2-11-2017

Software Design Description

Jordi van Velzen, Timo van den Hazel

Inhoudsopgave

[1. Inleiding 2](#_Toc497342608)

[2. Systeem structuur 3](#_Toc497342609)

[3. Detailed Design Description 4](#_Toc497342610)

[3.1. Design class diagram 4](#_Toc497342611)

[3.2. Registreren 5](#_Toc497342612)

[3.3. Berekenen kosten 6](#_Toc497342613)

[3.4. Reserveren auto 8](#_Toc497342614)

[3.5. Gebruiken auto 10](#_Toc497342615)

[3.6. Klantgegevens beheren 11](#_Toc497342616)

[3.6.1. Aanmaken klantgegevens 11](#_Toc497342617)

[3.6.2. Bekijken klantgegevens 12](#_Toc497342618)

[3.6.3. Klantgegevens wijzigen 13](#_Toc497342619)

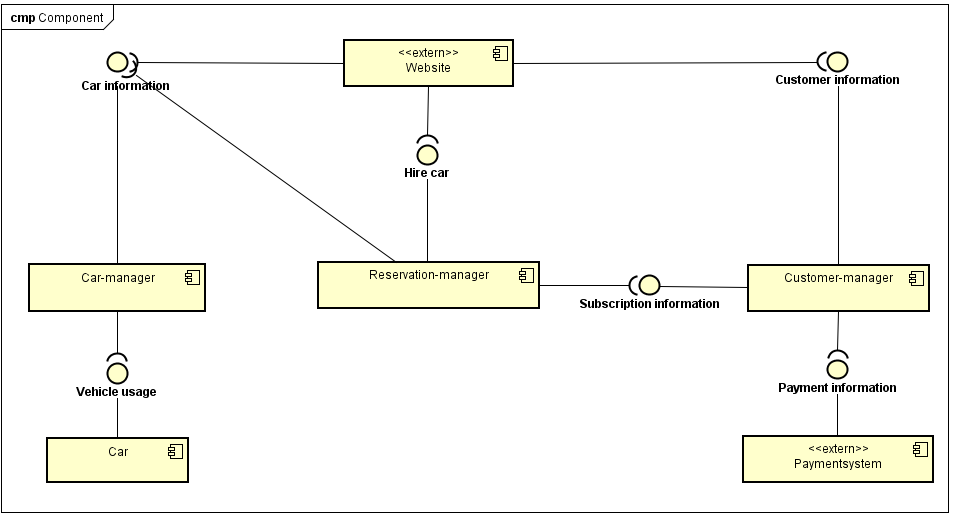
[3.6.4. Klantgegevens verwijderen 14](#_Toc497342620)

# 1. Inleiding

Dit document beschrijft de software architectuur en implementatie om aan te geven hoe het systeem in elkaar zit.

# 2. Systeem structuur

In onderstaand component diagram is de structuur van ons systeem te zien. We maken gebruik van drie verschillende managers die, waar nodig, met elkaar communiceren. De website en het betalingssysteem zal door externe partijen gemaakt worden.



# 3. Detailed Design Description

Om de structuur van het systeem in detail weer te geven hebben we een aantal diagrammen gemaakt. Deze worden geordend per Use case.

## 3.1. Design class diagram

Het Design class diagram is dusdanig groot dat het niet leesbaar op een afbeelding past. Het volledige diagram is in de inlever map als PNG bestand te vinden.

