OSM-Modeling Casus

Tim Klaassen(568583) & Jochem Grootherder(598765)

Hogeschool van Arnhem en Nijmegen

OSM-M Richard Holleman

01-11-2018

Inhoud

[1. Inleiding 2](#_Toc528782791)

[2. Use case diagram 3](#_Toc528782792)

[2.1 Brief description 3](#_Toc528782793)

[3. Niet functionele eisen 5](#_Toc528782794)

[4. Componenten diagram 6](#_Toc528782795)

[5. Deployment Diagram 7](#_Toc528782796)

[6. Domeinmodel 8](#_Toc528782797)

[7. Sequence diagrammen 10](#_Toc528782798)

[8. Design class diagram 15](#_Toc528782799)

[9. Applicatie 16](#_Toc528782800)

# Inleiding

In dit document staan alle gemaakt opdrachten voor de casus van OSM-M.

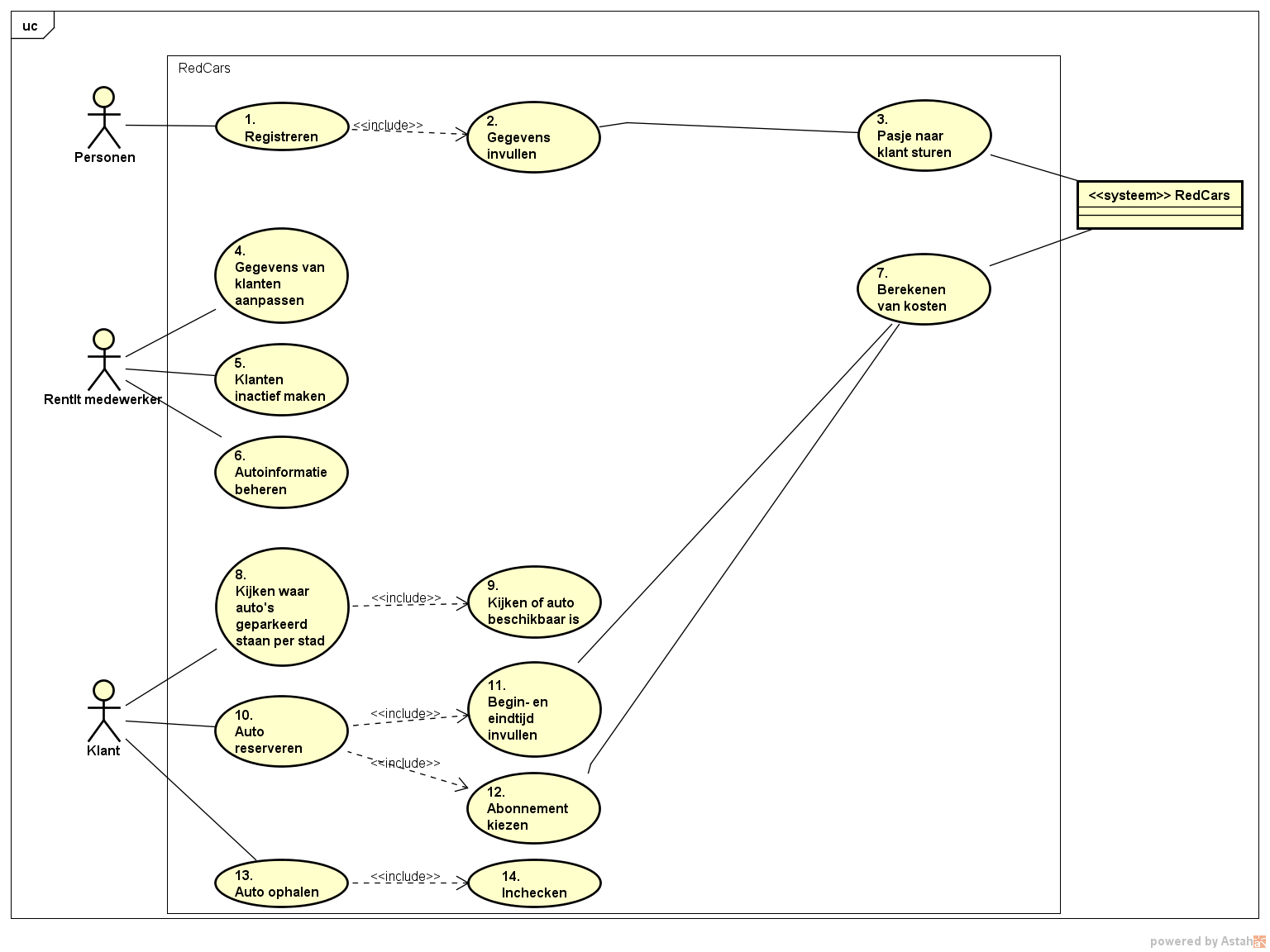
Bij alle gemaakte opdrachten staat een beknopte uitleg over wat het doel van de opdracht is en in welke vorm deze gemaakt is.

Na het lezen van dit document is de lezer bekend met alle gemaakte opdrachten en het beschreven systeem in de casus en zijn werking. Voor het systeem zijn extra vragen gesteld aan de opdrachtgever. Wanneer er naar aanleiding van de klant iets is aangepast word dit aangegeven. Ook wanneer er aannames gemaakt worden, wordt dit aangegeven

# Use case diagram

Een use case diagram is een simpele representatie van hoe een gebruiker op het systeem kan inwerken. Een use case diagram kan de verschillende types van gebruikers die het systeem zullen gebruiken identificeren. Een use case diagram wordt gebruikt om een “higher level view” van het systeem te krijgen. Dit betekent dat het een versimpelde en grafische versie van wat het systeem moet doen.

Het doel van deze opdracht was om een simpel beeld te krijgen van wat er uiteindelijk allemaal moet gebeuren.



