REDCARS

In opdracht van Chris van Uffelen

Michel Peters
StudentNr: 586134
Jelle Bouwhuis
StudentNr: 585731

Versie 2.0

01 november 2017 - Arnhem

OSM: Modelling

HBO-ICT – Embedded system developer

HAN - Hogeschool van Arnhem en Nijmegen

VERSIEBEHEER

Datum	Wie	Aanpassingen	Versie
12/10/2017	Michel	Eerste opzet	0.09
12/10/2017	Jelle	Tweede opzet	0.1
01/11/2017	Michel/Jelle	Versie 1	1.0
02/11/2017	Michel/Jelle	Laatste versie	2.0

Inhoudsopgave

1 Inleiding

Autoverhuurbedrijf RentIt wil gaan starten met het aanbieden van deelauto's. Deze service gaan ze RedCars noemen. In plaats van auto's centraal op moeten halen bij RentIt zelf, wil RentIt de auto's gaan plaatsen waar de vraag is. Om de RedCars goed te laten functioneren, wil RentIt een applicatie laten ontwikkelen.

RentIt is met dit idee naar de ICA gestapt waar verschillende groepen studenten de analyse en het ontwerpproces van deze applicatie uitwerken met behulp van verschillende UML modelleertechnieken.

Dit document beschrijft de uitwerking van deze technieken, uitgevoerd door Jelle Bouwhuis en Michel Peters, voor de course OSM-M.

Het eerste hoofdstuk zal beginnen met een omschrijving van de casus en het bedrijft om een goed beeld te schetsen over de context. Hier opvolgend is de supplementary specification uitgewerkt waar een lijst van de requirements van de applicatie middels de FURPS standaard zijn gecategoriseerd. Het hoofdstuk wordt afgesloten met een Usecasediagram, waar de interactie tussen de gebruikers en het systeem in kaart wordt gebracht. Het volgende hoofdstuk is toegewijd aan beschrijvingen van de usecases waar voor elke usecase een brief-usecasebeschrijving te vinden is. Voor de belangrijke usecases worden deze verder uitgewerkt in fully-dressed usecasebeschrijvingen. Aan de hand van deze modellen is het domeinmodel gemodelleerd, waar de concepten en bijbehorende relaties die een rol spelen binnen het systeem zijn geïdentificeerd. De verschillende componenten en de communicatie hiertussen zijn middels het component diagram en component sequence diagrams in het daaropvolgende hoofdstuk gemodelleerd. Tot slot is in hoofdstuk 6 de stap gemaakt naar het software ontwerp waar aan de hand van het klassendiagram, sequence diagrams en gekozen design patterns het ontwerp van de applicatie zo goed mogelijk volgens de specificaties is gemodelleerd.

2 Casus omschrijving

In dit hoofdstuk is een korte opdrachtomschrijving gemaakt, waarbij de globale lijnen binnen het nieuwe platform RedCars duidelijk worden. Hierna worden volgens FURPS standaard de requirements en specificaties opgesteld. Tot slot worden deze requirements omgezet in een usecasediagram met de verschillende actoren binnen het platform en welke handelingen deze kunnen uitvoeren.

2.1 Opdracht omschrijving

Er wordt een nieuw platform ontwikkeld voor autoverhuurbedrijf RentIt, dat het nieuwe onderdeel RedCars zal aanbieden. Op dit platform kunnen personen zich aanmelden om gemakkelijk auto's te huren. Na aanmelding ontvangt de nieuwe klant een ledenpas waarmee hij kan inloggen op het platform.

Na ingelogd te zijn kan een klant verschillende deelauto's bekijken. Hierbij is de beschikbaarheid en locatie zichtbaar van een deelauto. Wanneer een deelauto beschikbaar is, kan de klant deze reserveren door begin- en eindtijd aan te geven.

Als een klant een reservering heeft gemaakt, en de tijd van reservering is aangebroken, kan de klant de reservering verzilveren. Dit kan doet hij door in te checken bij registratiepaal die beschikbaar is op de parkeerplaats van de deelauto, dit wordt gedaan met zijn ledenpas. Na dat de klant ingecheckt is, kan deze gebruik maken van de auto. De klant gebruikt zijn ledenpas om de auto te ont- en vergrendelen. Na gebruik beëindigt de klant zijn reservering door uit te checken bij de registratiepaal.

Medewerkers van RentIt kunnen de verschillende gegevens binnen het platform beheren. Hieronder vallen het aanmaken, wijzigen en verwijderen van klantgegevens en autogegevens.

2.2 Aannames

Bij het uitwerken van deze casus zijn er een aantal aannames gedaan om plotholes te dichten. Hieronder de aannames opgesomd:

- Gebruikers moeten aangemeld zijn op het systeem om gebruik te maken van het platform. Wanneer een gebruiker niet ingelogd is, heeft deze alleen de keuze om:
    - Te registeren
    - Aan te melden
- De manier waarop een ledenpas verstuurd wordt na registratie valt voor nu nog buiten de scope.
- De verbinding met de redcars module in een deelauto werkt te allen tijde en de GPSlocatie van de module is op de centimeter nauwkeurig
- Voor het berekenen van de prijs van een reservering wordt er geen rekening gehouden met daadwerkelijke dagen. Dit is van toepassing op een betaling per weekend, waarbij niet gecontroleerd zal worden of de reservering binnen het weekend valt.

2.2.1 Afspraken opdrachtgever

In overleg met de opdrachtgever is de keuze gemaakt om een registratiepaal toe te voegen. De registratiepaal is te vinden bij de parkeerlocaties van de deelauto's. In plaats van de reservering te verzilveren door de eerste keer in te checken bij de deelauto, verloopt het in- en uitchecken via de registratiepaal. Deze staat in contact met het systeem en is bedoeld om op een duidelijke manier informatie aan klanten te verschaffen. Daarnaast biedt het de mogelijkheid tot interactie, zoals het melden van schade aan een auto (die mogelijk door een voorgaande klant is gemaakt).

2.3 Specificaties

In dit hoofdstuk worden alle requirements binnen de opdracht gegroepeerd volgens het FURPS+ standaard. Hieronder een kleine beschrijving van het FURPS+ standaard:

- (F)unctional: Verzameling functionele eisen binnen het systeem.
- (U)sability: Eisen over hoe gemakkelijk er iets voor elkaar gekregen kan worden.
- (R)eliability: Eisen over hoe betrouwbaar het systeem moet zijn voor gebruik.
- (P)erformance: Eisen over hoe snel of goed het systeem reageren.
- (S)uportability: Eisen over uitbreidbaarheid van het systeem.
- (+)Plus: Andere eisen, zoals gegevens, uiterlijk en omgeving.

Alle functionele requirements zijn genummerd met de letter F. De andere requirements (Non-functional) zijn genummerd met de letters NF.

2.3.1 Functional

Nr	Requirement
F1	Een persoon registreert zich als klant bij redcars.
F2	Nieuwe klant ontvangt ledenpas met pasnummer na succesvolle registratie.
F3	Klant logt in op het systeem met gebruikersnaam en wachtwoord.
F4	Medewerker beheert klantgegevens (CRUD).
F5	Medewerker beheert autogegevens (CRUD).
F6	Klant kan niet inloggen op het systeem als hij als inactief gemarkeerd is.
F7	Klant ziet de locatie van een deelauto op de website.
F8	Klant ziet de beschikbaarheid van een deelauto op de website.
F9	Klant reserveert deelauto op de website aan de hand van begin- en eindtijd.
F10	Klant verzilvert zijn reservering door in te checken bij registratiepaal met zijn ledenpas.
F11	Klant meldt schade (gebreken) van de auto bij inchecken.
F12	Klant ontgrendelt de deelauto met zijn ledenpas als sleutel.
F13	Klant vergrendelt de deelauto met zijn ledenpas als sleutel.
F14	Klant beëindigt reservering door uit te checken bij een registratiepaal met zijn ledenpas.
F15	Factuur wordt automatisch aangemaakt na uitchecken deelauto of na verloop eindtijd reservering.
F16	Betalingsopdracht wordt automatisch ingeplant zodra een factuur is aangemaakt.
F17	Klant heeft de mogelijkheid om van abonnement-type te wisselen.
F18	De kosten voor een reservering wordt berekend op basis van abonnement-type, deelautotype en tariefsoort.
F19	Klant kan geen reservering doen wanneer er sprake is van een betalingsachterstand.
F20	Klant kan een reservering niet verzilveren wanneer de klant al ingecheckt is.

Tabel 1-Functional Requirements

2.3.2 Usability

Nr	Requirement
NF1	De locatie van een beschikbare deelauto is eenvoudig zichtbaar.
NF2	Klant kan alleen bij een registratiepaal inchecken binnen de periode van zijn reservering.
NF3	Klant kan alleen uitchecken wanneer de deelauto geparkeerd staat op de inchecklocatie.
NF4	Klant ontvangt een boete wanneer hij zijn deelauto parkeert op zijn startlocatie en niet is uitgecheckt na 10 minuten.
NF5	Bij het te laat uitchecken van een deelauto wordt het huurbedrag plus extra uren als boete verrekend.
NF6	De klant heeft een betalingsachterstand wanneer een automatische incasso is mislukt.

Tabel 2-Usability Requirements

2.3.3 Reliability

Er zijn geen eisen die over betrouwbaarheid gaan.

2.3.4 Performance

Nr	Requirement
NF7	De locatie- en beschikbaarheid gegevens van een deelauto die de klant ziet, zijn maximaal 10 seconden oud.

Tabel 3-Reliability Requirements

2.3.5 Suportability

Nr	Requirement
NF8	Types van deelauto's zijn eenvoudig uitbreidbaar.
NF9	Types van abonnementen zijn eenvoudig uitbreidbaar.

Tabel 4-Suportalbility Requirements

2.3.6 Plus

Nr	Requirement
NF10	De applicatie wordt ontwikkelt met de programmeertaal c++.
NF11	De eerste versie van de applicatie zal met een commandline interface uitgevoerd worden.
NF12	Klant heeft de volgende gegevens: NAW, Email en Bankrekeningnummer.
NF13	Deelauto heeft de volgende gegevens: Kenteken, Type en Standplaats.
NF14	Abonnement kent de volgende types: Gratis en Betaald.
NF15	Deelauto kent de volgende typen: Personenauto en Stationwagen met trekhaak.
NF16	Elk type deelauto kent per abonnement-type de volgende tarieven: Prijs per uur, Prijs per dag, Prijs per weekend, Prijs per week, Prijs per kilometer, Kilometers vrij.
NF17	Reservering kent de volgende tariefsoorten: Betaling per uur, per dag, per weekend, per week.

Tabel 5-Plus Requirements

2.4 Usecasemodel

Alle bovenstaande functionele requirements worden in dit hoofdstuk verwerkt in een usecasediagram. Dit wordt gedaan aan de hand van functionarissen.

2.4.1 Functionarissen

In het systeem zijn een aantal actoren bekend. Elke actor heeft een tot meerdere functionaris rollen. De functionarissen komen terug in het usecasediagram. Op deze manier wordt er duidelijk onderscheid gemaakt tussen de verschillende rollen die een actor kan uitvoeren.

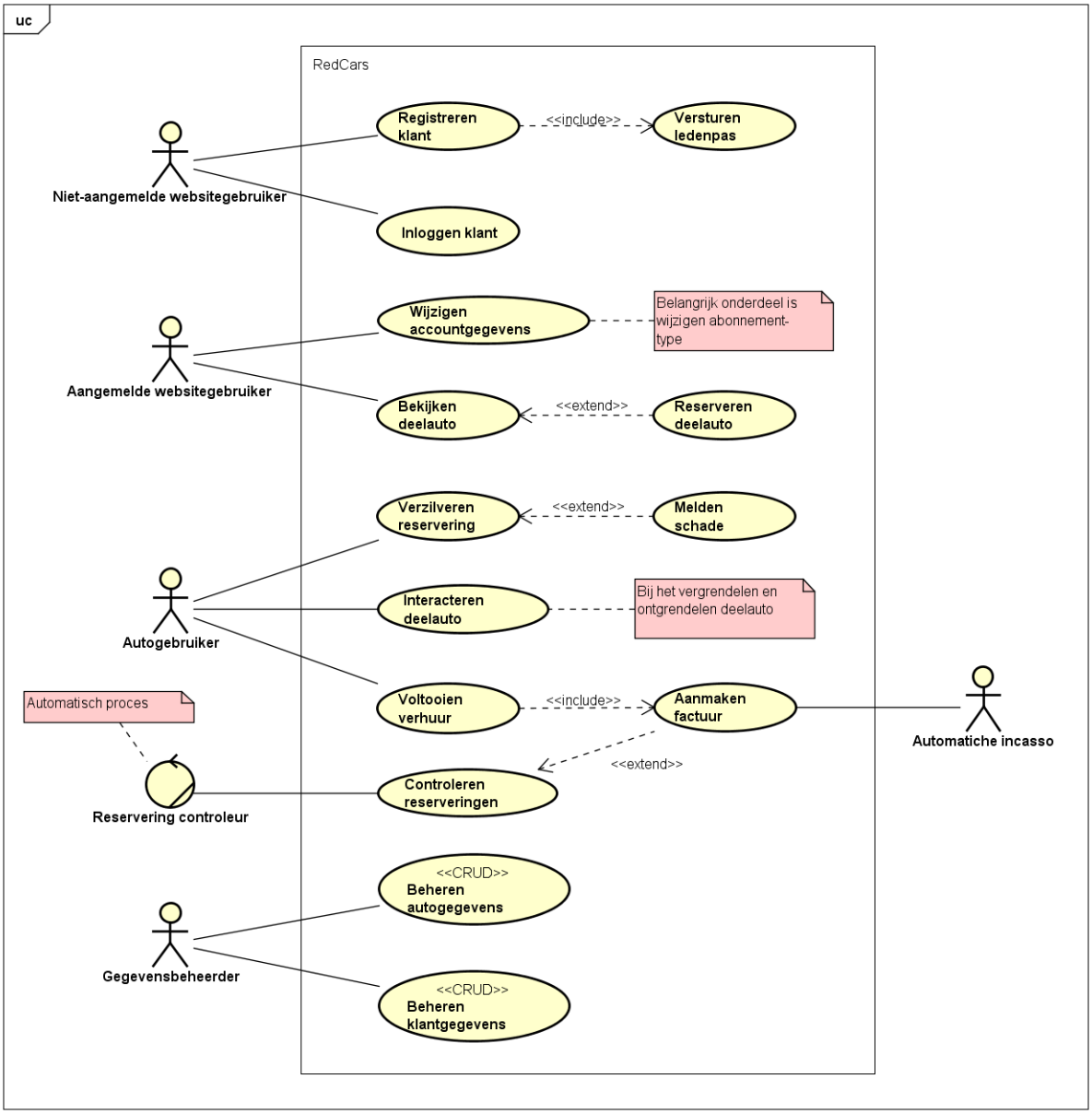
In het onderstaande tabel wordt weergegeven welke functionarisrollen worden vervuld per actor.

Functionaris Actor	Niet-aangemelde Websitegebruiker	Niet-aangemelde Websitegebruiker	Autogebruiker	Gegevens beheerder	Reservering controleur
Persoon	**X**				
Klant	**X**	**X**	**X**		
Medewerker				**X**	
Tijd					**X**

Tabel 6-Functionarissen

2.4.2 Usecasediagram

In het usecasediagram worden de functionarissen gekoppeld aan usecases. Abonnementen worden in deze iteratie niet geincasseerd.



Figuur 1-Usecasediagram

3 Usecase uitwerking

In dit hoofdstuk wordt het usecasediagram tekstueel uitgewerkt met behulp van brief usecases en fully-dressed usecases. Bij de brief usecases wordt een korte beschrijving gegeven tussen interactie van actor en systeem. Bij de fully-dressed usecases worden een aantal brief usecases verder uitgewerkt, waarbij pre- en post condities, stakeholders en alternatieve flows uitgebreid beschreven worden. Tot slot worden de fully-dressed usecasebeschrijvingen uitgewerkt in system sequence diagrammen. Hier wordt de interactie tussen actor en systeem visueel afgebeeld.