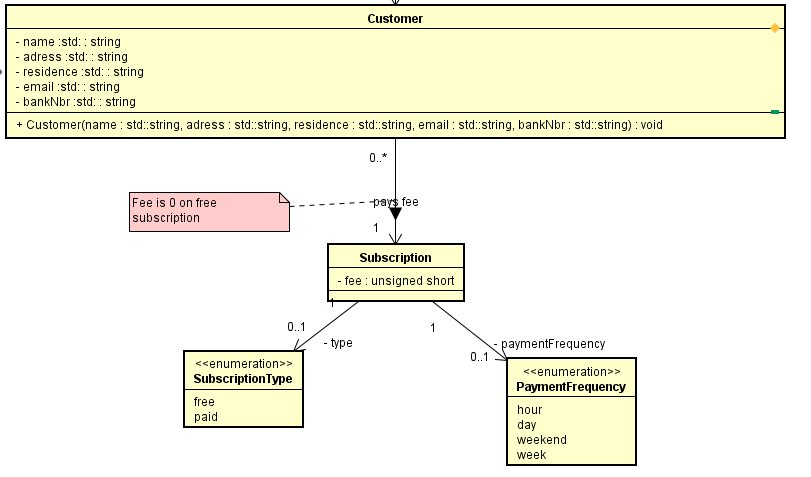
### 3.1.1. Aanpassingen

#### De type abonnementen moeten makkelijk uitbreidbaar zijn

In de eerste iteratie hebben we het probleem opgelost door een enum te gebruiken voor het type van een abonnement. In de tweede iteratie geven wij de voorkeur aan het implementeren van een parent classe Subscription met de type subscriptions als child classes. Toegepast op onderstaand scenario ziet het er dan als volgt uit: Een parent classe Subscription, met child classes FreeSubscription en PaidSubscription.

#### Payment Frequency

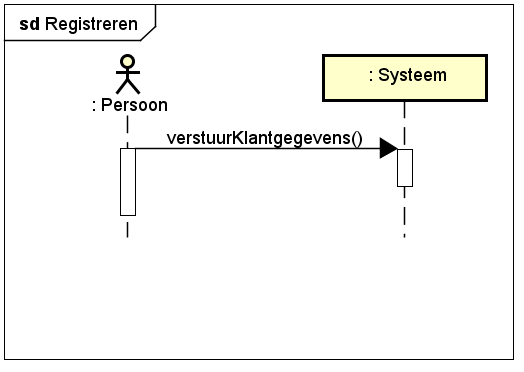
De Payment Frequency (Of je per uur, per dag, per week, etc… wil betalen voor de gereserveerde auto) verplaatsen we van Subscription naar Reservation.



## 3.2. Registreren

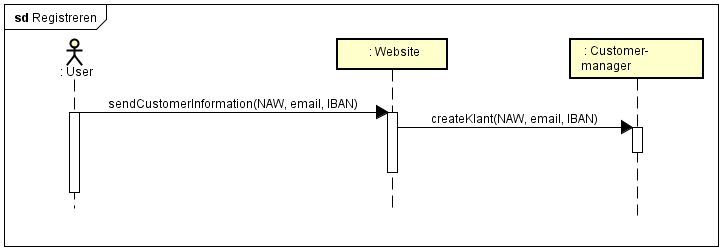
Nadat op de website alle benodigde informatie is ingevuld wordt deze naar het systeem opgestuurd en daar opgeslagen.

#### System sequence diagram

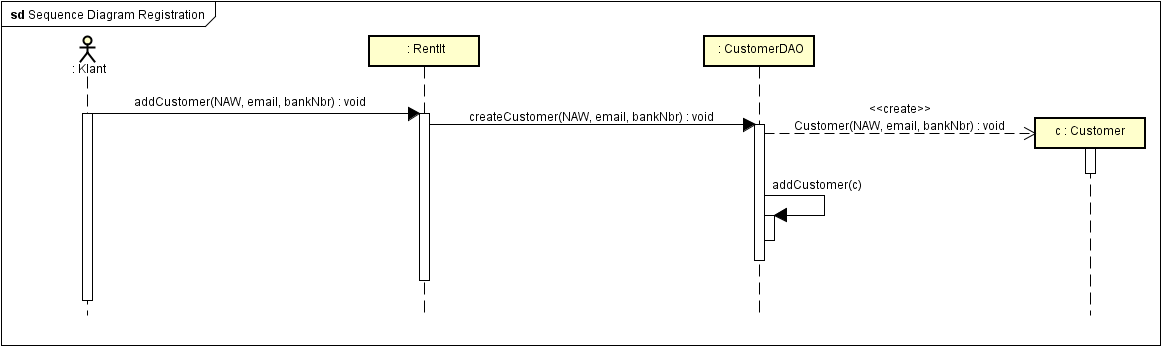


#### Component sequence diagram

De data van de website moet ergens opgeslagen worden, dit wordt in Customermanager geregeld.