## 2.1 Brief description

1. Registreren. Deze use case betekent dat een persoon zich moet kunnen registreren. Deze use case heeft use case 2 nodig anders weet hij niet welke gegevens de klant wilt gebruiken om zich te kunnen registreren.

2. Gegevens invullen. Deze use case houdt in dat wanneer een persoon zich wilt registeren hij zijn gegevens in moet vullen zodat hij deze later weer kan gebruiken om in te loggen.

3. Pasje naar klant sturen. Deze use case houdt in dat wanneer een persoon zich geregistreerd heeft en zijn gegevens ingevuld heeft een pasje thuisgestuurd krijgt door het systeem. Dit pasje moet de klant later gebruiken wanneer hij een auto wilt gebruiken

4. Gegevens van klanten aanpassen. Deze use case houdt in dat een RentIt medewerker de gegevens van een klant aan kan passen. Dit is er voor het geval de klant iets verkeerd gedaan heeft en dit opgelost moet worden.

5. Klanten inactief maken. Deze use case is er voor RentIt medewerkers om te zorgen dat klanten die zich niet aan de regels gehouden hebben inactief gemaakt kunnen worden. Dit is er zodat wanneer een klant zich niet aan de regels gehouden heeft hij geen gebruik meer zal kunnen maken van het systeem.

6. Autoinformatie beheren. Deze use case is er voor RentIt medewerkers om gegevens van auto’s aan te passen. Dit zal bijvoorbeeld gebeuren wanneer er bepaalde data van een auto niet meer klopt.

7. Berekenen van kosten. Deze use case wordt uitgevoerd door het systeem. Het houdt in dat de kosten die gemaakt worden door een klant bij elkaar opgeteld worden en verekend worden bij de klant. Dit gebeurt alleen nadat een klant een auto gereserveerd heeft.

8. Kijken waar auto’s geparkeerd staan per stad. Deze use case is er voor de klanten. Het houdt in dat een klant in staat moet zijn om te kunnen zien waar elke auto geparkeerd staat.

9. kijken of de auto beschikbaar is. Deze use case is er voor de klanten. Het houdt in dat wanneer een klant kijkt naar waar elke auto staat ook moet kunnen zien welke van deze auto’s beschikbaar is.

10. Auto reserveren. Deze use case is er voor de klanten. Het houdt in dat klanten in staat moeten zijn om een auto te reserveren. Dit is er zodat de klant later ook daadwerkelijk gebruik kan maken van een auto.

11. Begin- en eindtijd invullen. Deze use case is er voor de klanten. Dit is er zodat wanneer een klant een reservering wilt plaatsen hij in kan vullen van wanneer tot wanneer de klant de auto wilt gebruiken.

12. Abonnement kiezen. Deze use case is er voor de klanten. Dit wordt gebruikt wanneer een klant een reservering wilt plaatsen. Hier kan de klant voor een abonnement kiezen wat voor hem het meest gepast lijkt.

13. Auto ophalen. Deze use case is er voor klanten die een auto gereserveerd hebben en deze auto willen gebruiken binnen de tijd dat ze de auto gereserveerd hebben.

14. inchecken. Deze use case is er voor klanten die een auto op willen halen. Wanneer een klant een auto ophaalt moet hij eerst ingecheckt zijn, omdat een klant een auto niet mag gebruiken zonder ingecheckt te zijn.

# Niet functionele eisen

Hieronder staan een aantal niet functionele eisen die geïnterpreteerd kunnen worden door het lezen van de casus. Deze eisen zijn georganiseerd per alinea van de tekst en ze zijn volgens FURPS gerangschikt. FURPS is een acroniem dat een model weergeeft om kwaliteitseigenschappen van software in te delen in functionele en niet functionele eisen. Hierin staat de F voor functionaliteit, de U voor Usability, de R voor reliability, de P voor performance en de S voor supportability.

Beheer van gegevens:

Van auto’s moet kenteken, type en standplaats vastgelegd zijn. Usability  
Hiervoor is gekozen voor Usability omdat het vastleggen van gegevens onderdeel is van documentatie en dat valt onder usability.

Gebruik van de auto:

Op de website moet staan waar auto’s geparkeerd staan. Usability  
Hiervoor is gekozen voor Usability omdat het vastleggen van gegevens onderdeel is van documentatie en dat valt onder usability.

Het bekijken waar auto’s staan moet binnen 1 minuut kunnen, 90% van de tijd Performance  
Hiervoor is gekozen voor performance omdat dit iets zegt over de snelheid en dat valt onder performance.

Op de website moet staan welke auto’s beschikbaar zijn. Usability  
Hiervoor is gekozen voor usability omdat het vastleggen waar auto’s staan onderdeel is van documentatie en dat valt onder usability.