3.1 Brief usecases

Voor elke usecase wordt een brief usecasebeschrijving opgesteld. Elke brief usecasebeschrijving begint met een trigger die het proces in gang zet. Hierna volgt een interactie tussen actor en systeem, zonder verdere alternatieve flows.

3.1.1.1 Registreren klant

Persoon wil graag een van de deelauto's van RedCars huren en is nog geen klant. Klant vult registratieformulier in met vereiste gegevens (NAW, email, bank) en geeft hierbij toestemming tot automatisch incasso. Het systeem controleert of de gegevens correct zijn ingevuld. System slaat nieuwe klantgegevens op. Systeem voert het proces verzenden ledenpas uit (usecase <<verzenden ledenpas>>). Systeem laat succesmelding zien. Klant kan nu geduldig wachten op zijn ledenpas. Klant heeft zich geregistreerd op het systeem.

3.1.1.2 Versturen ledenpas

Klant wil beschikken over ledenpas. Systeem genereert ledenpas met uniek pasnummer. Systeem print pas uit. Systeem print envelop met adresgegevens klant uit. Systeem verstuurt via een nog te definiëren manier de ledenpas naar klant. Ledenpas is verstuurd.

3.1.1.3 Aanmelden klant

Klant wil zich graag aanmelden zodat hij gebruik kan maken van het platform. Klant vult het aanmeldformulier in met zijn email als gebruikersnaam en pasnummer als wachtwoord. Systeem controleert gegevens. Systeem stuurt klant door naar het platform. Klant heeft zich aangemeld op het systeem.

3.1.1.4 Bekijken deelauto

Klant wil kijken welke deelauto's op een locatie beschikbaar zijn. Klant selecteert plaats. Klant vraagt lijst van deelauto's op. Systeem toont deelauto's in omgeving van geselecteerde plaats. Klant selecteert deelauto. Systeem toont locatie en beschikbaarheid informatie van deelauto. Klant heeft de mogelijkheid om deelauto te reserveren (usecase <<reserveren deelauto>>).

3.1.1.5 Reserveren deelauto

Klant heeft bij het bekijken van een deelauto besloten om deze te reserveren. Klant vult reserveringsformulier in met gewenste periode en tariefsoort. Systeem controleert juistheid gegevens en beschikbaarheid. Systeem maakt nieuwe reservering aan en slaat deze op. Systeem stuurt reserveringsbevestiging via e-mail. Systeem laat succesmelding zien. Klant heeft deelauto voor de ingegeven periode gereserveerd.

3.1.1.6 Wijzigen accountgegevens

Klant wil zijn abonnement-type of persoonsgegevens aanpassen. Klant vraagt accountgegevens op. Systeem toont persoonsgegevens (NAW, email, bankrekening) en abonnement-type van klant. Klant wijzigt persoonsgegevens. Klant slaat wijzigingen op. Systeem controleert gegevens. Systeem slaat gegevens op. Systeem toont succesmelding. Klant wijzigt abonnement-type. Klant slaat nieuwe

abonnement-type keuze op. Systeem controleert gegevens. Systeem slaat abonnement-type keuze op. Systeem toont succesmelding. Klant heeft zijn accountgegevens aangepast.

3.1.1.7 Verzilveren reservering

Klant wil gaan rijden in zijn gereserveerde deelauto. Klant bevindt zich bij de parkeerlocatie van de gereserveerde deelauto. Klant houdt zijn ledenpas in de buurt van de scanner bij een van de beschikbare reserveringspalen. Systeem leest pasnummer. Systeem controleert of er een geldige reservering bekend is voor de klant. Systeem upgrade reservering tot verhuur. Systeem werkt incheckmoment verhuur bij. Systeem toont informatie over de parkpeerplek en kenteken van de deelauto. Systeem biedt mogelijkheid tot invoeren schadeformulier (usecase <<melden schade>>). De klant is ingecheckt en kan gebruik maken van de deelauto.

3.1.1.8 Melden schade

Klant heeft bij het inchecken aangegeven om schade aan de auto te melden. Systeem toont schadeformulier. Klant vult omschrijving schade in. Systeem slaat schademelding op. Klant heeft schade gemeld.

3.1.1.9 Interacteren deelauto

Klant wil een deelauto ontgrendelen of vergrendelen. Klant houdt zijn ledenpas in de buurt van de redcarmodule. Systeem leest pasnummer. Systeem controleert of de klant een verhuur heeft over betreffende deelauto. Systeem ontgrendelt of vergrendelt de deelauto. Klant heeft de deelauto ontgrendelt/vergrendelt.

3.1.1.10 Voltooien verhuur

Klant wilt deelauto uitchecken. Klant parkeert de deelauto op de originele parkeerlocatie. Klant houdt zijn ledenpas in de buurt van de scanner bij een van de beschikbare reserveringspalen. Systeem leest pasnummer. Systeem controleert verhuur. Het systeem controleert of de deelauto zich op de juiste parkeerlocatie bevindt. Systeem werkt gegevens bij. Systeem maakt een factuur aan d.m.v. usecase <<aanmaken factuur>>. Systeem toont verhuurgegevens en de totaalprijs. Verhuur is voltooid.

3.1.1.11 Aanmaken factuur

Klant moet een factuur ontvangen over een product. Systeem berekend kosten product. Systeem slaat factuurgegevens op (datum, bedrag, klant, product). Systeem dient afschrijving in bij automatische incasso. Factuur is aangemaakt en ingediend.

3.1.1.12 Controleren reserveringen

Reservering controleur wil weten of er reserveringen verlopen zijn. Reservering controleur vraagt alle reserveringen op waarbij de eindtijd verlopen is en er niet is ingecheckt. Systeem geeft een lijst van reserveringen die aan deze conditie voldoen. Reserveringcontroleur maakt voor elke reservering een factuur aan (usecase <<aanmaken factuur>>).

3.1.1.13 Beheren autogegevens

Medewerker wil de gegevens van een deelauto aanpassen. Medewerker vraagt de lijst autogegevens op. Systeem toont een lijst van deelauto's. Medewerker selecteert een deelauto om te wijzigen. Systeem toont gegevens van geselecteerde deelauto. Medewerker wijzigt gegevens. Medewerker slaat de nieuwe gegevens op. Systeem controleert de ingevulde gegevens. Systeem slaat de nieuwe gegevens op. Medewerker voegt een nieuwe deelauto toe. Systeem toont formulier voor nieuwe deelauto. Medewerker vult het formulier in. Medewerker slaat de gegevens op. Systeem controleert de gegevens. Systeem slaat de gegevens op. Medewerker selecteert een deelauto om te wijzigen.

Systeem toont gegevens van geselecteerde deelauto. Medewerker geeft aan de deelauto te verwijderen. Systeem verwijdert deelauto. Autogegevens zijn beheerd.

3.1.1.14 Beheren klantgegevens

Medewerker wil de gegevens van een klant beheren:

Create Klant
Medewerker wil een nieuwe kant toevoegen. Systeem toont formulier voor toevoegen nieuwe klant. Medewerker past gegevens klant in. Medewerker verzend de gegevens. Systeem controleert de gegevens. Systeem slaat gegevens op. Systeem toont succesmelding van toevoeging.

Update Klant
Medewerker wil klantgegevens wijzigen. Medewerker vraagt lijst van bestaande klanten op. Systeem toont lijst van alle klanten. Medewerker selecteert een klant om te wijzigen. Systeem toont formulier met gegevens. Medewerker past gegevens aan. Medewerker verzend de nieuw ingevulde gegevens. Systeem controleert de gegevens. Systeem slaat gegevens op. Systeem toont succesmelding van wijziging.

Delete Klant
Medewerker wil een klant verwijderen. Medewerker vraagt lijst van bestaande klanten op. Systeem toont lijst van alle klanten. Medewerker selecteert een klant om te verwijderen. Systeem vraagt om confirmatie. Klant geeft aan klant te willen verwijderen. Systeem verwijdert klantgegevens. Systeem toont succesmelding van verwijdering.

3.2 Usecase detaillering

Een aantal belangrijke usecases zijn volledig uitgewerkt met behulp van fully-dressed usecasebeschrijving, system sequence diagrams en waar nodig activity diagrams. Voor iteratie 1 zijn dit *reserveren deelauto, verzilveren reservering* en *voltooi verhuur*. In iteratie 2 zijn hier de fully-dressed beschrijvingen van *beheren klantgegevens* bij gekomen, die opgedeelt zijn in *create klant, update klant* en *delete klant*.

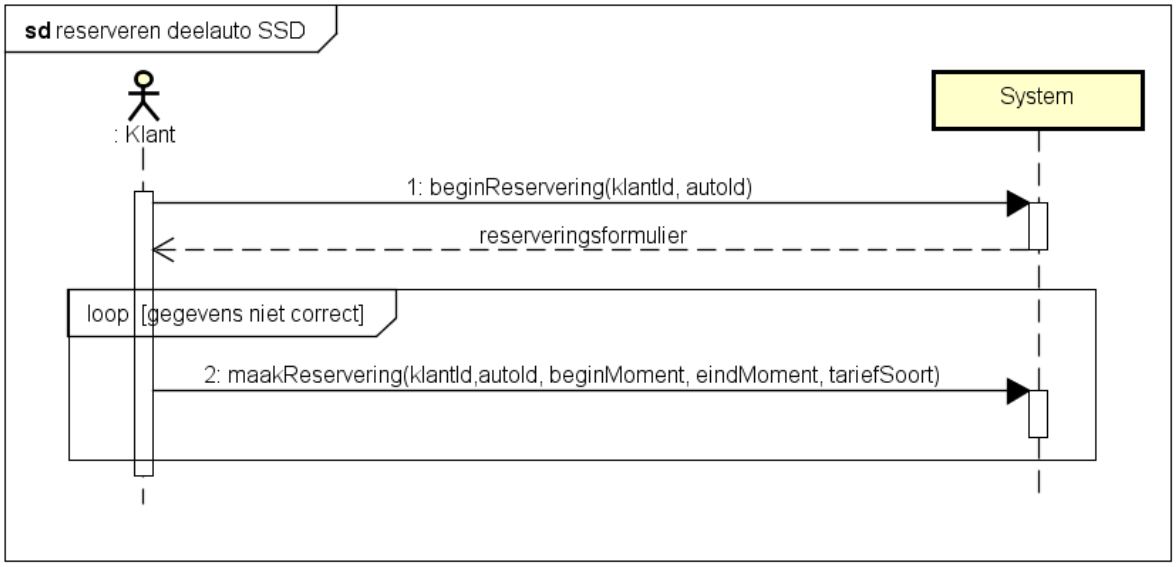
3.2.1 Fully-dressed usecasebeschrijving: Reserveren deelauto

Reserveren deelauto	
Beschrijving	Klant wilt deelauto voor een bepaalde periode reserveren
Primary actor	Klant
Secondary actors	
Stakeholders & Interests	• RentIt: Klant maakt gebruikt van platform en levert (waarschijnlijk) geld op in de toekomst
Pre-condities	• Klant is aangemeld op het systeem • Klant heeft een deelauto geselecteerd
Post-condities	• De gegevens van de reservering zijn opgeslagen • Er is een e-mail bevestiging verstuurd

Hoofd succes scenario	
ACTOR	SYSTEEM
1. Klant heeft een deelauto uit de bestaande selectie gekozen en geeft aan een reservering te willen plaatsen	2. Systeem laat formulier met gegevens reservering zien
3. Klant voert gegevens (Begin moment, eindmoment, tariefsoort)	4. Systeem controleert ingevoerde gegevens
	5. [Gegevens correct] Systeem slaat reserveringsgegevens op
	6. Systeem verstuurd bevestigingsemail
	7. Systeem toont succesmelding
8. Klant heeft reservering geplaatst	
Alternative flows	
3a. [Klant stopt met reserveren]	
Usecase eindigt	
5a. [Gegevens incorrect: deelauto niet beschikbaar]	
	1. Systeem toont een foutmelding dat de deelauto niet beschikbaar is op de ingevoerde periode
Usecase gaat verder bij stap 3.	
5b. [Gegevens incorrect: klant heeft al een reservering binnen dezelfde periode]	
	1. Systeem toont foutmelding dat de klant al een reservering heeft binnen de ingevoerde periode
Usecase gaat verder bij stap 3.	

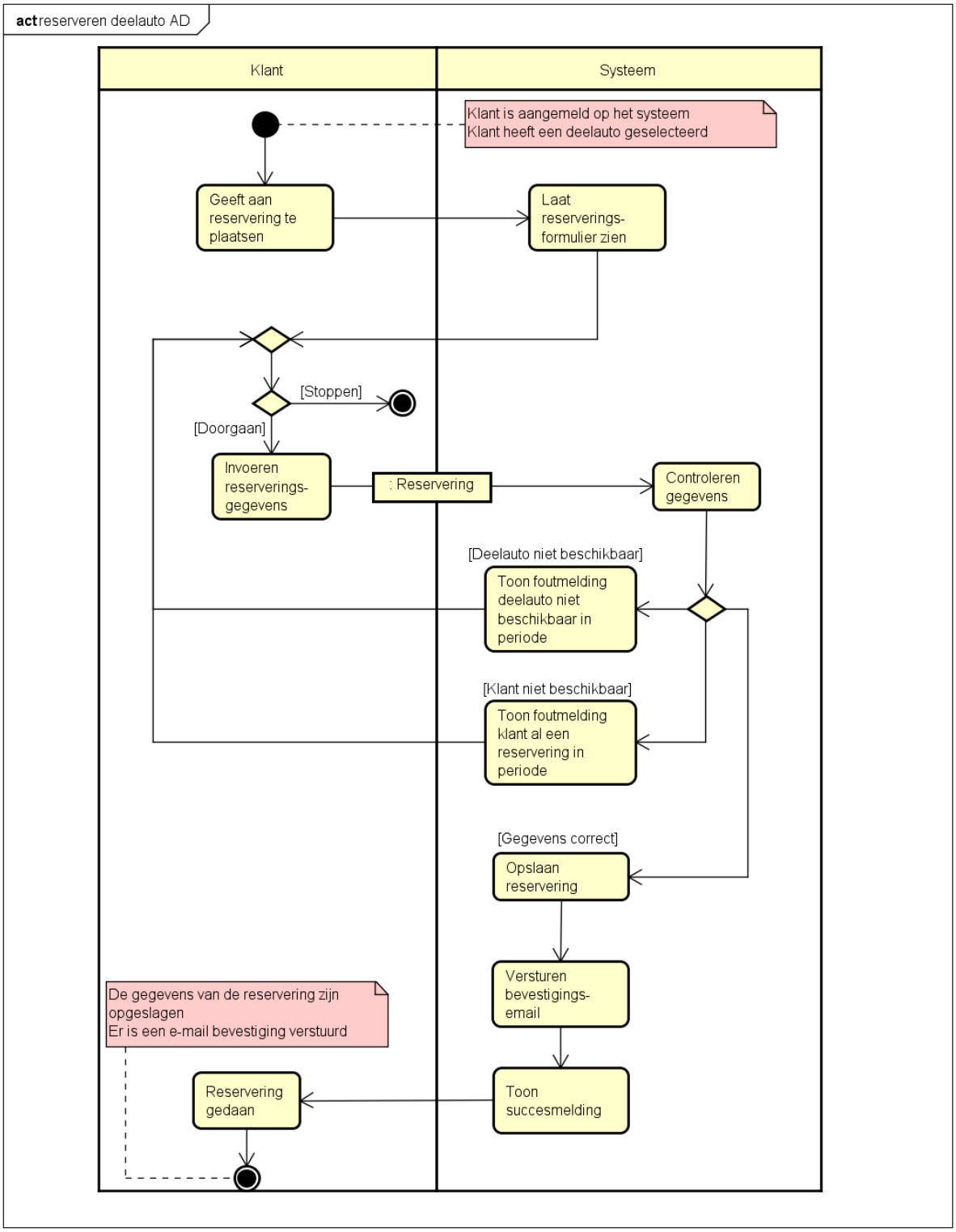
Tabel 7-Fully-dressed usecasebeschrijving reserveren deelauto