#### Sequence diagram

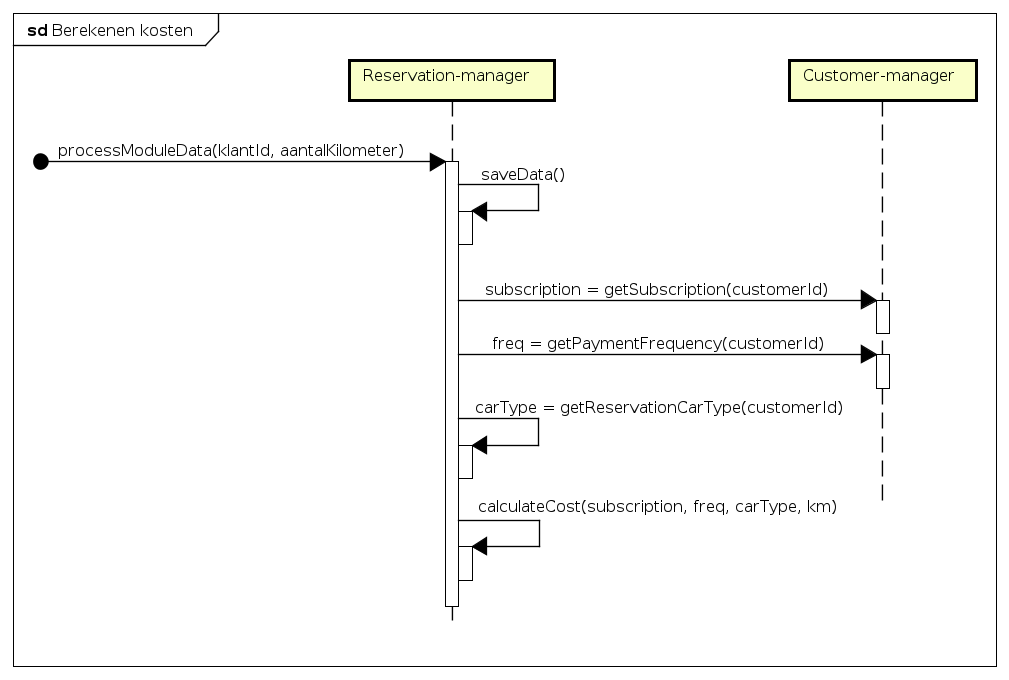
Het opslaan van de gegevens (in ons geval het aanmaken van een nieuwe Customer en deze in de mock-database stoppen) wordt geregeld door de CustomerDAO. Deze beheert al instanties van Customer en is daarom hier de Creator.