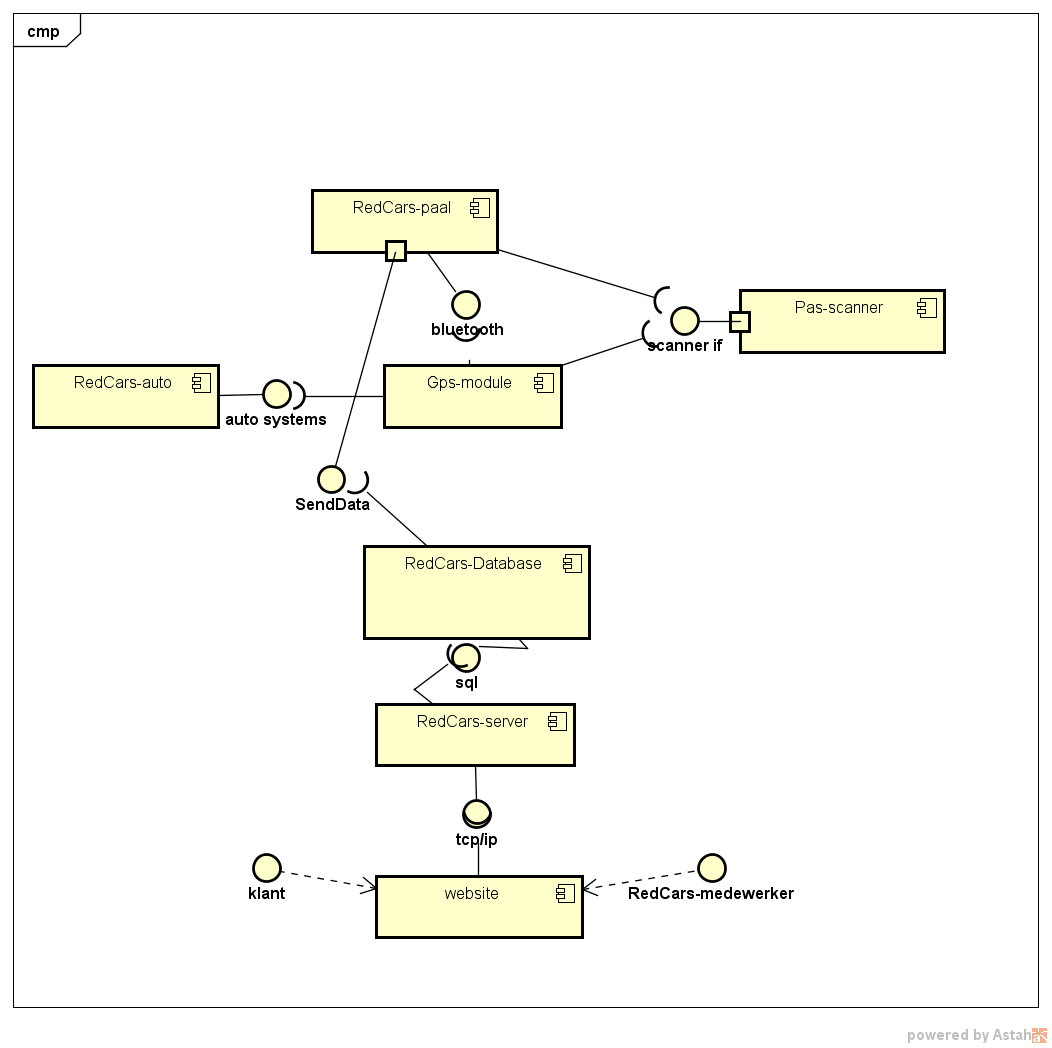
Het bekijken welke auto’s beschikbaar zijn moet binnen 1 minuut kunnen, 90% van de tijd  
Hiervoor is gekozen voor performance omdat het iets zegt over de snelheid en dat valt onder performance.

Performance

# Componenten diagram

Een component diagram is een diagram waarin de structurele relaties tussen componenten van een systeem weergeven worden. Componenten zijn gemaakt van software objecten die hetzelfde doel hebben. Alle componenten communiceren met elkaar via interfaces.

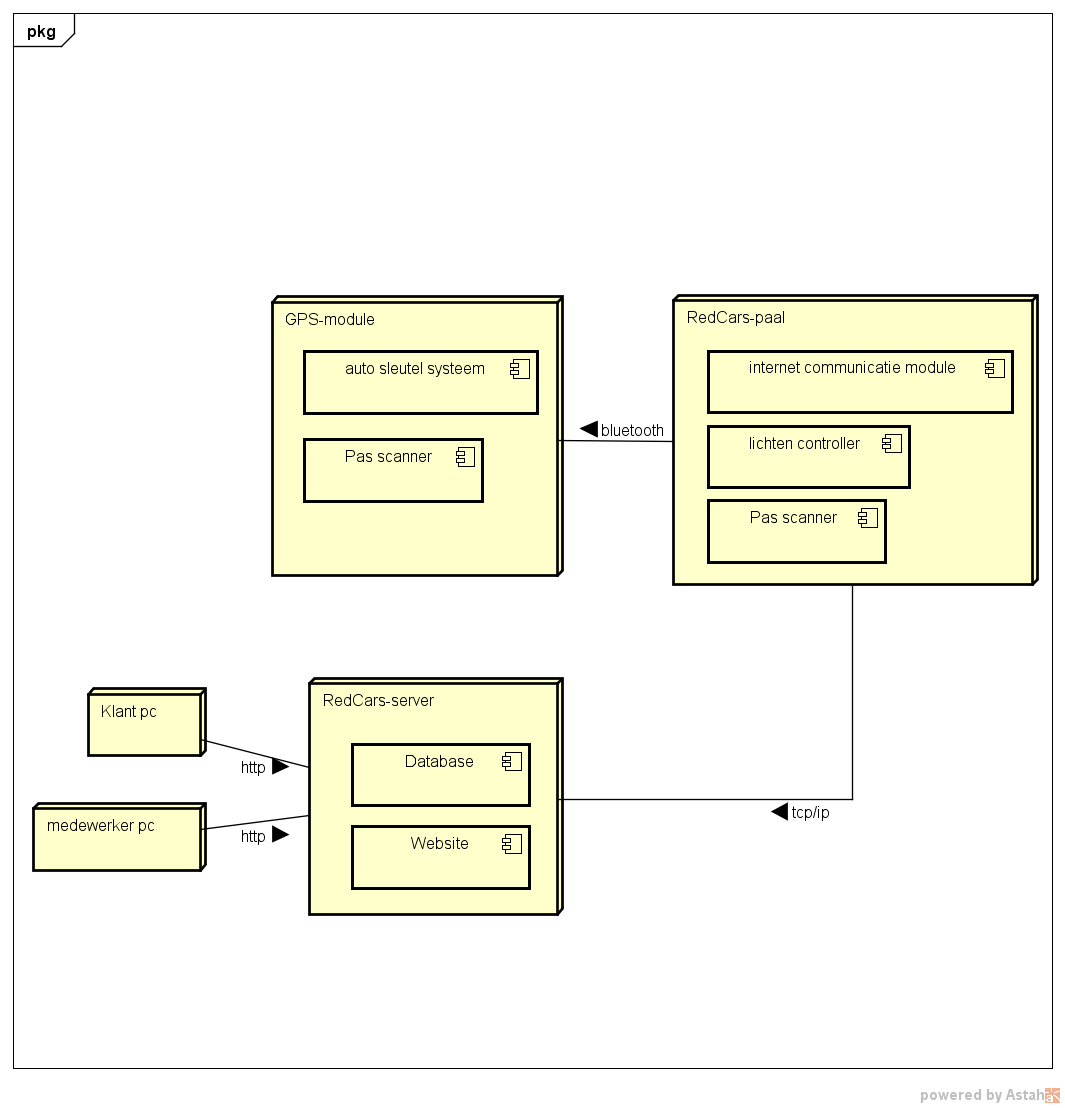
Hieronder staat het componenten diagram voor Redcars. Hierin staat welke componenten er allemaal benodigd zullen zijn voor het bedrijf RedCars om alles te realiseren wat er bereikt moet worden.