3.2.1.1 System sequence Diagram: Reserveren deelauto



Figuur 2-SSD reserveren deelauto

3.2.1.2 Activity diagram: Reserveren deelauto



Figuur 3-AD reserveren deelauto

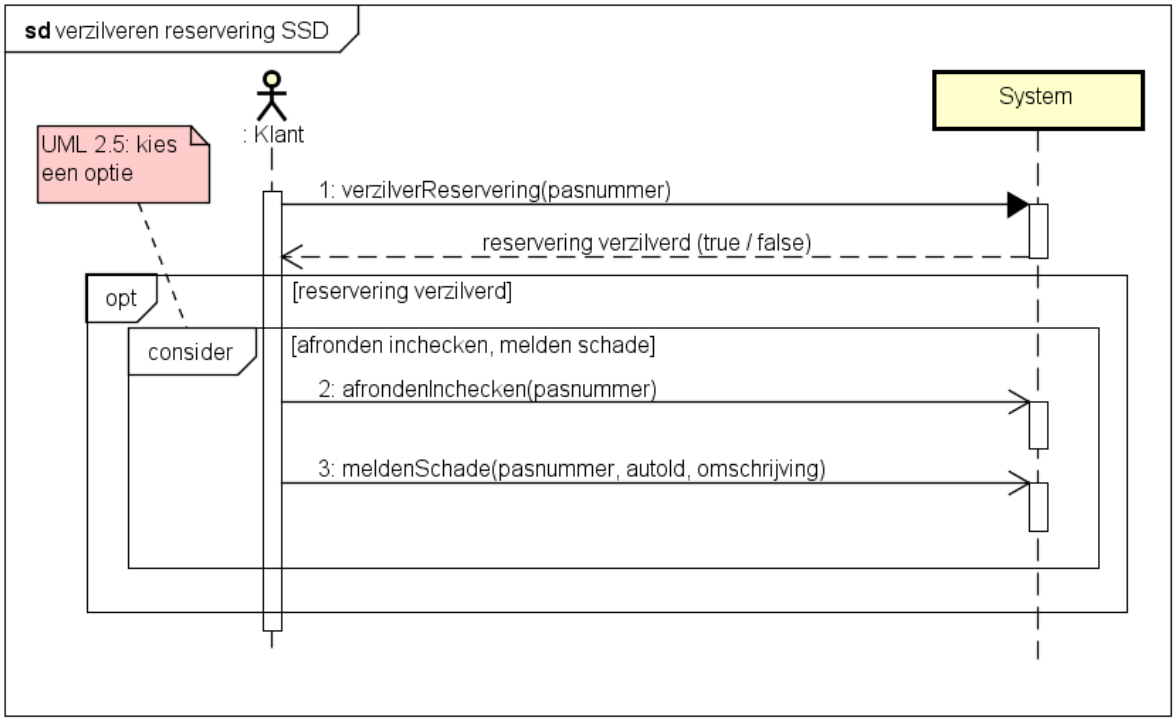
3.2.2 Fully-dressed usecasebeschrijving: Verzilveren reservering

Verzilveren reservering	
Beschrijving	Klant wil zijn reservering verzilveren zodat hij gebruik kan maken van de gereserveerde deelauto.
Primary actor	Klant
Secondary actors	-
Stakeholders & Interests	-
Pre-condities	• Klant heeft aangegeven te willen inchecken • Klant is in bezit van een ledenpas
Post-condities	• Er is een nieuw verhuur aangemaakt op basis van reservering • Het incheckmoment van het verhuur is huidige datum / tijd • Het aantal kilometers van de redcarmodule staat op 0 • Het pasnummer van de redcarmodule is het pasnummer van de klant

Hoofd succes scenario	
ACTOR	SYSTEEM
1. Klant houdt zijn ledenpas in de buurt van de scanner bij een van de beschikbare reserveringspalen	2. Systeem leest pasnummer
	3. Systeem controleert of er een geldige reservering bekend is voor de klant
	4. [Reservering gevonden] Systeem maakt een nieuw verhuur aan op basis van reservering.
	5. Systeem zet incheckmoment op huidige datum / tijd
	6. Systeem zet pasnummer van de redcarsmodule als het pasnummer van de klant
	7. Systeem zet aantal kilometers van de redcarmodule op 0
	8. Systeem toont informatie over auto locatie en kenteken
9. Klant maakt keuze tussen afronden inchecken of schade melden	10. [keuze: afronden inchecken] Systeem toont begin scherm
11. Klant kan nu gebruik maken van de deelauto	
Alternative flows	
4a. [Geen reservering gevonden]	
	1. Systeem toont melding dat klant geen geldige reservering heeft
Usecase eindigt	
10a. [Keuze: schade melden]	
Klant voert de usecase <<melden schade>> uit	
Usecase gaat verder bij stap 9	

Tabel 8-Fully-dressed usecasebeschrijving verzilveren reservering

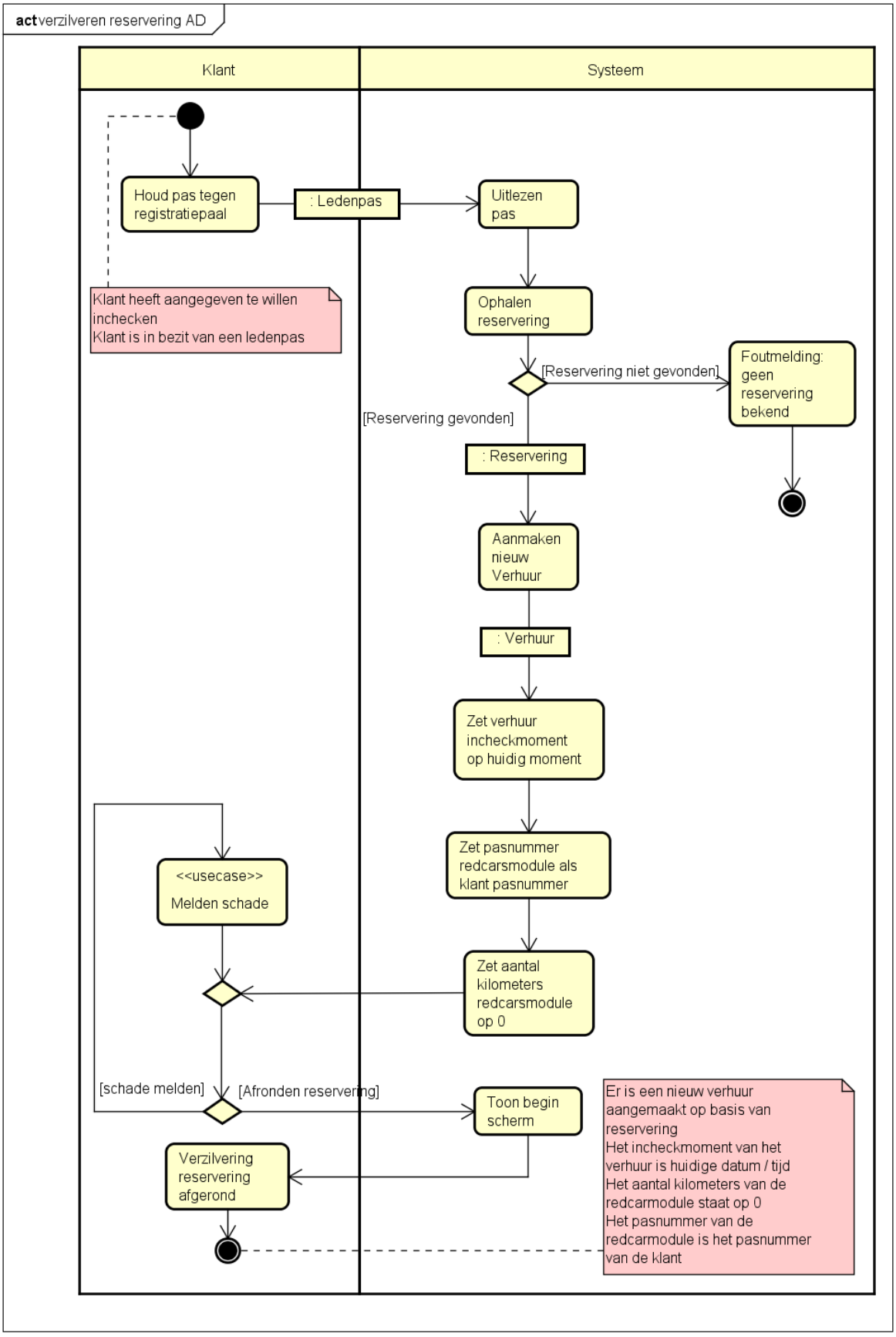
3.2.2.1 System sequence diagram: Verzilveren reservering



Figuur 4-SSD verzilveren reservering[1]

[1] 1. http://www.uml-diagrams.org/sequence-diagrams-combined-fragment.html#operator-consider

3.2.2.2 Activity diagram: Verzilveren reservering



Figuur 5-AD verzilveren reservering

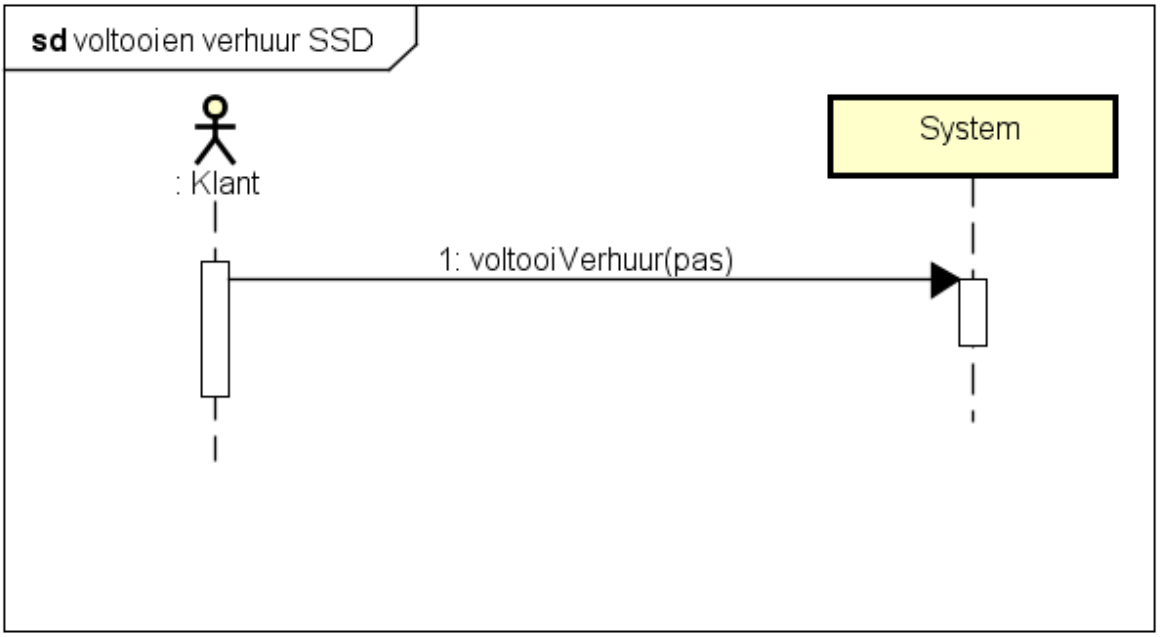
3.2.3 Fully-dressed usecase beschrijving: Voltooien verhuur

Beëindigen reservering	
Beschrijving	Klant wilt zijn verhuur afronden
Primary actor	Klant
Secondary actors	
Stakeholders & Interests	• RentIt: Deelauto is teruggebracht en ziet betaling klant tegemoet
Pre-condities	• Klant heeft aangegeven te willen uitchecken • Klant is in bezit van een ledenpas
Post-condities	• Verhuurgegevens (uitcheck moment, aantal kilometers) zijn bijgewerkt • Het pasnummer van de redcarmodule staat op 0 • Usecase <<factuur aanmaken>> is uitgevoerd

Hoofd succes scenario	
ACTOR	SYSTEEM
1. Klant houdt zijn ledenpas in de buurt van de scanner bij een van de beschikbare reserveringspalen	2. Systeem leest pasnummer
	3. Systeem controleert of er een geldige verhuur bekend is voor de klant
	4. [Vehuur gevonden] Systeem controleert of de deelauto zich op de juiste locatie bevindt
	5. [Deelauto locatie ok] Systeem werkt verhuurgegevens (uitcheck moment en aantal kilometers) bij
	6. Systeem zet het pasnummer van de redcarmodule op 0
	7. Systeem voert usecase <<aanmaken factuur>>
	8. Systeem toont verhuurgegevens en totaalprijs
9. Klant heeft zijn verhuur succesvol afgerond	
Alternative flows	
4a. [Verhuur niet gevonden]	
	1. Systeem toont een foutmelding dat er geen verhuur bekend is
Usecase eindigt	
5a. [Deelauto locatie niet ok]	
	1. Systeem toont een foutmelding dat de deelauto niet (op de juiste locatie) kon worden gelokaliseerd
Usecase eindigt	

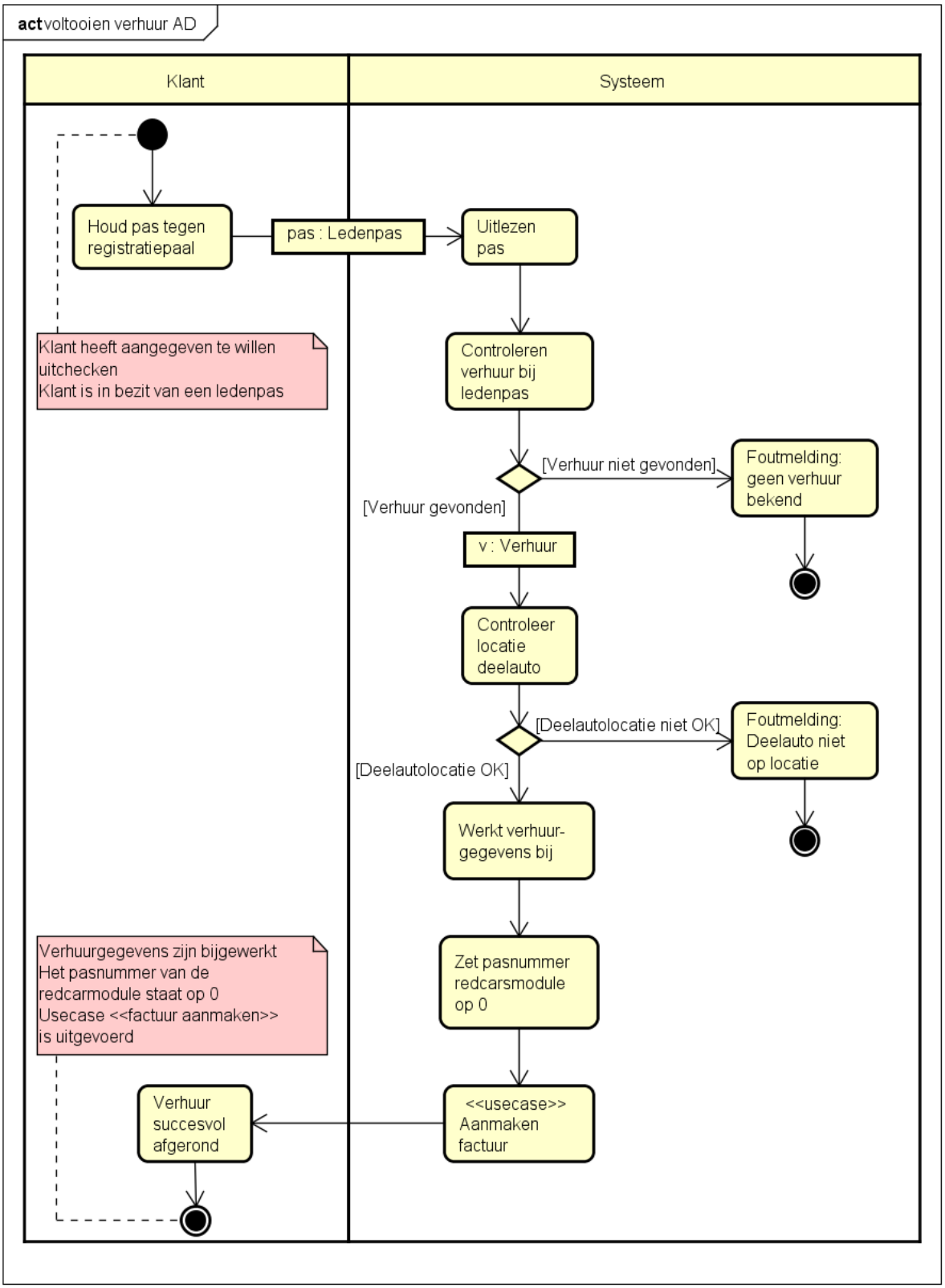
Tabel 9-Fully-dressed usecasebeschrijving voltooien verhuur