## 3.3. Berekenen kosten

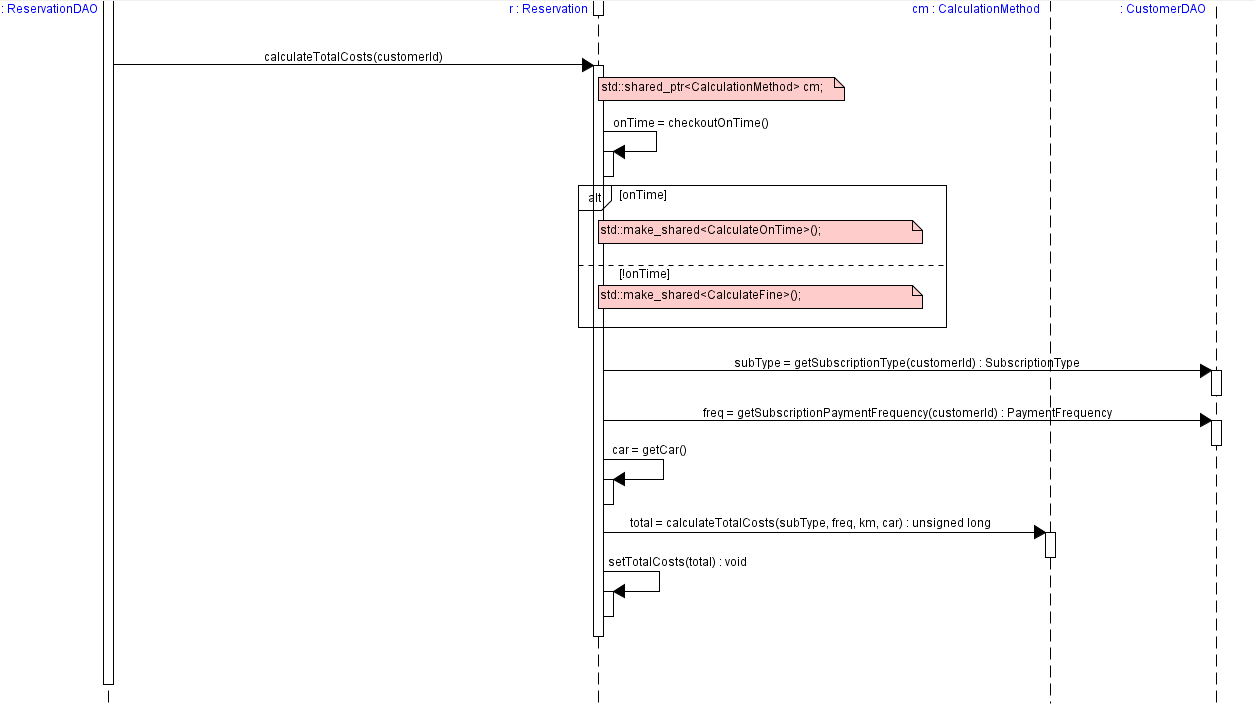
Het berekenen van de kosten wordt niet direct door een actor aangeroepen, maar door een actie van een actor, daarom is er geen System sequence diagram. De actie kan een van de volgende drie scenario’s zijn:

* De klant heeft ingecheckt en checkt nu uit.
* De klant heeft niet ingecheckt en de gereserveerde tijd is verlopen.
* De klant heeft ingecheckt, maar niet uitgecheckt en de gereserveerde tijd + een bepaalde wacht periode is verlopen.

#### Component sequence diagram

De gegevens van de module worden opgeslagen, daarna worden de totale kosten berekent.

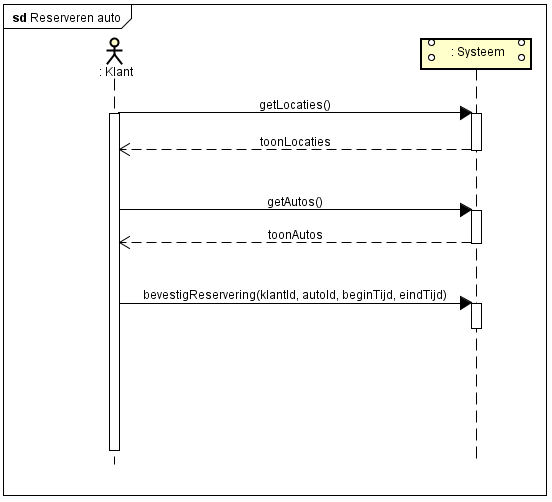
#### Sequence diagram

Het bereken van de kosten is hier verder uitgewerkt. Afhankelijk van of er op tijd uitgecheckt is wordt de standaard of de boete berekening gebruikt. Hierbij wordt gebruik gemaakt van het strategy pattern zodat de berekeningen makkelijk uitbreidbaar zijn.

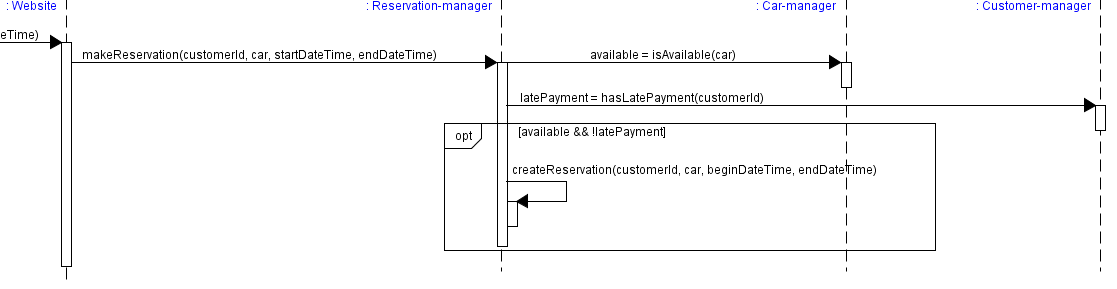
## 3.4. Reserveren auto

#### System sequence diagram

De klant vraagt om de locaties van alle auto’s en selecteert er een. Deze keuze wordt naar het systeem doorgestuurd en daar verwerkt.