# Deployment Diagram

Een deployment diagram laat de configuraties zien van de run time processing nodes en de componenten die daarin bestaan. Deployment diagrammen worden gebruikt voor het modeleren van het statische deployment overzicht van een systeem. Dit is grotendeels het modeleren van de topologie van de hardware waar het systeem op uitvoert.

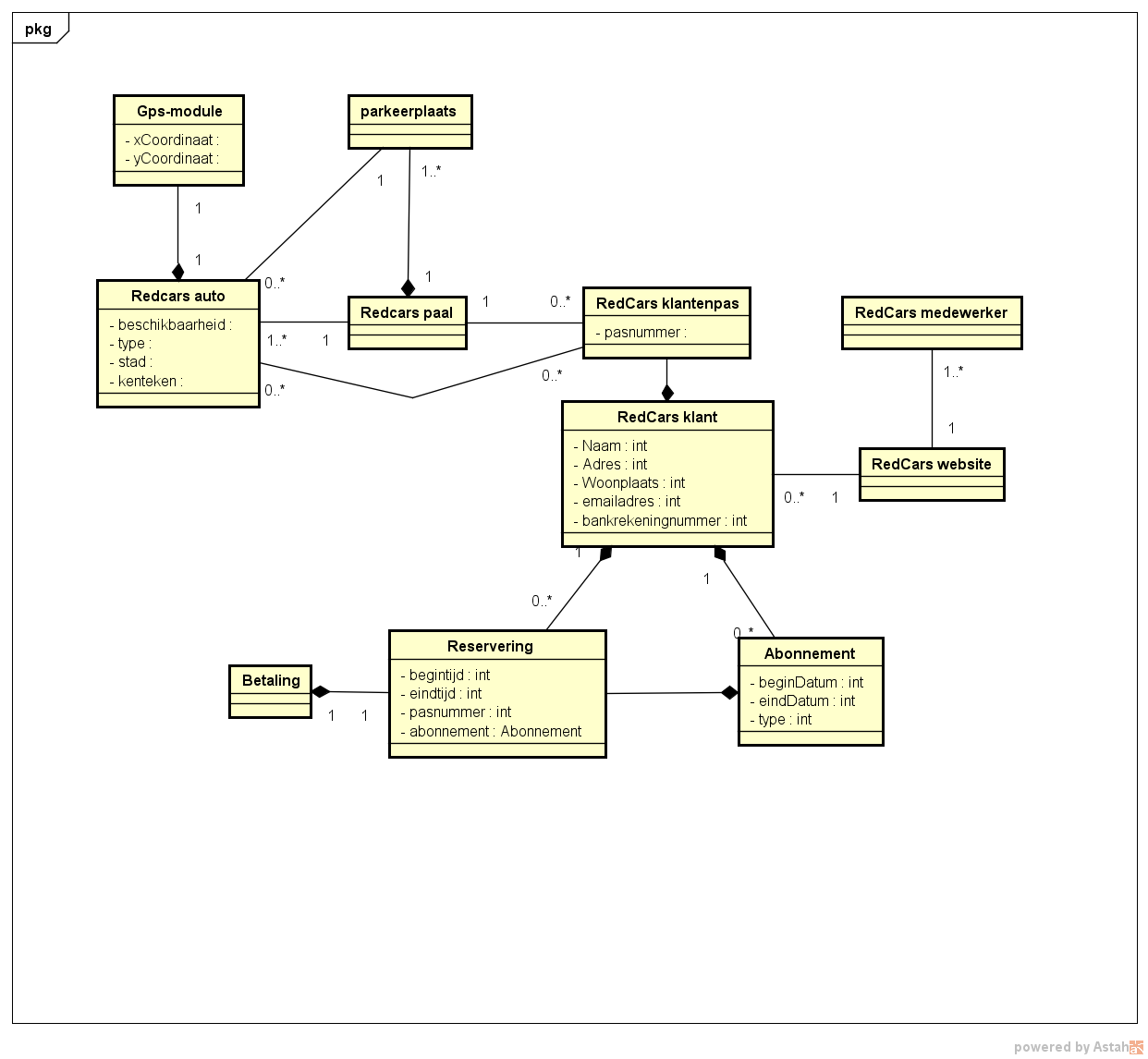
Hieronder staat het deployment diagram voor RedCars. Hierin staat welke fysieke objecten die geheugen of processing geschiktheid hebben. Ook staat erin hoe alle objecten met elkaar verbonden zijn en welke componenten in de objecten bestaan.