3.2.3.1 System sequence diagram: Voltooien verhuur



Figuur 6-SSD voltooien verhuur

3.2.3.2 *Activity diagram: Voltooien verhuur*



Figuur 7-AD voltooien verhuur

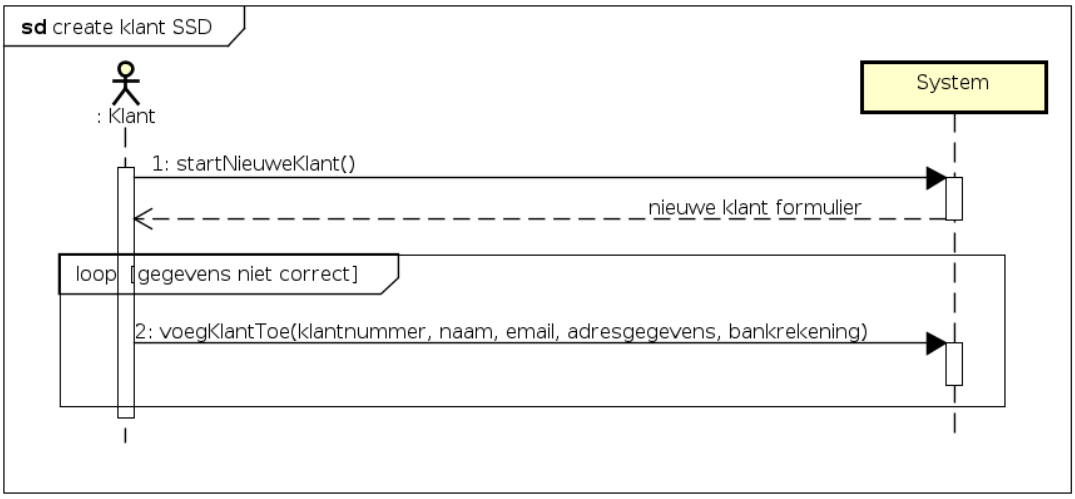
3.2.4 Fully-dressed usecase beschrijving: Create Klant

Create Klant	
Beschrijving	Medewerker wil een klant toevoegen
Primary actor	Medewerker
Secondary actors	
Stakeholders & Interests	• Klant: In het geval van problemen met het aanmaken van een Klant door de klant zelf, kan een medewerker hierbij helpen
Pre-condities	• Medewerker wil een nieuwe klant in het systeem invoeren
Post-condities	• Klantgegevens (NAW, bankgegevens, email, klantnummer) van de nieuwe klant zijn in het systeem opgeslagen. • **Notitie**: omdat dit een handmatige bewerking (het aanmaken van een klant) betreft, wordt er in dit geval niet automatisch een ledenpas verstuurd (zoals bij usecase <<Registreren klant>> gebeurt).

Hoofd succes scenario	
ACTOR	SYSTEEM
1. Medewerker bevindt zich bij het klantbeheer systeem en geeft aan een nieuwe klant aan te willen maken	2. Systeem toont een formulier met velden voor de gegevens van het toevoegen van een klant.
3. Medewerker vult de gevraagde gegevens in	
4. Medewerker verstuurt ingevulde gegevens	5. Systeem controleert de gegevens op geldigheid
	6. [Gegevens ok] Systeem slaat gegevens op
	7. Systeem toont een melding dat het toevoegen met succes is uitgevoerd
Alternative flows	
6a. [Gegevens niet ok]	
	1. Systeem toont een foutmelding dat er onjuiste gegevens zijn ingevoerd
Usecase eindigt	

Tabel 10-Fully-dressed usecasebeschrijving create klant

3.2.4.1 System sequence diagram: Create Klant



Figuur 8-SSD create klant

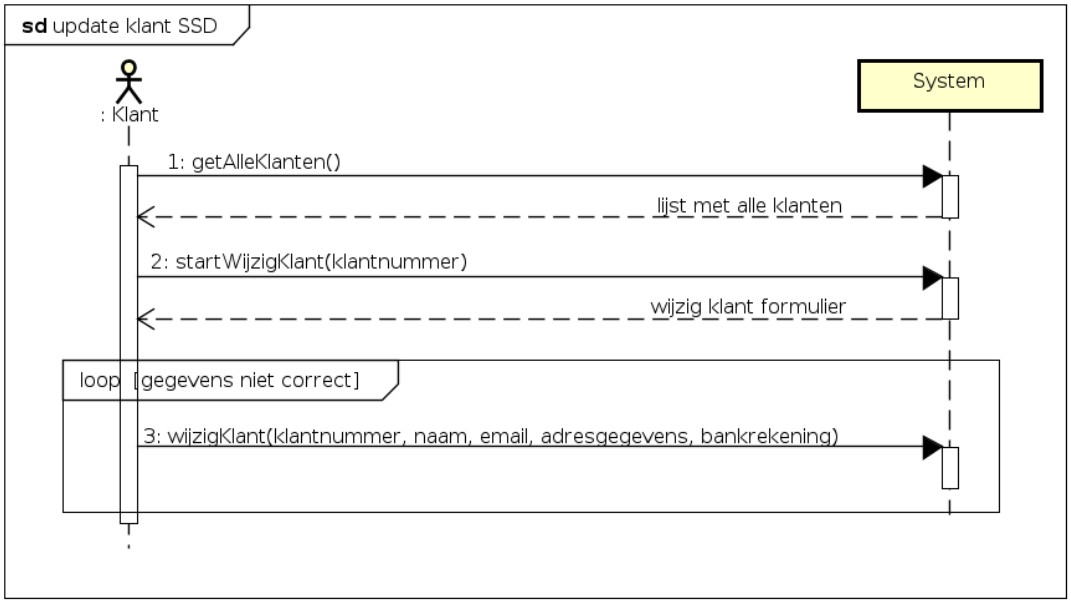
3.2.5 Fully-dressed usecase beschrijving: Update Klant

Update Klant	
Beschrijving	Medewerker wil een klant toevoegen
Primary actor	Medewerker wil de gegevens van een bestaande klant wijzigen
Secondary actors	
Stakeholders & Interests	• Klant: wordt geholpen en is er gebaat bij dat zijn gegevens juist zijn
Pre-condities	• Er is een bestaande klant voor wie de gegevens gewijzigd worden
Post-condities	• Een of meer gegevens (NAW, bankgegevens, email, klantnummer) van de klant zijn aangepast en opgeslagen.

Hoofd succes scenario	
ACTOR	SYSTEEM
1. Medewerker bevindt zich op het klantbeheersysteem en geeft aan dat hij de gegevens van een bestaande klant wilt opvragen	2. Systeem geeft een lijst met bestaande klanten terug
3. Medewerker selecteert de gewenste klant	4. Systeem toont de een formulier met de huidige gegevens van de geselecteerde klant
5. Medewerker past de gewenste gegevens aan een stuurt deze op	6. Systeem controleert de gegevens op geldigheid
	7. [Gegevens ok] Systeem slaat gegevens op
	8. Systeem toont een melding dat het toevoegen met succes is uitgevoerd
Alternative flows	
6a. [Gegevens niet ok]	
	2. Systeem toont een foutmelding dat er onjuiste gegevens zijn ingevoerd
Usecase eindigt	

Tabel 11-Fully-dressed usecasebeschrijving update klant

3.2.5.1 System sequence diagram: Update Klant



Figuur 9-SSD update klant

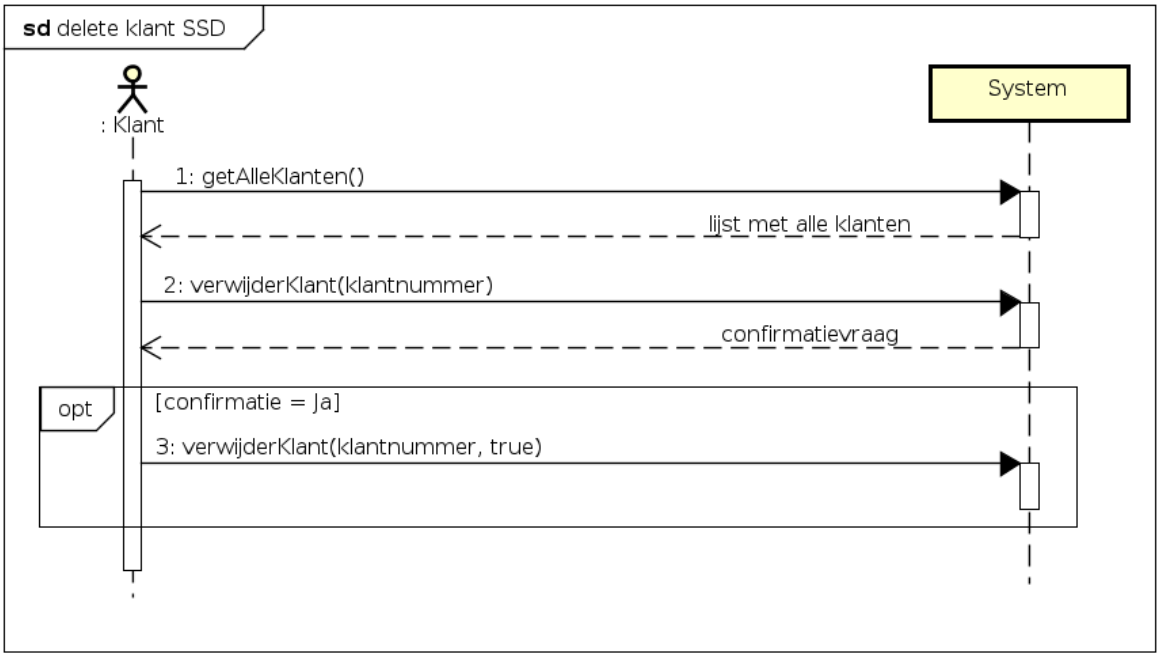
3.2.6 Fully-dressed usecase beschrijving: Delete Klant

Delete Klant	
Beschrijving	Medewerker wil een klant verwijderen
Primary actor	Medewerker: wilt een bestaande klant verwijderen
Secondary actors	Klant: zijn gegevens zijn verwijderd uit het systeem
Stakeholders & Interests	• Klant: klant wordt uit het system verwijderd en verliest hiermee de faciliteiten die gelden voor klanten (op het systeem)
Pre-condities	• Er is een bestaande klant
Post-condities	• Klant is verwijderd van het systeem en deze wijziging is opgeslagen

Hoofd succes scenario	
ACTOR	SYSTEEM
1. Medewerker bevindt zich op het klantbeheersysteem en vraagt een lijst met bestaande klanten op	2. Systeem geeft een lijst met bestaande klanten terug
3. Medewerker selecteert de gewenste klant en geeft aan deze te willen verwijderen	4. Systeem vraagt klant om confirmatie)
5. Medewerker conformeert en geeft aan de klant te willen verwijderen	6. Systeem verwijder de klantgegevens
	7. Systeem toont een melding dat het verwijderen is gelukt
Alternative flows	
n.v.t.	

Tabel 12-Fully-dressed usecasebeschrijving delete klant

3.2.6.1 System sequence diagram: Delete Klant

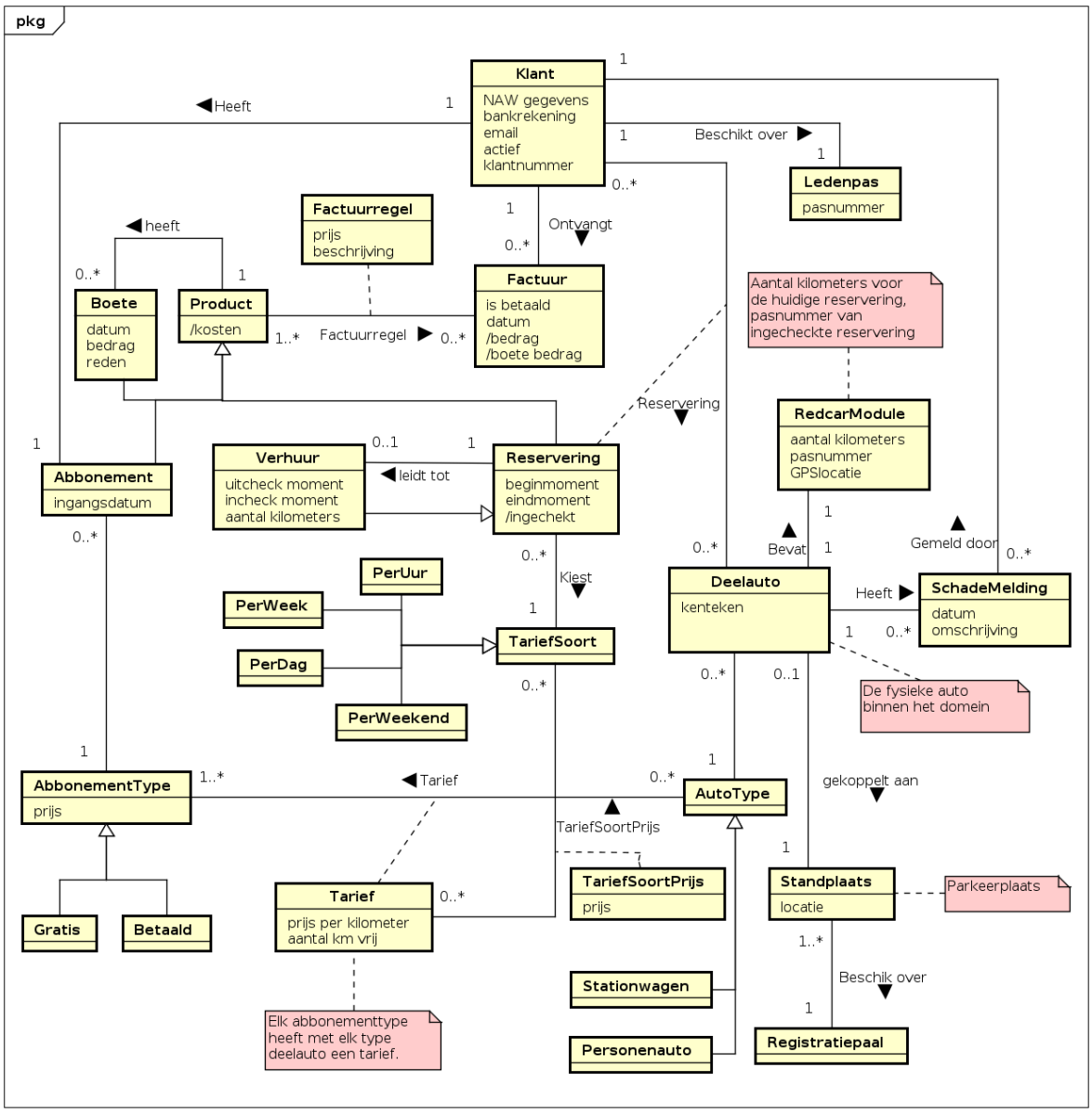


Figuur 10-SSD delete klant

4 Domein

In dit hoofdstuk is het domein uitgewerkt. Alle concepten binnen de casus, gebruikt in de usecases, worden in het domein model uitgewerkt. Met associaties worden de relaties tussen verschillende concepten aangegeven.