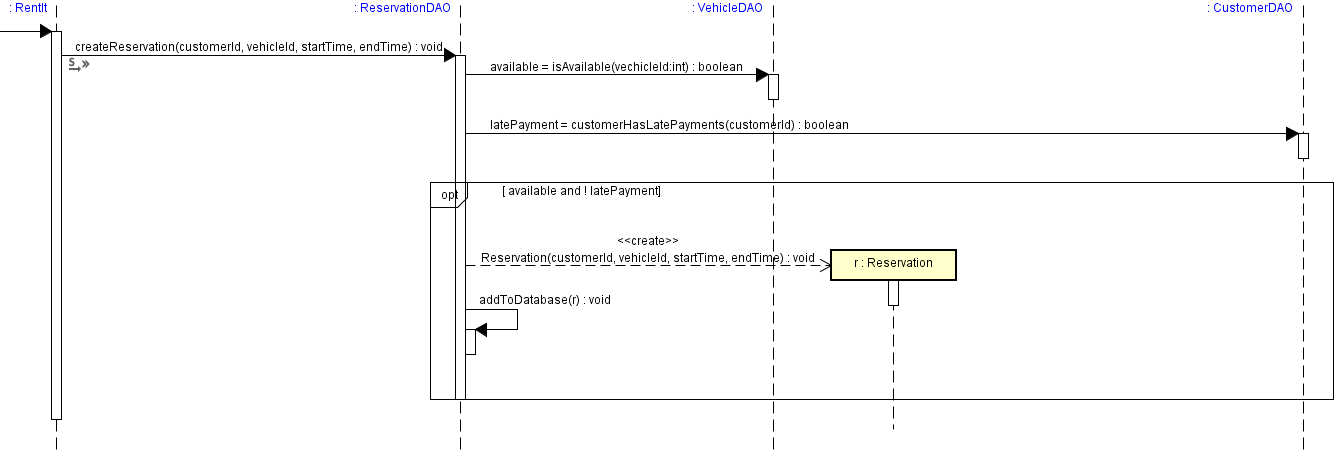


#### Component sequence diagram

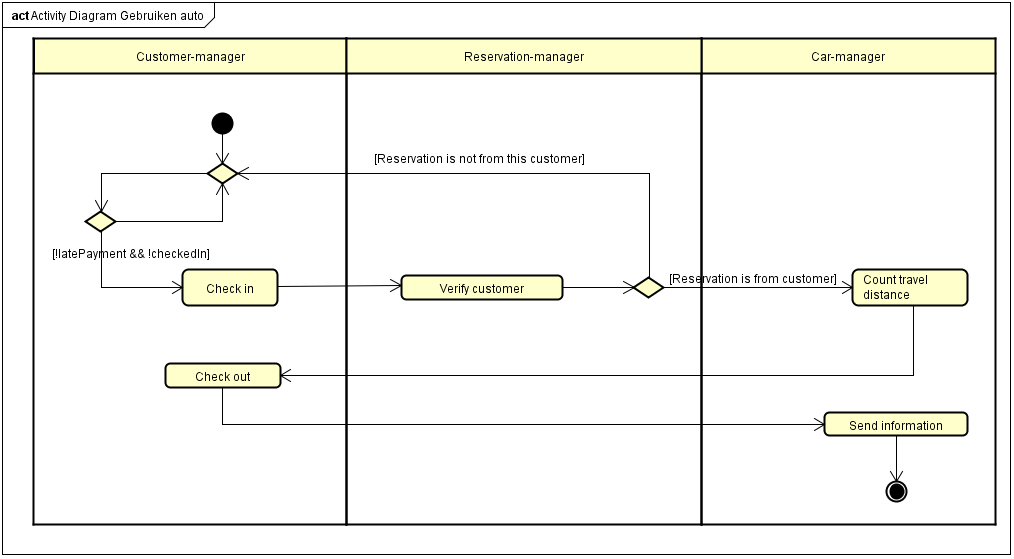
De reservering mag alleen gemaakt worden wanneer de auto beschikbaar is en de klant geen betalingsachterstand heeft.