# Domeinmodel

Een domein model is een manier om dingen die in de echte wereld bestaan te beschrijven en modeleren en de relaties ertussen te tonen.

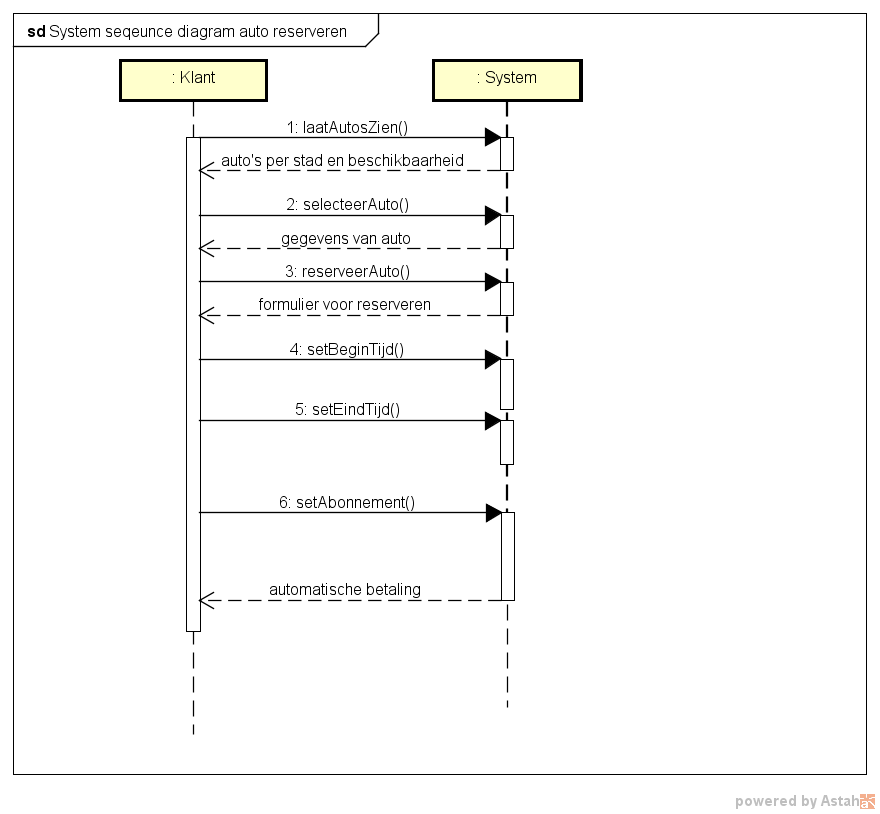
Hieronder staat het domein model voor RedCars. Hierin is alles wat binnen het domein van RedCars valt opgenomen met de relaties en multipliciteit ertussen.