4.1 Domein model



Figuur 11-Domeinmodel

4.2 Uitleg model

In dit hoofdstuk is de samenhang van de concepten beschreven.

4.2.1 Klanten

Klanten binnen systeem, hebben een abonnement en kunnen een deelauto reserveren. Voor deze producten ontvangt een klant een factuur. Elke klant heeft ook een ledenpas met pasnummer.

4.2.2 Producten

Alle factureerbare producten, zoals abonnement, reservering en verhuur. Een reservering gaat altijd over een deelauto. Nadat een reservering verzilverd is leidt deze tot een verhuur, die extra gegevens bevat. Bij elk product kunnen boetes opgenomen worden, zoals extra kosten voor te laat betalen, reservering niet verzilverd of auto te laat teruggebracht. Aan elk product kunnen facturen gekoppeld worden. Dit gaat aan de hand van factuurregels.

4.2.3 Deelauto's

De deelauto's zijn de auto's die verhuurt kunnen worden. Elke deelauto heeft een RedCarModule, die de kilometers en locatie bij houdt. Deze slaat ook het pasnummer op van de klant die de auto mag gebruiken. Deelauto's hebben een autotype, zoals stationwagen en personenauto, en staan gekoppeld aan een parkeerplaats. Een deelauto kan ook schademeldingen hebben, deze worden gemeld door een klant.

4.2.4 Tarieven

Het tarief voor een auto wordt bepaald door het autotype en abonnement type. Elk tarief heeft standaard kosten per kilometer en aantal kilometers vrij, met daarbij de mogelijkheid om dit uit te breiden met verschillende tariefsoorten. Een tariefsoort is de manier waarop betaald wordt, zoals per uur of per dag. Deze keuze wordt gemaakt per reservering. Elk tarief heeft zijn eigen prijs per tariefsoort.

5 Componenten

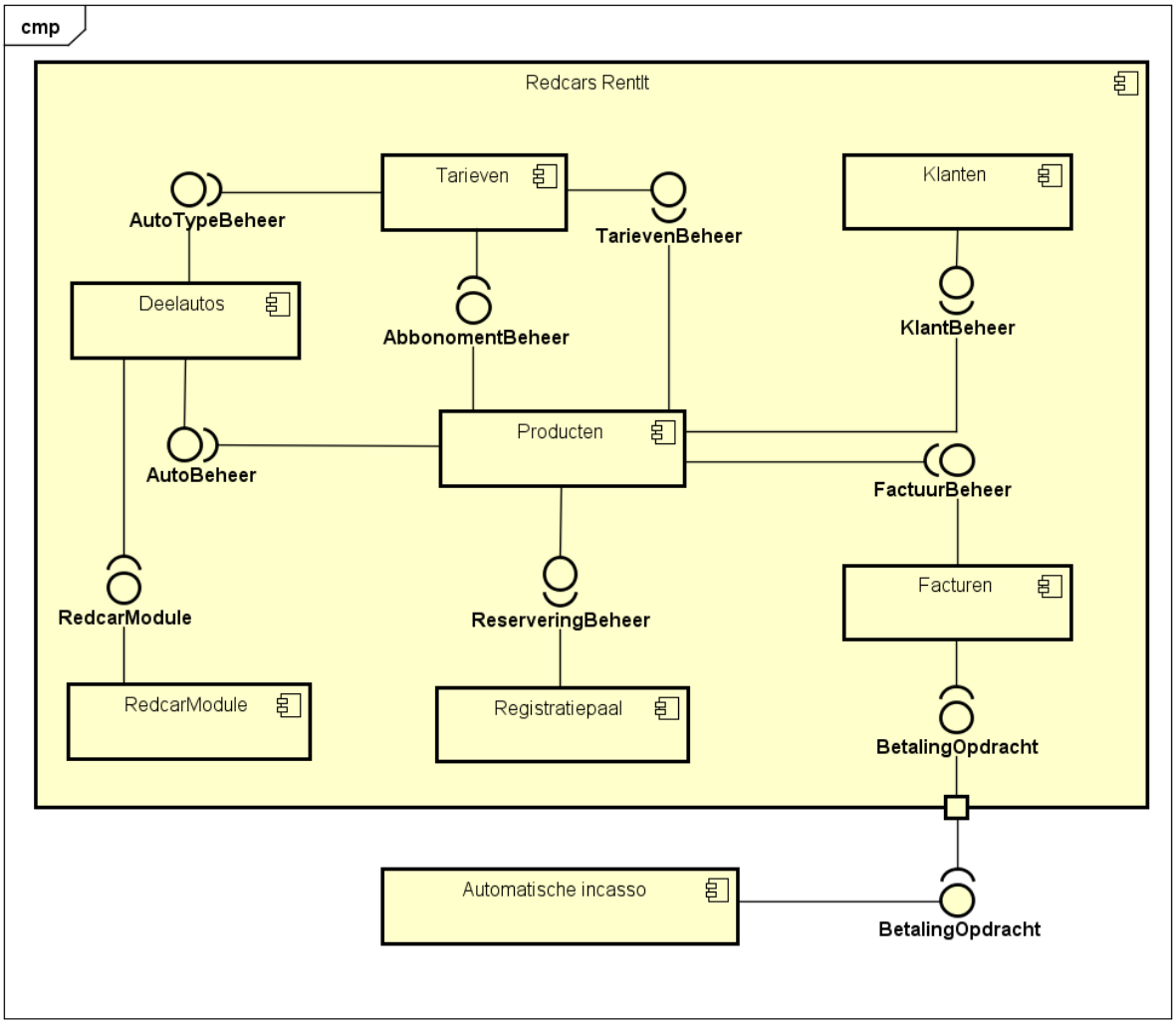
Een systeem bestaat vaak uit verschillende modules (componenten) om de gewenste functionaliteiten te kunnen realiseren. De UML-standaard kent een aantal verschillende modelleringstechnieken in het 'componenten' domein. Zo is er eerst een component diagram uitgewerkt waarin de verschillende componenten van het systeem in kaart worden gebracht. De communicatie tussen de verschillende componenten is gemodelleerd aan de hand van 'interfaces', welke de specificaties van de communicatie vastleggen.

Na het component diagram zijn er tabellen uitgewerkt waarvoor per component de aangeboden en benodigde interfaces aan de hand van een beredenering in kaart worden gebracht, en verduidelijking geeft over welke gegevens er uitgewisseld worden.

Gewapend met deze modellen kunnen de system sequence diagrams worden gedetailleerd, wat leidt tot component sequence diagrams. Hiermee wordt de blackbox, 'het systeem', vervangen door de in kaart gebrachte componenten. De communicatie stromen geven aan hoe, en met welke gegevens een functionaliteit tot stand komt. Al deze modellen samen vormen uiteindelijk de basis van het 'software ontwerp' oftewel het klasse diagram.

5.1 Component diagram

In het component diagram zijn de verschillende componenten van het systeem gemodelleerd waarbij de onderliggende relaties middels benodigde en aangeboden interfaces in kaart zijn gebracht.



Figuur 12-Componentendiagram

5.1.1 Omschrijving componenten

Hieronder worden de componenten in het diagram beschreven met hun provided interfaces en waarvoor de required interfaces nodig zijn.

5.1.1.1 Deelautos

	Beschrijving
Algemeen	Component Deelautos bevat alle informatie omtrent een auto. Concepten zoals deelauto, auto types, schademeldingen en standplaatsen worden binnen dit component opgenomen.
Provided interface(s)	
AutoTypeBeheer	Biedt mogelijkheid tot opvragen, toevoegen, verwijderen, en aanpassen van AutoType's.
AutoBeheer	Biedt mogelijkheid tot het opvragen, toevoegen, verwijderen en aanpassen van Deelauto's, Standplaatsen en Schademeldingen. Het communiceren met een RedCarModule behorend tot een deelauto
Required interface(s)	
RedcarModule	Benodigd om de gegevens van een RedcarModule behorend tot een specifieke deelauto te kunnen opvragen en aanpassen

Tabel 13-Componentbeschrijving deelautos

5.1.1.2 RedcarModule

	Beschrijving
Algemeen	Component RedCarModule bevat informatie over de redcarmodule in een fysieke deelauto.
Provided interface(s)	
RedcarModule	Biedt mogelijkheid tot opvragen en aanpassen van de gegevens van een RedCardModule
Required interface(s)	
Geen	

Tabel 14-Componentbeschrijving redcarmodule

5.1.1.3 Registratiepaal

	Beschrijving
Algemeen	Component Registratiepaal heeft de mogelijkheid om een ledenpas uit te lezen en deze informatie te communiceren richting de rest van het systeem.
Provided interface(s)	
Geen	
Required interface(s)	
ReserveringsBeheer	Benodigd om een reservering te kunnen opvragen, aanpassen en wijzigen aan de hand van een pasnummer

Tabel 15-Componentbeschrijving registratiepaal

5.1.1.4 Tarieven

	Beschrijving
Algemeen	Component Tarieven bevat alle informatie over auto tarieven, tarief soorten en abonnement prijzen. (product tarieven)
Provided interface(s)	
TarievenBeheer	Biedt mogelijkheid tot opvragen, toevoegen, verwijderen, en aanpassen van Tarieven en TariefSoorten.
Required interface(s)	
AutoTypeBeheer	Benodigd om AutoType's op te kunnen vragen
AbonnementBeheer	Benodigd om AbonnementType's op te kunnen vragen

Tabel 16-Componentbeschrijving tarieven

5.1.1.5 Producten

	Beschrijving
Algemeen	Component Producten bevat alle informatie over de producten binnen het systeem: Reserveringen, Verhuur en Abonnementen. Hierbij horen ook de boetes.
Provided interface(s)	
ReserveringBeheer	Biedt mogelijkheid voor het opvragen, toevoegen, verwijderen en aanpassen van Reserveringen en Verhuur.
AbonnementBeheer	Biedt mogelijkheid voor het opvragen, toevoegen, verwijderen en aanpassen van Abonnementen en Abonnement types.
Required interface(s)	
TarievenBeheer	Benodigd om de kosten voor een product te kunnen berekenen
KlantBeheer	Benodigd om Klanten op te vragen
AutoBeheer	Benodigd om gegevens van een auto te kunnen opvragen
FactuurBeheer	Benodigd om facturen van een klant te kunnen controleren (op wanbetaling e.d.)

Tabel 17-Componentbeschrijving productens

5.1.1.6 Klanten

	Beschrijving
Algemeen	Component Klanten bevat alle informatie over de klanten binnen het systeem.
Provided interface(s)	
KlantBeheer	Biedt mogelijkheid tot opvragen, toevoegen, verwijderen, en aanpassen van Klanten
Required interface(s)	
Geen	

Tabel 18-Componentbeschrijving klanten

5.1.1.7 Facturen

	Beschrijving
Algemeen	Component Facturen bevat alle informatie van de facturen binnen het systeem.
Provided interface(s)	
FactuurBeheer	Biedt mogelijkheid tot opvragen, toevoegen, verwijderen, en aanpassen van Facturen
Required interface(s)	
BetalingOpdracht	Benodigd om een betalingsopdracht (verzoek) tot automatisch incasso in te dienen

Tabel 19-Componentbeschrijving facturen

5.1.1.8 Automatisch incasso

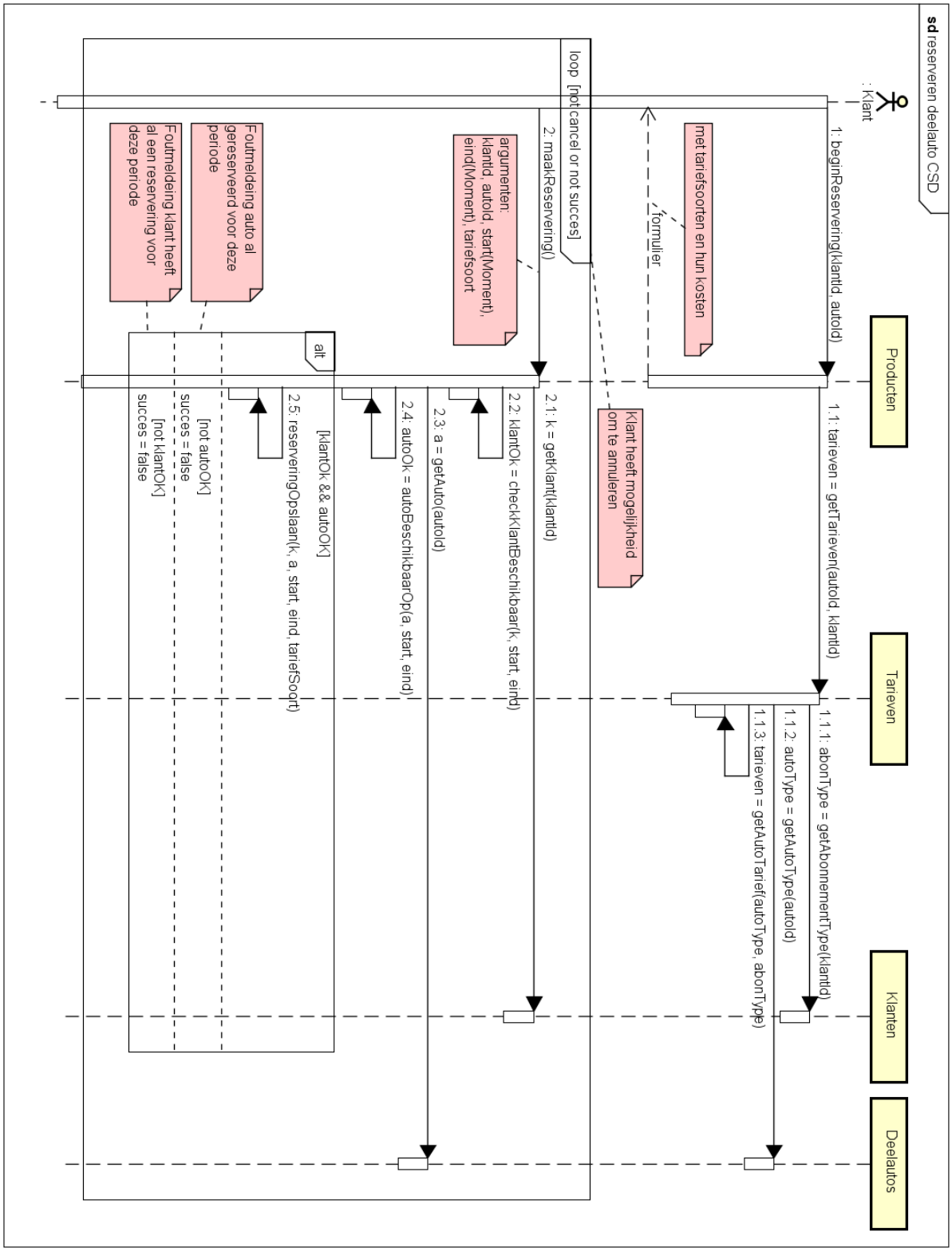
	Beschrijving
Algemeen	Extern Component Automatische incasso biedt mogelijkheid tot betalingsverzoeken.
Provided interface(s)	
BetalingOpdracht	Biedt mogelijkheid tot indienen van betalingsverzoek voor een automatische incasso
Required interface(s)	
Geen	