#### Sequence diagram

Hier is het maken van de reservatie verder uitgewerkt. ReservationDAO beheert al instanties van Reservation en is daarom de Creator.



## 3.5. Gebruiken auto

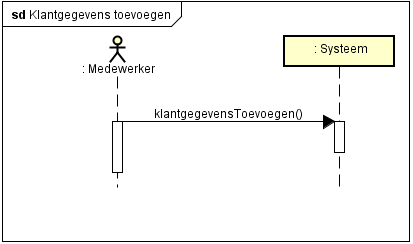


## 3.6. Klantgegevens beheren

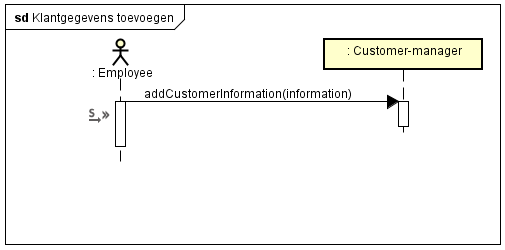
De CRUD Use case bestaat uit meerdere kleine Use cases, deze staan hieronder uitgewerkt.

### 3.6.1. Aanmaken klantgegevens

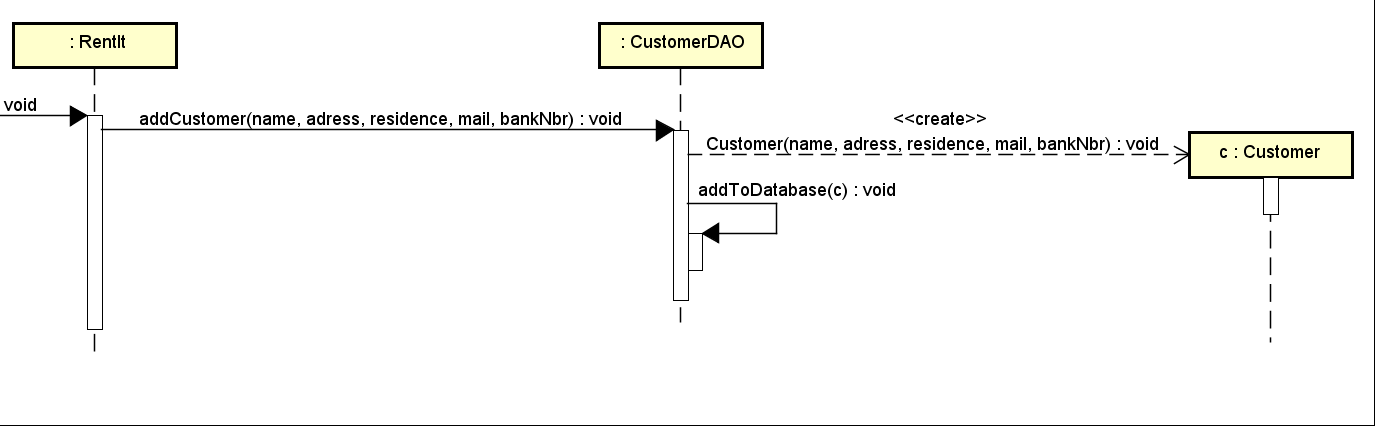
#### System sequence diagram



#### Component sequence diagram



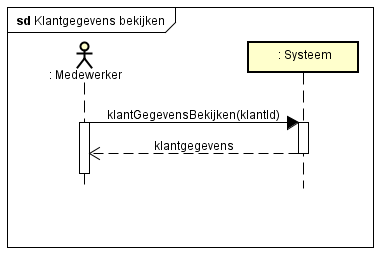
#### Sequence diagram

De CustomerDAO voegt de nieuwe Customer toe aan de database.