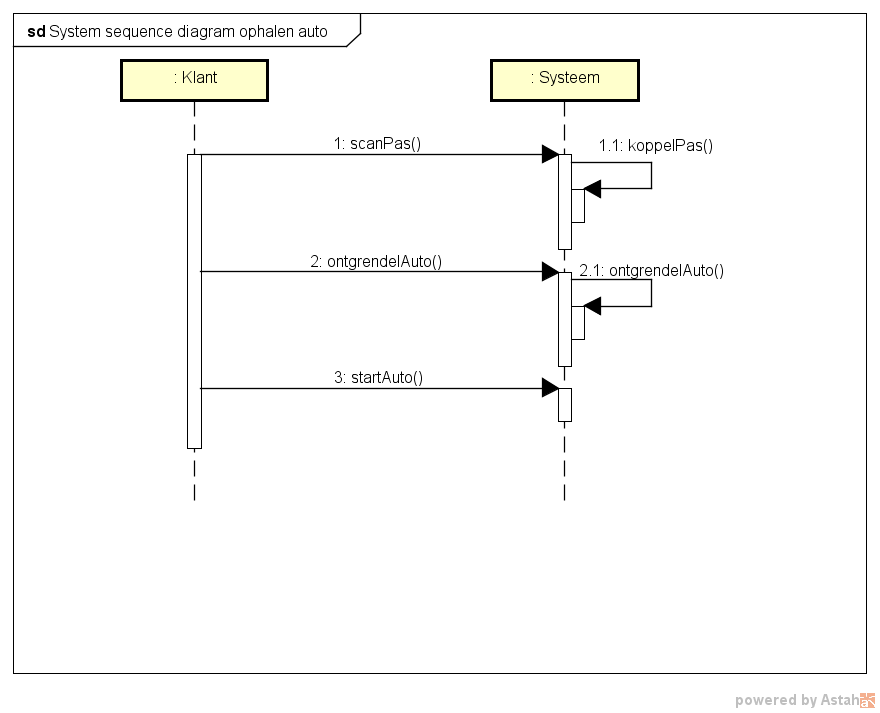


Voor het domein van RedCars is het bovenstaande model gemaakt. Hierin zijn een aantal beslissingen gemaakt die toegelicht worden.  
Elke auto heeft een Gps-module, dit is zodat het systeem bij kan houden waar de auto op alle momenten is. De Gps-module zal ook gebruikt worden voor het vergrendelen en ontgrendelen van de auto. De Gps-module kan niet bestaan als de auto ook niet bestaat. Ook heeft elke auto een parkeerplaats waar hij op moet staan wanneer hij niet in gebruik is. Sommige van deze parkeerplaatsen hebben mogelijkheid om meerdere RedCars auto’s op kwijt te kunnen. Elke parkeerplaats heeft een paal. Dit is zodat wanneer de auto geparkeerd wordt op een parkeerplaats de gebruiker via de paal uit kan checken. De paal kan ook gebruikt worden voor meerdere auto’s tegelijkertijd. Een auto kan alleen wel maar een paal tegelijk hebben. Met de klantenpas kan ingecheckt worden via de paal, dit zorgt ervoor dat de pas gebruikt kan worden om de auto te ontgrendelen. Een pas kan maar bij een paal tegelijkertijd gebruikt worden. Een klantenpas hoort bij een specifieke klant, hierom kan deze ook niet bestaan als de klant niet bestaat. Vandaar de compositite. Een klant is in staat om een reservering te plaatsen, hij kan ook meerdere reserveringen hebben tegelijkertijd. Alleen een reservering kan niet bestaan zonder dat de klant bestaat. Hetzelfde geldt voor een abonnement. Zonder een reservering kan er ook geen betaling bestaan want dan is er niks om te betalen. Verder kan een reservering ook niet bestaan zonder dat er een abonnement is anders kunnen er geen kosten berekend worden. In het model staat ook een website, dit is een onderdeel van het domein omdat de klant via de website reserveringen aan kan maken. Deze website wordt onderhouden door medewerkers.

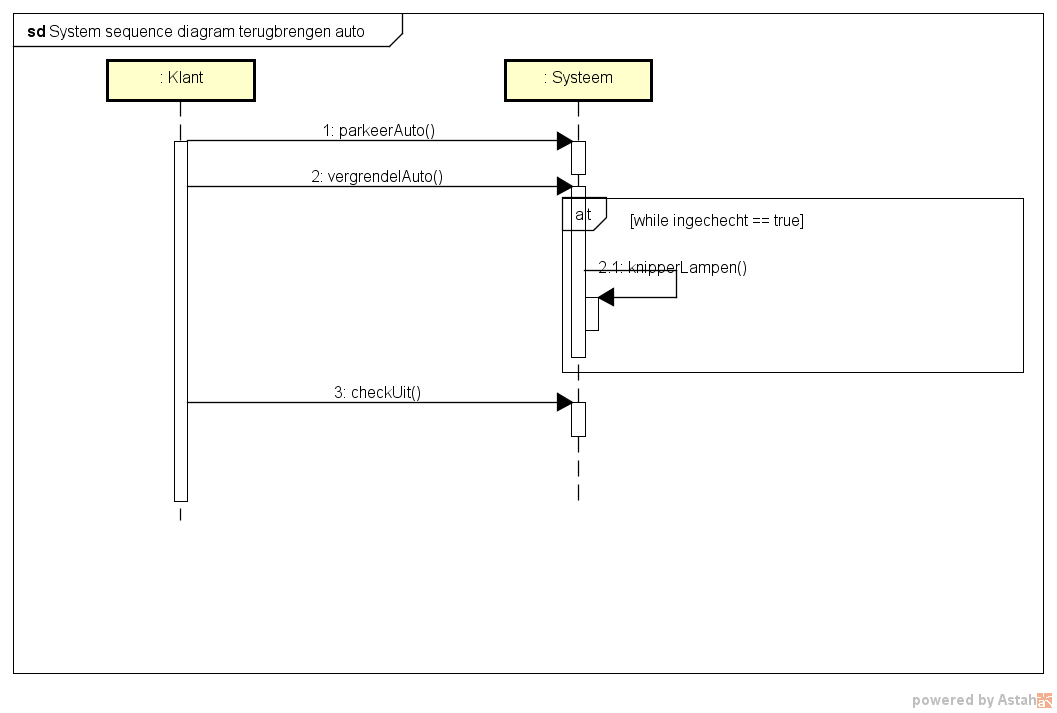
# Sequence diagrammen

Voor Redcars zijn er twee verschillende soorten sequence diagrammen ontworpen, ‘system sequence diagrammen’ en ‘component sequence diagrammen’. Een system sequence diagram is een diagram waarin de events getoond worden die een actor genereert. Hierin wordt het systeem als black box gezien. In een component sequence diagram wordt het systeem niet meer gezien als black box, maar als white box. Dit houdt in dat alle klasses en functies die gebruikt worden om de events die door de actor genereert worden weergeven worden.