Tabel 20-Componentbeschrijving automatische incasse

5.2 Component sequence diagrams

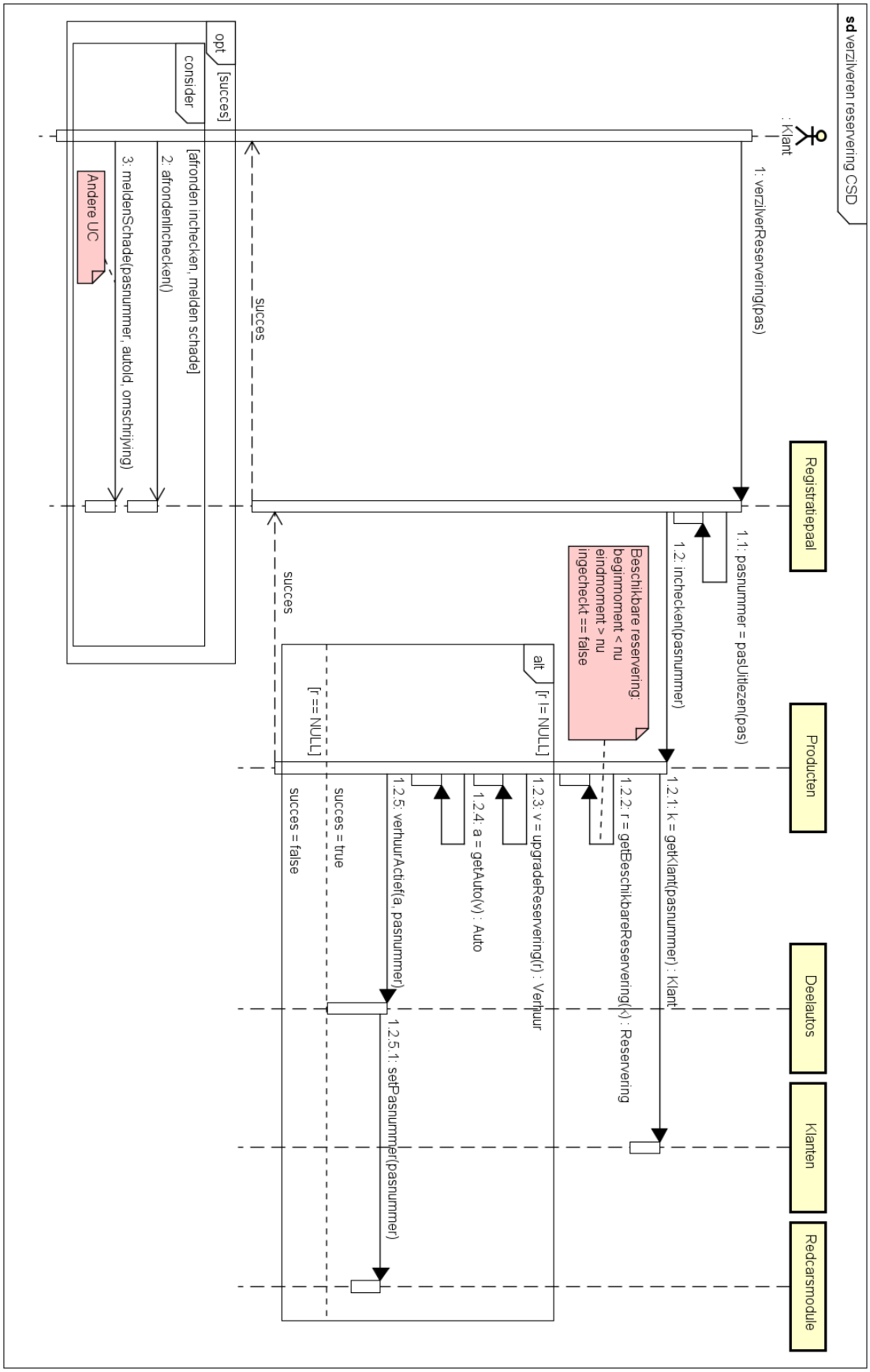
In dit hoofdstuk worden de system sequence diagrammen uit hoofdstuk 3.2 uitgewerkt in component sequence diagrammen. Hierbij wordt gebruik gemaakt van de bovenstaande componenten en worden de relaties tussen de componenten gebruikt.

5.2.1 Reserveren deelauto



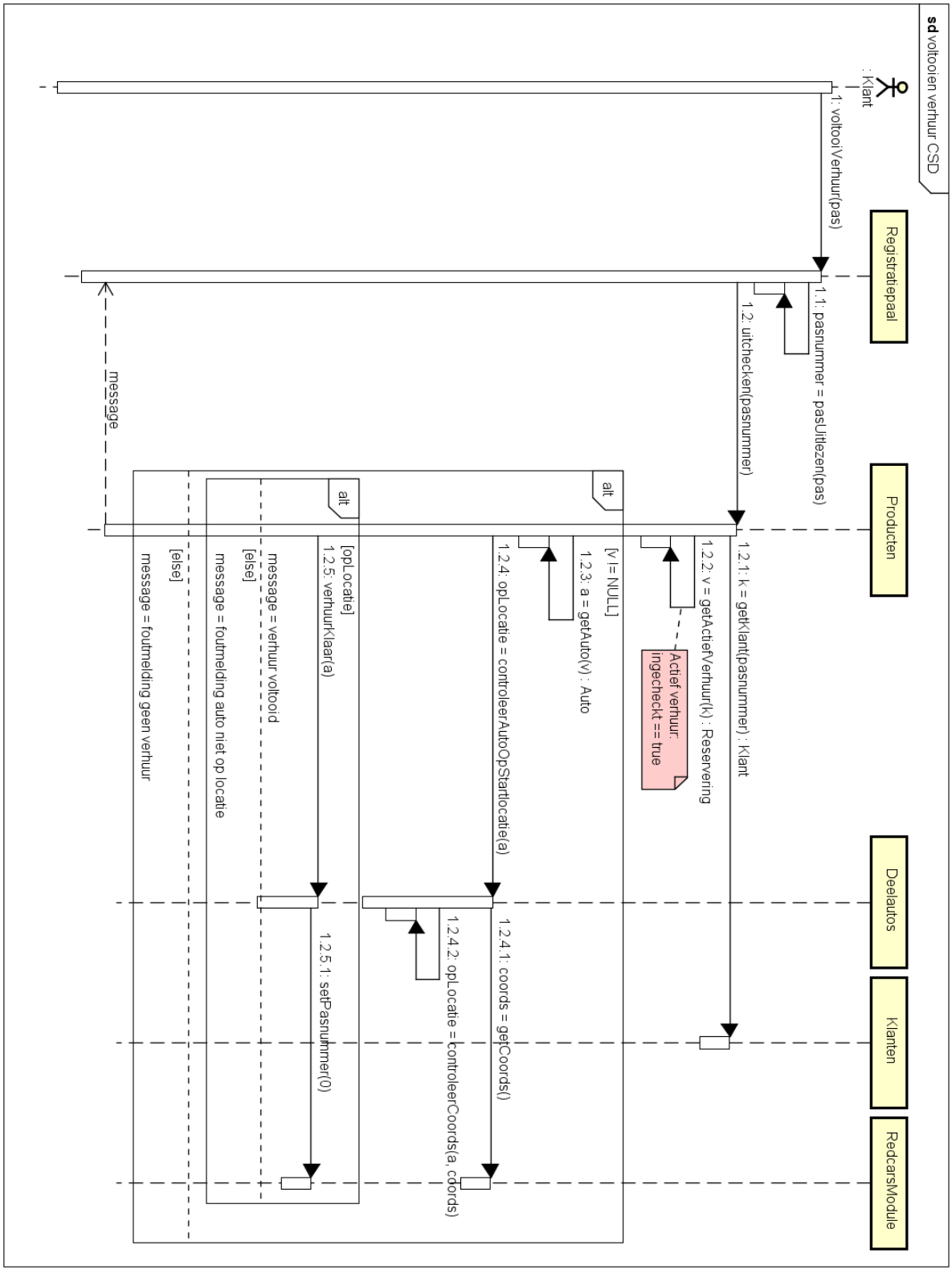
Figuur 13-CSD reserveren deelauto

5.2.2 Verzilveren reservering



Figuur 14-CSD verzilveren reservering

5.2.3 Voltooien verhuur



Figuur 15-CSD voltooi verhuur

5.2.4 Component Sequence diagrams CRUD Klant

De System Sequence Diagrammen van Beheer Klanten wordt niet verder uitgewerkt in component sequence diagrammen. De reden hiervoor is dat deze diagrammen hetzelfde zouden zijn als de system sequence diagrams, maar met het systeem vervangen met het klanten component.

Deze diagrammen worden wel (waar nodig) uitgewerkt in sequence diagrams in het volgende hoofdstuk.

6 Software ontwerpen

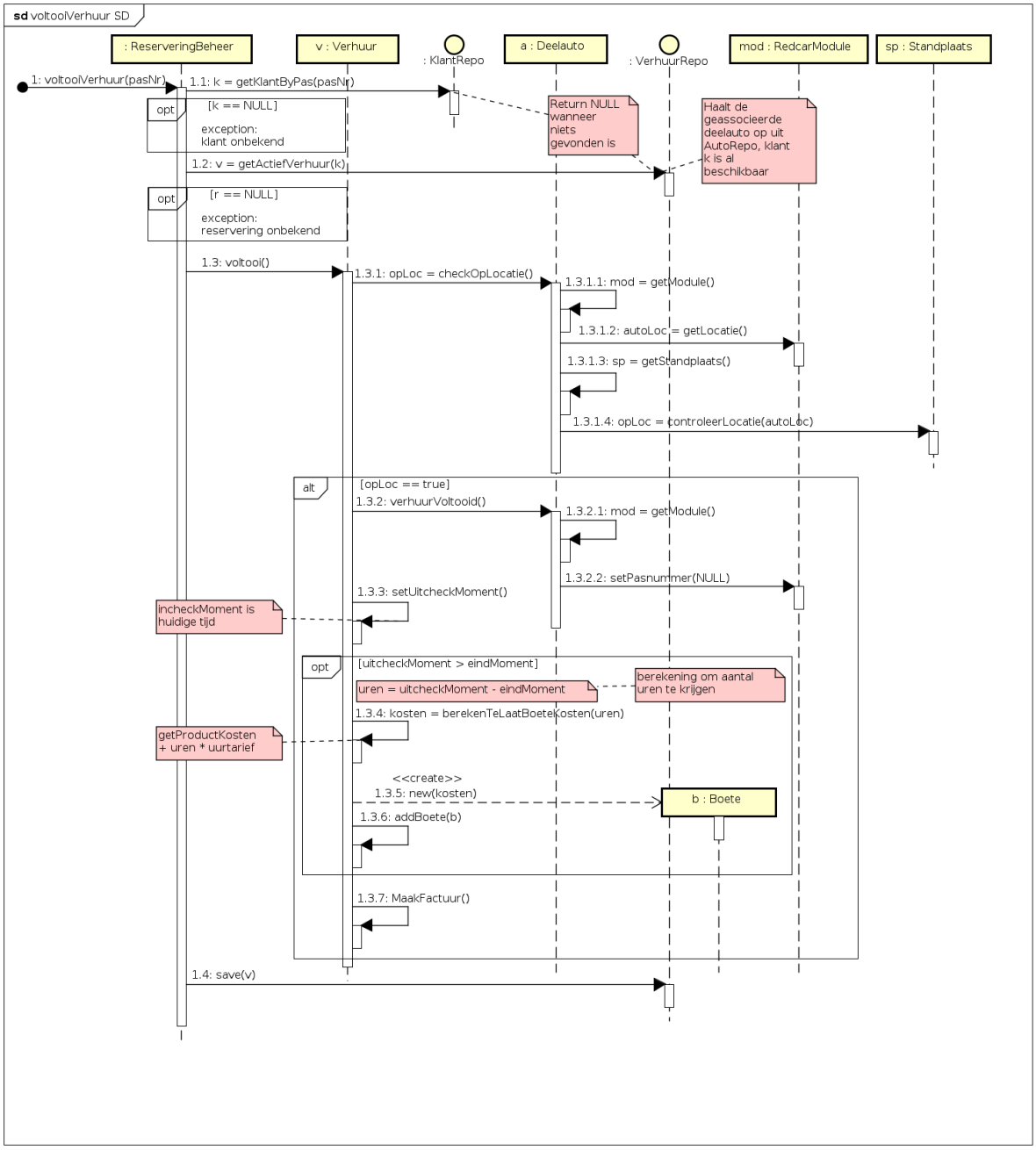
In dit hoofdstuk worden de ontwerpen omschreven die gebruikt gaan worden om de applicatie te realiseren. Eerst worden de interessante calls uit binnen de component sequence diagrams uitgewerkt in sequence diagrams. Hierbij wordt gebruik gemaakt van de concepten uit het domeinmodel. Op basis van deze sequence diagrammen wordt een design class diagram opgesteld, waarbij rekening wordt gehouden met eventuele implementaties van design patterns. De toegepaste patterns worden daarna verder uitgelegd.

6.1 Sequence diagrams

Hier zijn een aantal calls uit de component sequence diagrammen uitgewerkt tot sequence diagrammen, voor het reserveren, verzilveren en voltooien. Ook zijn er sequence diagrammen voor het berekenen van de kosten uitgewerkt.

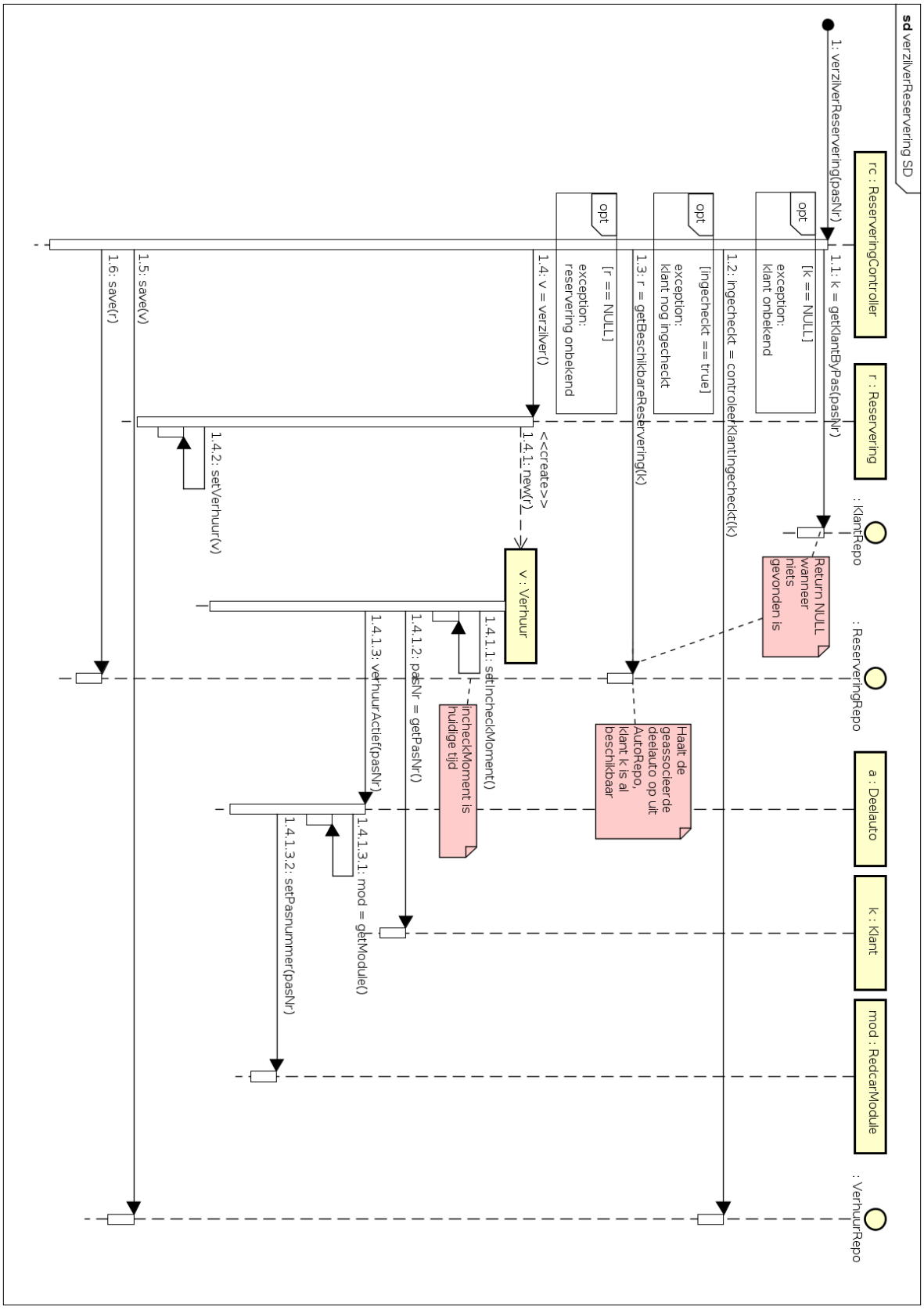
Bij het maken van de sequence diagrammen is rekening gehouden met GRASP design patterns, zoals high cohesion, low coupling en information expert.