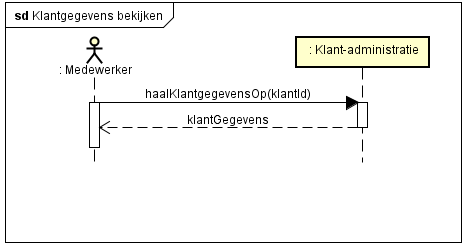
### 3.6.2. Bekijken klantgegevens

De klantgegevens moeten worden opgehaald.

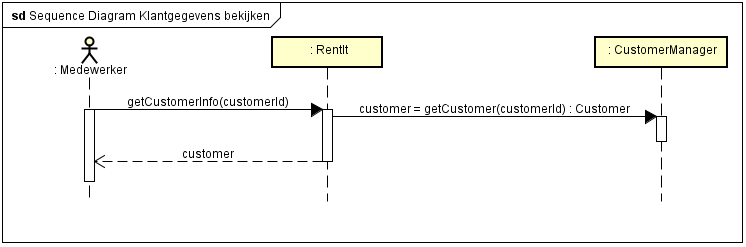
#### System sequence diagram



#### Component Sequence diagram



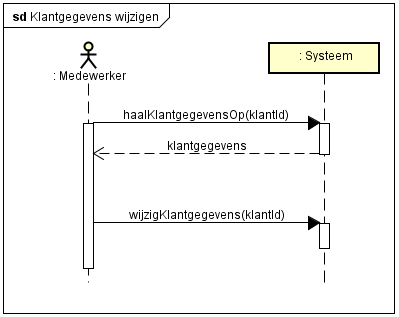
#### Sequence diagram



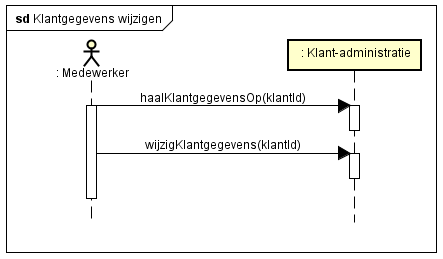
### 3.6.3. Klantgegevens wijzigen

Deze Use case heeft als preconditie dat er een klant is geselecteerd om te wijzigen.

#### System sequence diagram

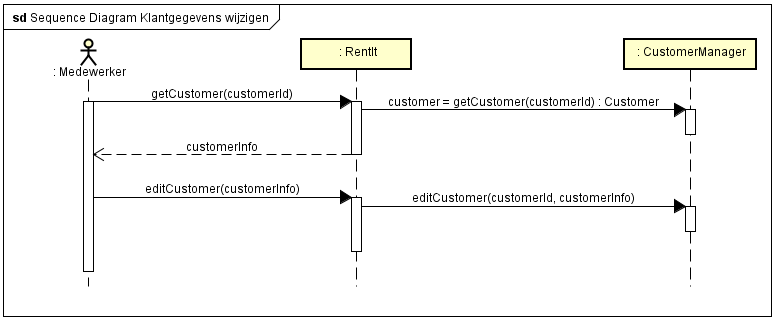


#### Component sequence diagram



#### Sequence diagram

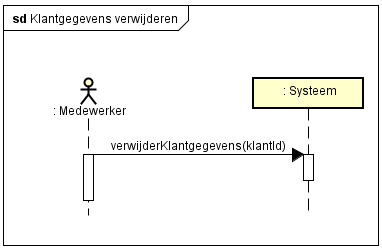
De instantie van customer wordt gewijzigd en terug



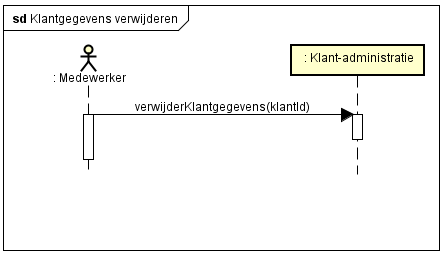
### 3.6.4. Klantgegevens verwijderen

Deze Use case heeft als preconditie dat er een klant is geselecteerd om te verwijderen.

#### System sequence diagram



#### Component sequence diagram



#### Sequence diagram