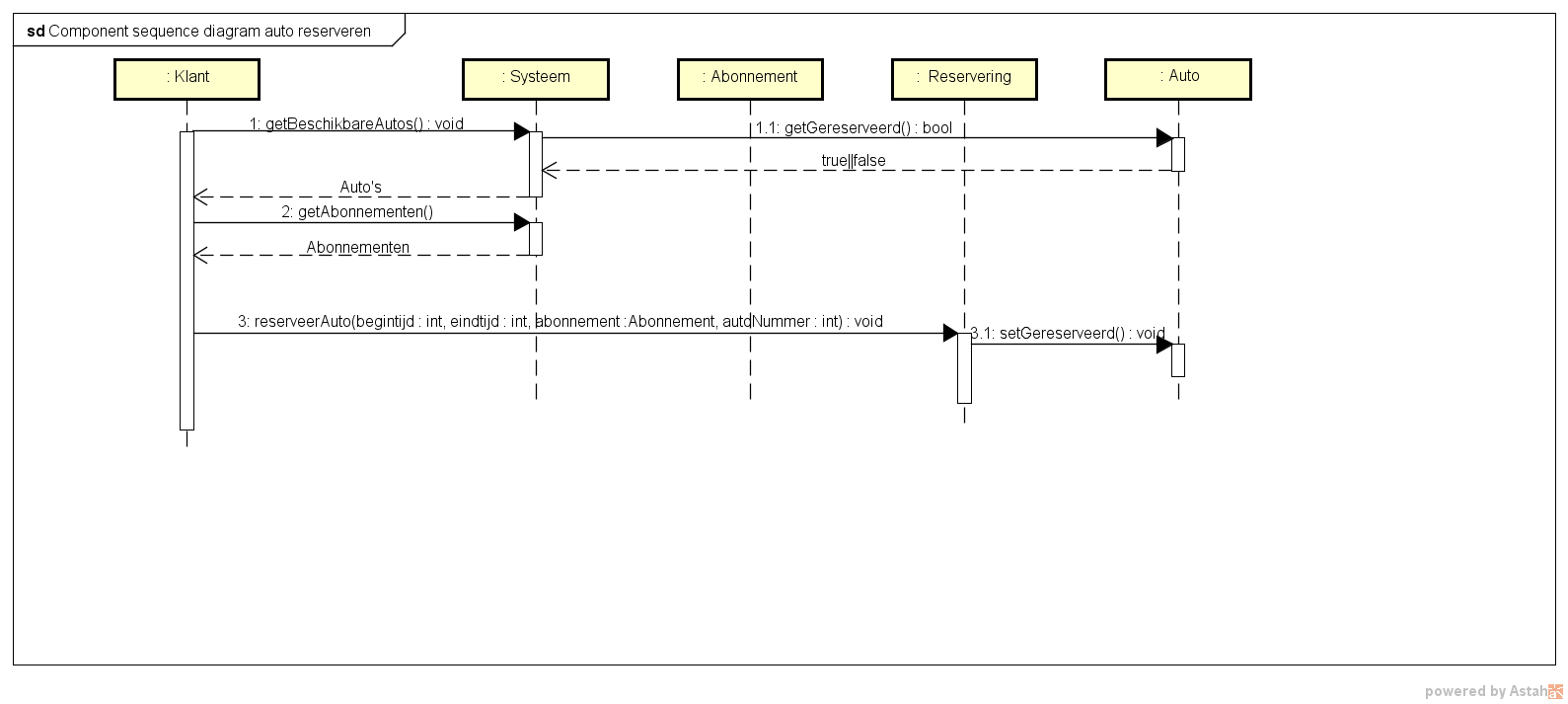
Hierboven staat het system sequence diagram voor het reserveren van een auto. Hier wilt de klant eerst zien welke auto’s er allemaal zijn. Deze worden getoond door het systeem. Vervolgens kiest de klant ervoor om een auto te selecteren, het systeem zal dan de gegevens van de auto tonen. Als de klant de geselecteerde auto wilt reserveren zal hij een formulier in moeten vullen. In dit formulier moet de begintijd, de eindtijd en het abonnement dat de klant wilt ingevuld worden. Wanneer dit allemaal gedaan is zal er een automatische betaling plaats moeten vinden.



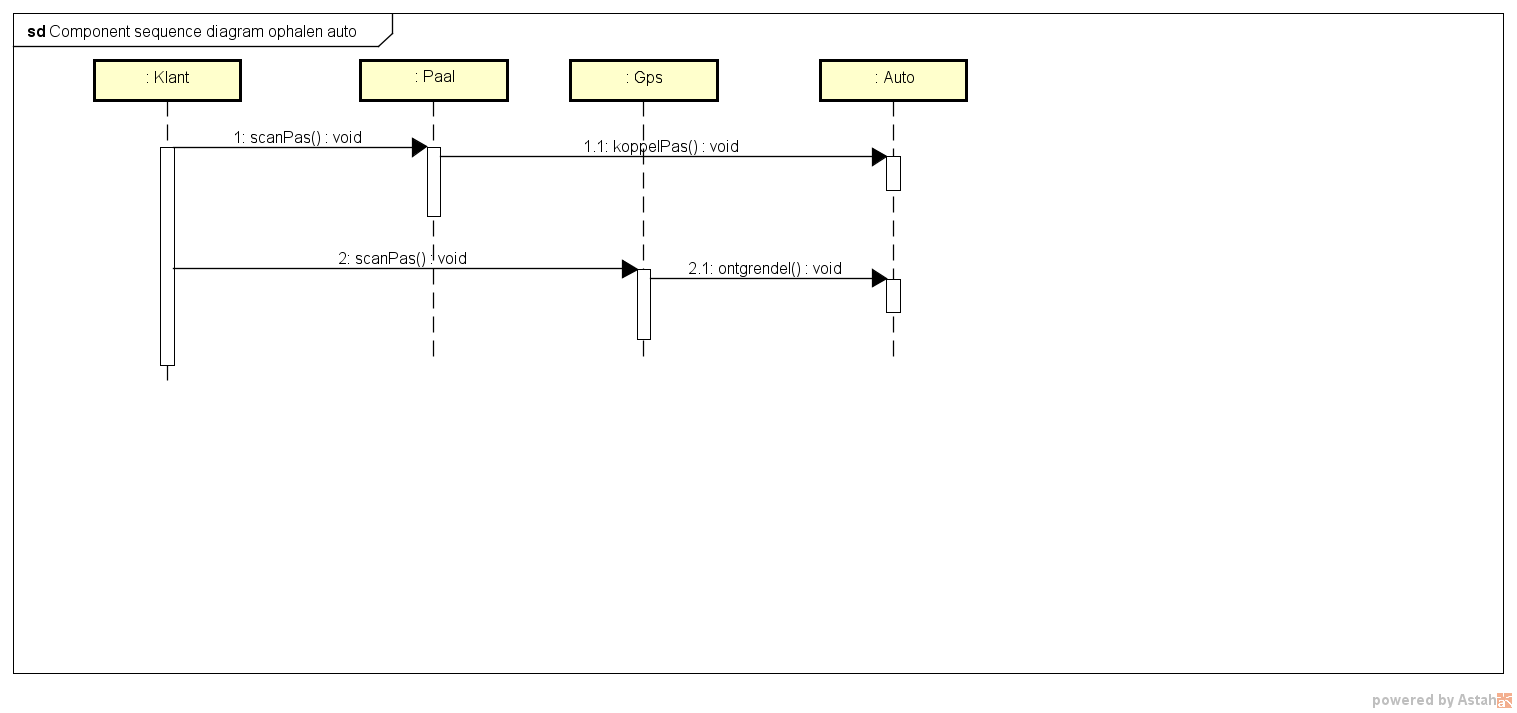
Hierboven staat het system sequence diagram van het ophalen van een auto. Hier wilt de klant een auto ophalen, hiervoor moet hij zijn klantenpas scannen en koppelt het systeem zijn pas aan een auto. Vervolgens kan de klant naar de auto toe gaan en deze opendoen en kan hij er gebruik van maken.