6.1.1 Voltooi verhuur



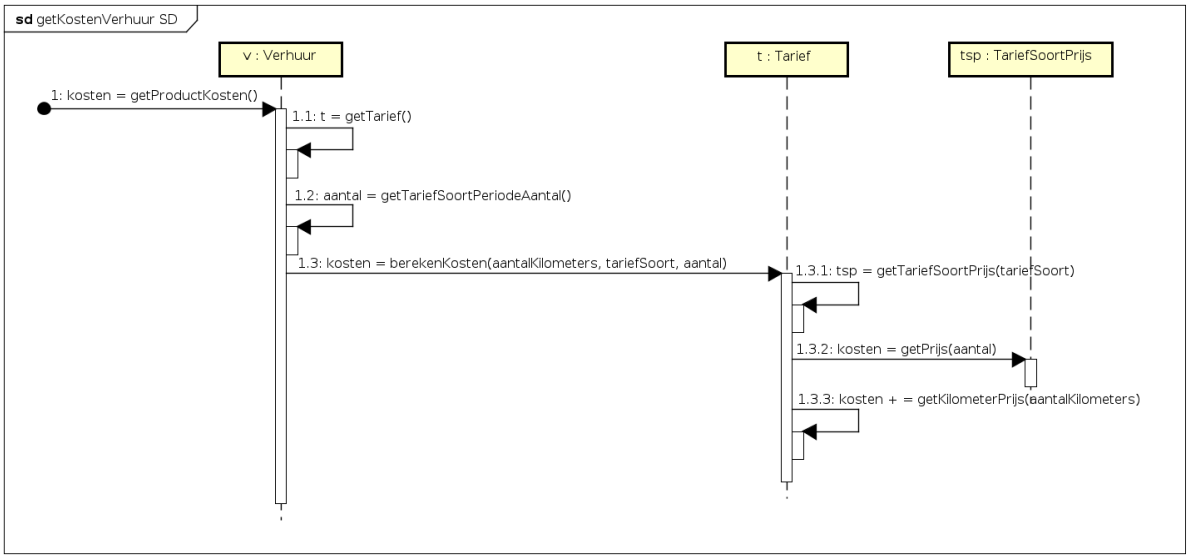
Figuur 16-SD voltooi verhuur

6.1.2 VerzilverReservering



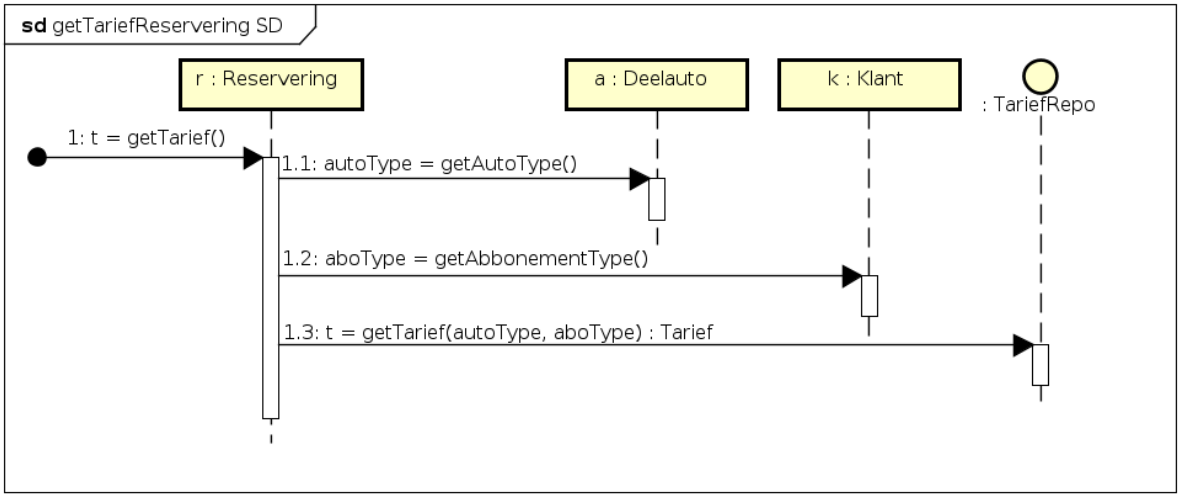
Figuur 17-SD verzilver reservering

6.1.3 getKosten



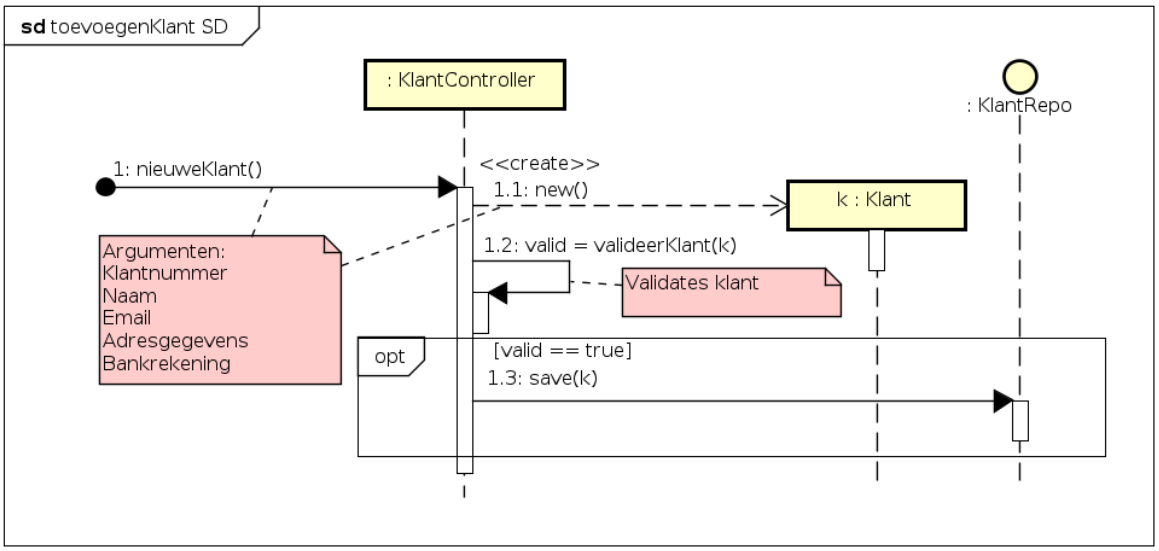
Figuur 18-SD get kosten

6.1.4 getTarief



Figuur 19-SD get tarief

6.1.5 toevoegenKlant



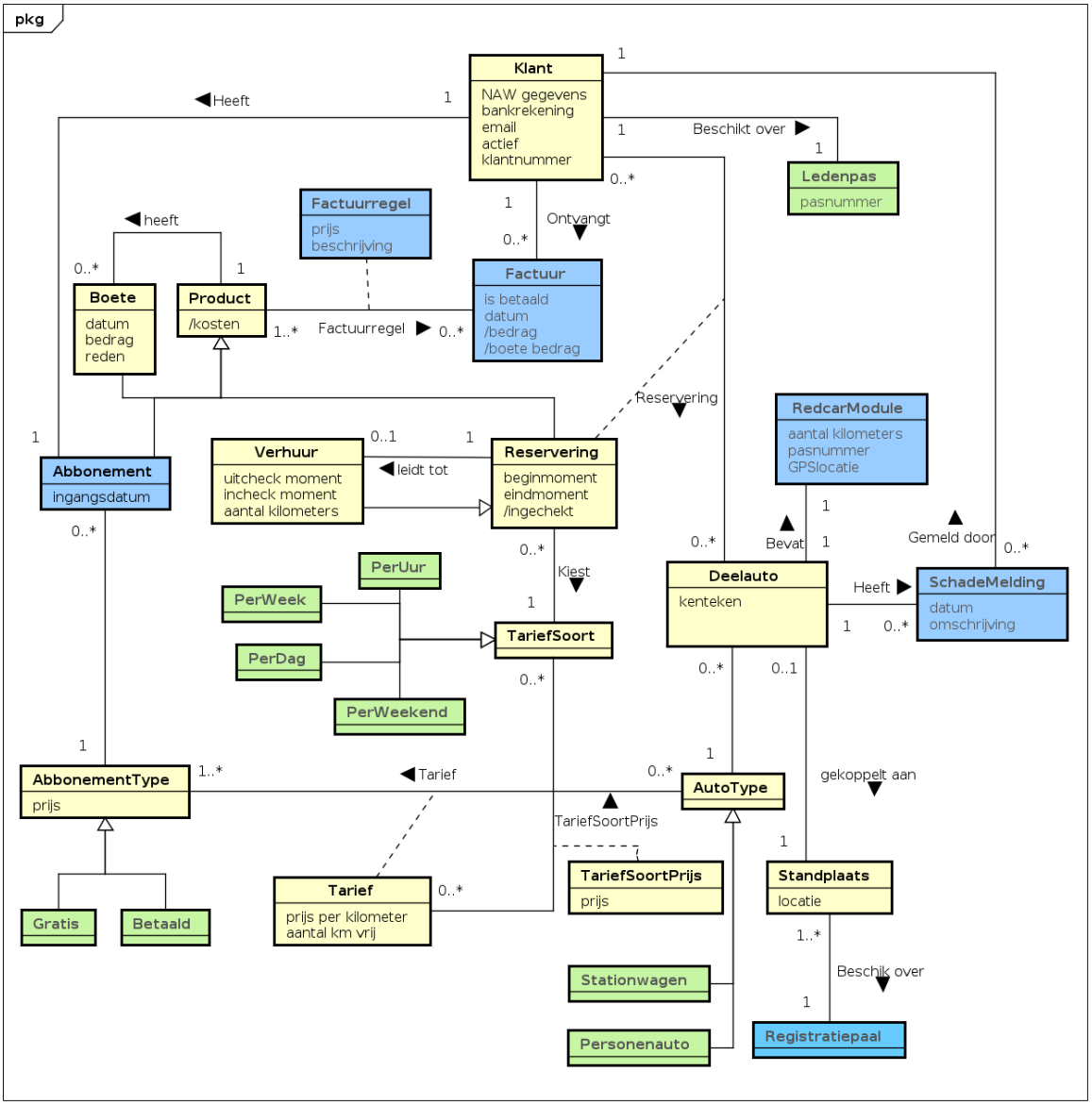
Figuur 20-SD toevoegen klant

6.2 Klassendiagram

In dit hoofdstuk wordt er op basis van het domeinmodel en de bovenstaande sequence diagrammen een klassendiagram ontworpen. Hierbij wordt eerst een terugkoppeling gemaakt naar het domeinmodel, om aan te geven welke onderdelen gemodelleerd zullen worden in het klassendiagram.

Om het leesbaar te houden wordt het klassendiagram opgedeeld in 3 klassendiagrammen. Dit is niet op basis van componenten (namespaces), maar op een logische verdeling van samenhang: Core, dataopslag en datamodels.

6.2.1 Terugkoppeling domeinmodel

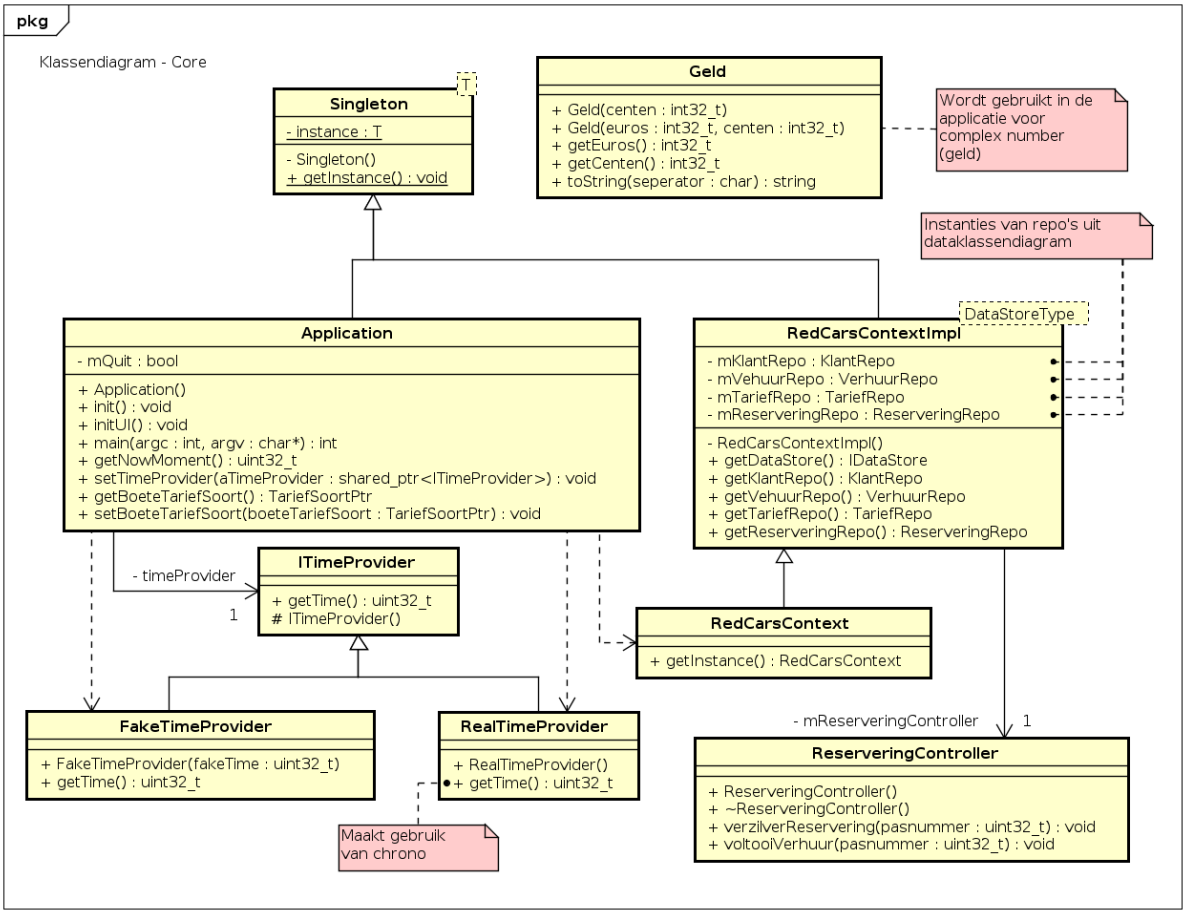


Figuur 21-Domeinmodel: beschrijving klassendiagram

Groene concepten zijn in het klassendiagram verwerkt in andere concepten. Ledenpas heeft een pasnummer, deze is in klant opgenomen. De verschillende types en soorten zijn niet als klassen gemodelleerd, omdat dit de uitbreidbaarheid limiteert.

