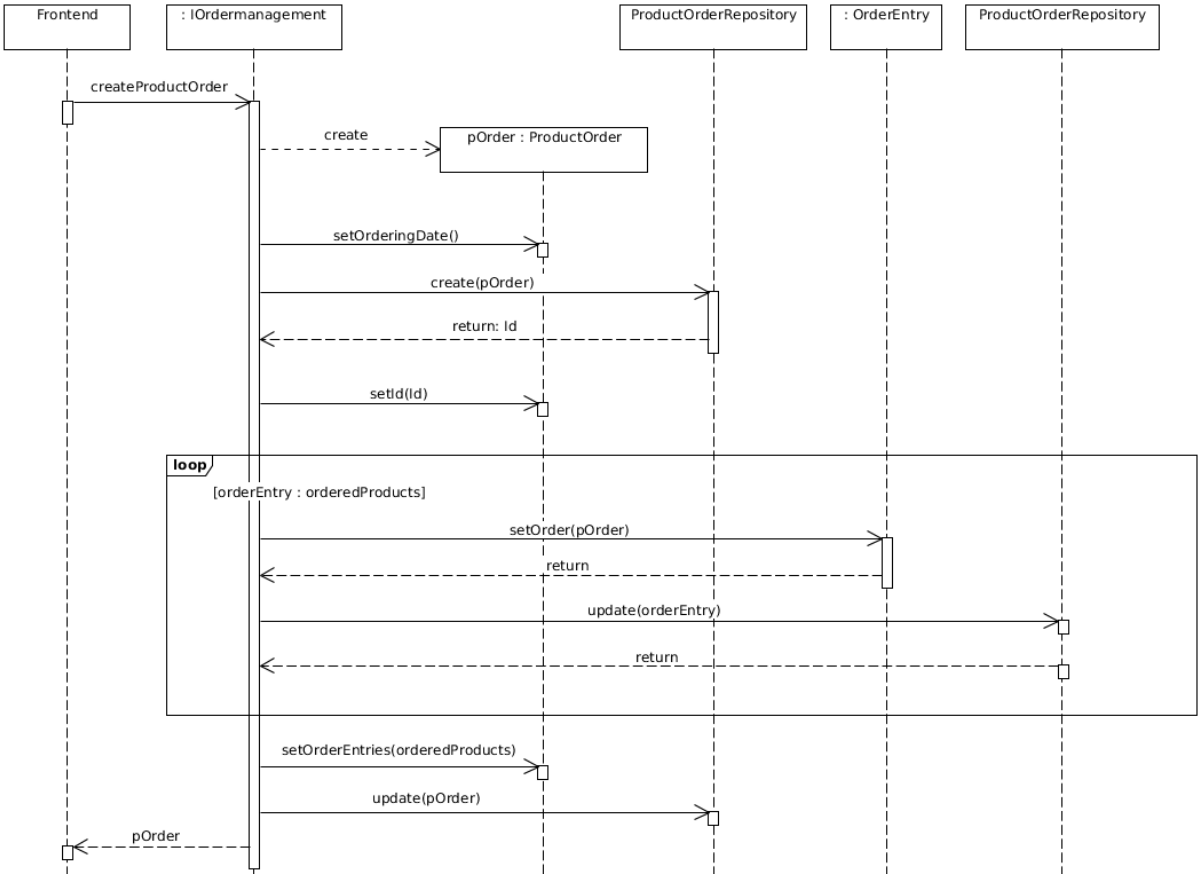

Figure 4.7: Sequenze Diagram for creation of new Product Orders

5 Conclusion

Bibliography

[1] R. Heinrich, K. Rostami, and R. Reussner. The cocome platform for collaborative empirical research on information system evolution. Technical Report 2, 2016.

[2] H. H. Olsson, H. Alahyari, and J. Bosch. Climbing the" stairway to heaven"–a mulitiple-case study exploring barriers in the transition from agile development towards continuous deployment of software. In *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, pages 392–399. IEEE, 2012.

[3] J. Schnabel. Mobile application client for a cloud based software system. https://github.com/cocome-community-case-study/cocome-cloud-jee-app-shop/blob/master/doc/SQEE1617_MobileApp_Schnabel.pdf.

[4] N. Sommer. Erweiterung und wartung einer cloud-basierten jee-architektur. https://github.com/cocome-community-case-study/cocome-cloud-jee-microservices-rest/blob/master/doc/report.pdf.