Blauwe concepten vallen voor de huidige iteratie (iteratie 1) buiten het ontwerpmodel. Deze zullen niet ontwikkelt worden en waar nodig gemockt worden. Klant zal alleen een abonnementtype hebben en geen abonnement, RedCarModule wordt gemockt.

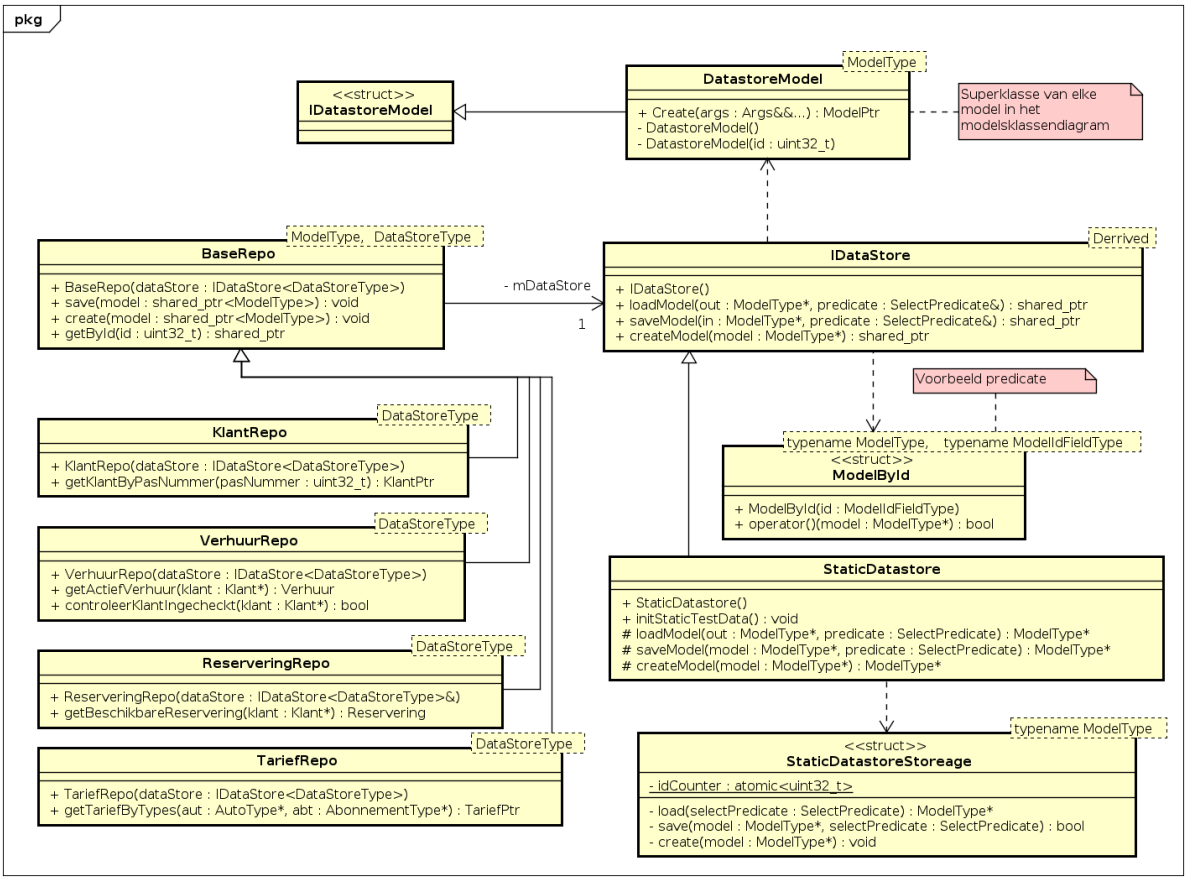
6.2.2 Klassendiagram Core



Figuur 22-Design class diagram - core

De core namespace bevat voornamelijk zaken die de maken hebben met het onderliggende platform/os (CLI interface) in ons geval. Tijd bijvoorbeeld is iets dat aangeleverd wordt door het platform. Wij hebben besloten om tijd via een Strategy pattern te realiseren. Waar `ITimeProvider` de interface is met een methode om de huidige tijd te verkrijgen. Wij hebben hiervoor gekozen omdat dit handig is bij het 'mocken' van de data. Application is de klasse (of controller) welke het starten van de applicatie verzorgt. Het voort de nodige initialisaties uit om het systeem klaar voor gebruik te maken. Verder hebben we deze klassen gebruikt om wat globale zaken omtrent de applicatie te regelen, zoals de eerder beschreven `ITimeProvider.` De RedCarsContext is vanuit ons ontwerp gezien de hoofdklassen van het systeem van redcars. Het bevat alle interfaces naar de verschillende componenten. Er is bersloten om dit in een op zichzelf staande klasse te stoppen zodat het platform onafhankelijk kan worden ingezet. Verder is te zien dat de teamplteklasse RedCarsContextImpl de daadwerkelijke implementatie bevat van de context middels de aangegeven templateklasse. Tot slot bevat het klassen, "bouwstenen" om bijvoorbeeld een design pattern toe te passen.

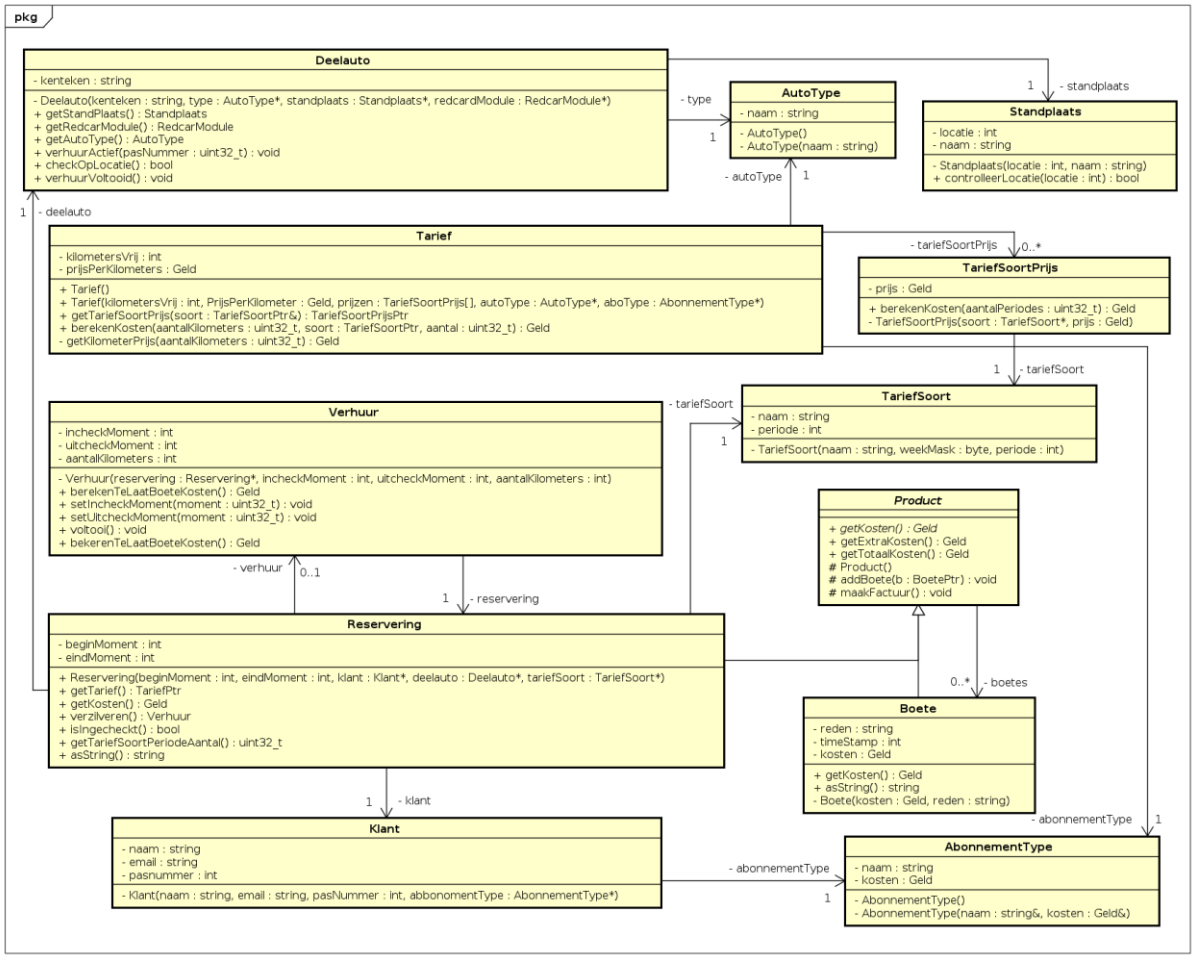
6.2.3 Klassendiagram Datastoring



Figuur 23-Design class diagram - datastore

Het ontwerp van de Datastore heeft enige tijd gekost. Het leek ons dat hier de strategy pattern een logische keuze zou zijn, zodat er verschillende implementaties zouden kunnen worden gebruikt. Maar er was vrij snel een probleem, hoe kan je een abstracte interface maken die een in theorie nog niet bestaande datamodel objecten zou kunnen ondersteunen. De normale manier van polymorfisme toepassen waarbij je een abstract basis klasse hebt met een aantal virtual methodes, het concept definieert, werkt niet als je in theorie met 'oneindig' veel objecten moet werken. C++ heeft mooi gereedschap voor dit soort probleem, namelijk template classes. Ondersteuning voor generiek programmeren. Wij moesten de oplossing in deze hoek zoeken. Wij hebben een eigenlijk polymorfisme toegepast door gebruikt te maken van **CRTP**, meer hierover is te lezen in hoofdstuk Curiously recurring template pattern.

6.2.4 Klassendiagram Models



Figuur 24-Design class diagram - models

In dit diagram zijn de data models binnen de applicatie weergegeven. Deze structuur komt op veel vlakken overeen met het domeinmodel. De functies die in de sequence diagrammen zijn beschreven, zijn in dit klassendiagram te vinden.

Om het diagram leesbaar te houden is er gekozen om de dependencies buiten het diagram te beschrijven. Hieronder staan de ontbrekende afhankelijkheden per klasse opgesomd:

- Reservering
  - Tarief
  - AutoType
  - AbonnementType
- Tarief
  - TariefSoort
- Verhuur
  - TariefSoort
  - Tarief
  - Boete

6.3 Toepassing design patterns

Bij de overgang van domein model naar designklassen diagram zijn er een aantal design patterns toegepast. Dit hoofdstuk beschrijft elke pattern en hoe deze toegepast is.

6.3.1 Singleton

Singleton is een handige patten als je een klasse hebt waar je eigenlijk altijd maar 1 instantie va zult hebben. Verder is het ook handig omdat je altijd makkelijk de instantie van de klasse kan vinden door de statische getInstance methode aam te roepen. Een ander bijkomend voordeel is dat er gebruikt gemaakt wordt van 'Lazy Initialization'. Waar dit op neer komt is dat de het object pas bij de eerste call naar getInstance zal worden aangemaakt. Maak je geen gebruik van de klasse, 'betaal' je er ook niet voor. Wij hebben deze pattern op 2 klassen toegepast. Application, het tekstboekvoorbeeld voor het gebruik van een singleton. Een klasse waar je maar 1 instantie van wilt hebben, en er handig dat je altijd bij de instantie kan. Bij de uitleg van het klassendiagram ben ik verder ingegaan op de keuzes voor RedCarContext en zal deze niet herhalen.

6.3.2 Facade

De reserveringcontroller zorgt voor de functionaliteit reserveringen te verzilveren en verhuringen te voltooien. In deze functies wordt via de datastore de benodigde data opgehaald, gevalideerd en uitgevoerd.

6.3.3 Factory method

De factory methode strategy kan je inzetten als je specifieke eisen hebt voor het aanmaken van objecten. Deze pattern hebben wij ingezet bij DatastoreModel objecten. Om te zorgen dat wij een simpele statische opslagplaats klassen konden maken, was de policy dat alle verwijzingen naar andere objecten via een shared_ptr verlopen. Hiermee is een de ellende van handmatig memory bookkeeping opgelost. DatastoreModel erft over van std::enabled_shared_from_this, een basisklassen waarmee je vanuit je eigen object de controle hebt over de lifetime. Om deze policiy te forceren hebben wij in de basisklasse DatastoreModel de Factory method 'Create' gemaakt. De constructor van de basisklassen wordt op private gezet, waarmee alleen de Factory method nog objecten kan aanmaken.

6.3.4 Strategy

Binnen de applicatie worden een aantal strategieën toegepast, binnen de datastore, het ophalen van tijd en het berekenen van de kosten voor reserveringen.

De datastore, die zorgt voor de opslag en opvraging van data, is de grootste strategy in applicatie. Bij het initiëren van de repo's wordt de datastore meegegeven. Een repo voert bijvoorbeeld loadModel uit, om het model uit de datastore te laden. Hierbij kan een predicate meegegeven worden zodat de datastore zoekt naar het model dat hieraan voldoet. De staticDataStore haalt het model dan uit zijn static, predefined data, maar een toekomstige datastore met link naar een database kan deze ui de database halen.

De predicate binnen de DataStore is ook een soort strategy, maar deze heeft geen base klasse. De predicatie moet uitgevoerd kunnen worden, dus voor complexe queries kan een struct gebruikt worden die een () operator implementeert. Voor de meeste predicates gewoon wordt een lambda gebruikt.

De applicatie maakt gebruik van tijd, voor onder andere het reservergings startmoment, eindmoment en factuurdatum. Deze tijd is op te halen vanuit de Application singleton. Om de applicatie gemakkelijk te kunnen testen, is er een strategy geïmplementeerd, ITimeProvider met de

functie getTime, om zowel de echte tijd, als een gefabriceerde tijd op te halen. RealTimeProvider retourneert de huidige tijd en FakeTimeProvider krijgt een tijd mee in de constructer om te retourneren.

Om voor reserveringen de kosten te berekenen, wordt uit de datastore een tarief opgehaald op basis van deelauto type en abonnement type. Het Tarief object wordt dan gebruikt als een strategy om de kosten te berekenen voor de reservering.

6.3.5 Curiously recurring template pattern

Het werkt als volgt: er wordt overerft van een basis templateklasse en geeft jouw klasse as type mee. Hiermee weet de template base klasse dus de class die van hem overerft, en meestal de implementatie levert. Dit kan je handig inzetten om een statische vorm van polymorfisme toe te passen, de base kan namelijk methodes van zijn subklasse aanroepen. Wij hebben deze strategy om deze rede toegepast om de implementatie van IDataStore te maken. Een statische variant van de staretgy pattern.

Een ander mooi voorbeeld van het gebruik van deze strategy is bij de Singleton templateklasse. Ook hier gaat het erom dat de baseklasse functionaliteiten kan leveren die alleen mogelijk zijn als je het type van je subklasse weet. Singleton definieert een statische methode getInstance, waarmee de enige instantie van de klasse kan worden opgehaald. Erg handig hulpmiddel.

7 Kanttekeningen iteratie 1

7.1 Usecases, system sequence en activity diagrammen

- Er is geen rekening gehouden met wanbetalers bij verzilveren reservering.

7.2 Domein model, componenten diagram en component sequence diagrammen

- Geen opmerkingen

7.3 Sequence diagrammen en klassendiagram

- Er was een idee dat Verhuur een subklasse zou zijn van Reservering. Dit is uiteindelijk niet gebeurt en resulteert nu in high coupling binnen de verhuur klasse. Om dit op te lossen kan reservering meer functionaliteiten opvangen, zodat verhuur deze uitvoert op reservering.
- Product klasse had eigenlijk de link moeten hebben met persoon, in plaats van reservering. Op deze manier kunnen toekomstige producten ook gelijk aan klanten gekoppeld worden en kan het factuur vanuit product worden aangemaakt.

TABELLEN

FIGUREN