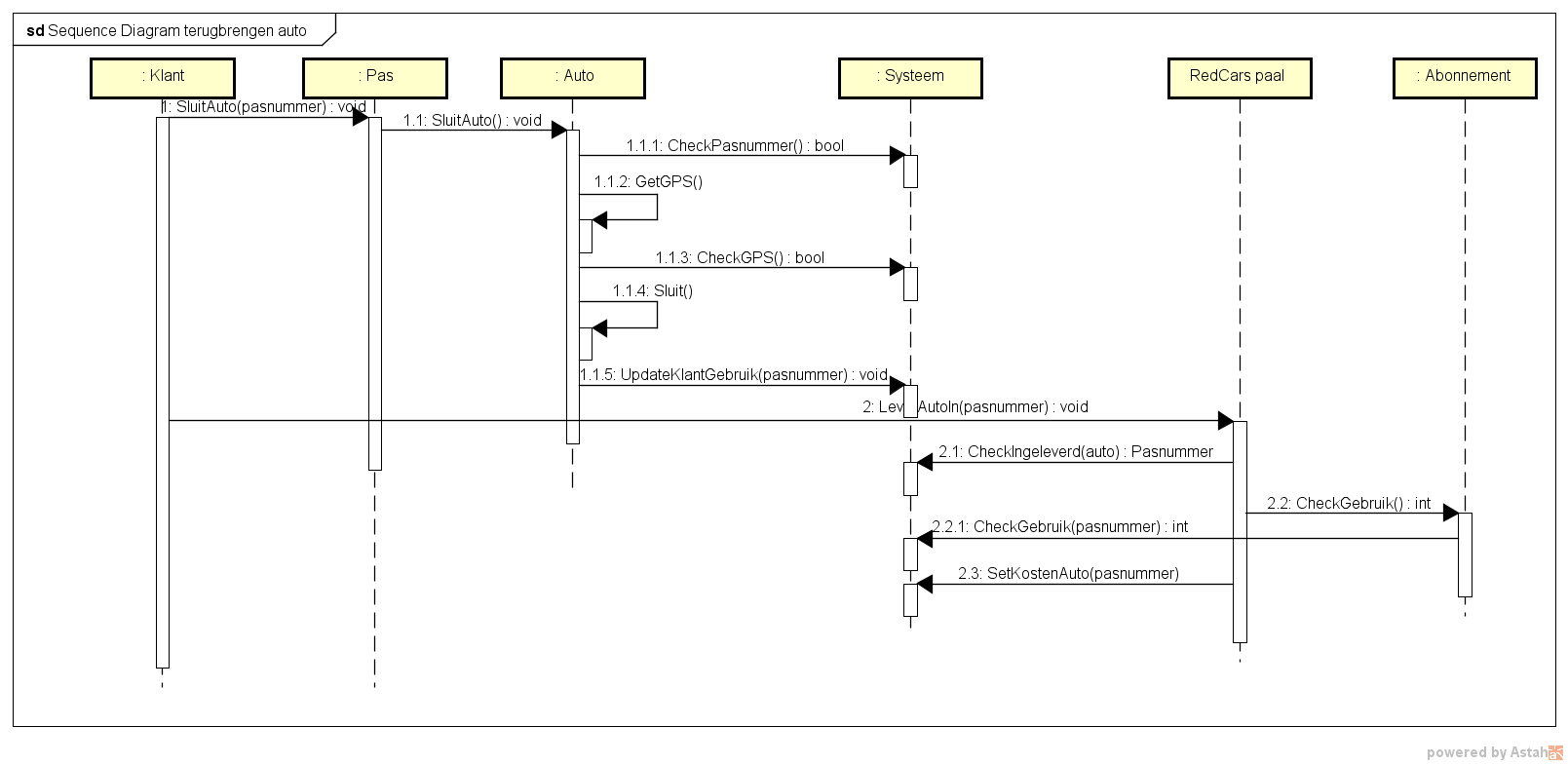
Hierboven staat het system sequence diagram voor het terugbrengen van een auto, om dit te doen zal de auto geparkeerd moeten worden op een RedCars parkeerplaats. Vervolgens moet de auto op slot gedaan worden. Wanneer de klant van de auto wegloopt en nog niet uitgecheckt is zullen er lampen gaan knipperen als de klant nog ingecheckt is zodat hij niet vergeet uit te checken. Vervolgens checkt de klant uit.



Hierboven staat het component sequence diagram voor het reserveren van een auto. Om een auto te reserveren moet de klant eerst weten welke auto’s er allemaal beschikbaar zijn. Hiervoor wordt functie getBeschikbareAutos() van Systeem aangeroepen. Deze returned de autos die niet gereserveerd zijn. Dit doet het systeem door voor elke auto die bestaat op te vragen of deze wel of niet gereserveerd is. Vervolgens moet de klant weten uit welke abonnementen hij allemaal kan kiezen voor om een reservering te plaatsen. Hiervoor vraagt hij aan het systeem op welke abonnementen hij allemaal uit kan kiezen. Als de klant dit allemaal eenmaal weet kan hij een reservering plaatsen. Dit doet hij door de functie reserveerAuto(begintijd, eindtijd, abonnement, autonummer) van Reservering aan te roepen. Wanneer dit succesvol is wordt de auto gereserveerd.

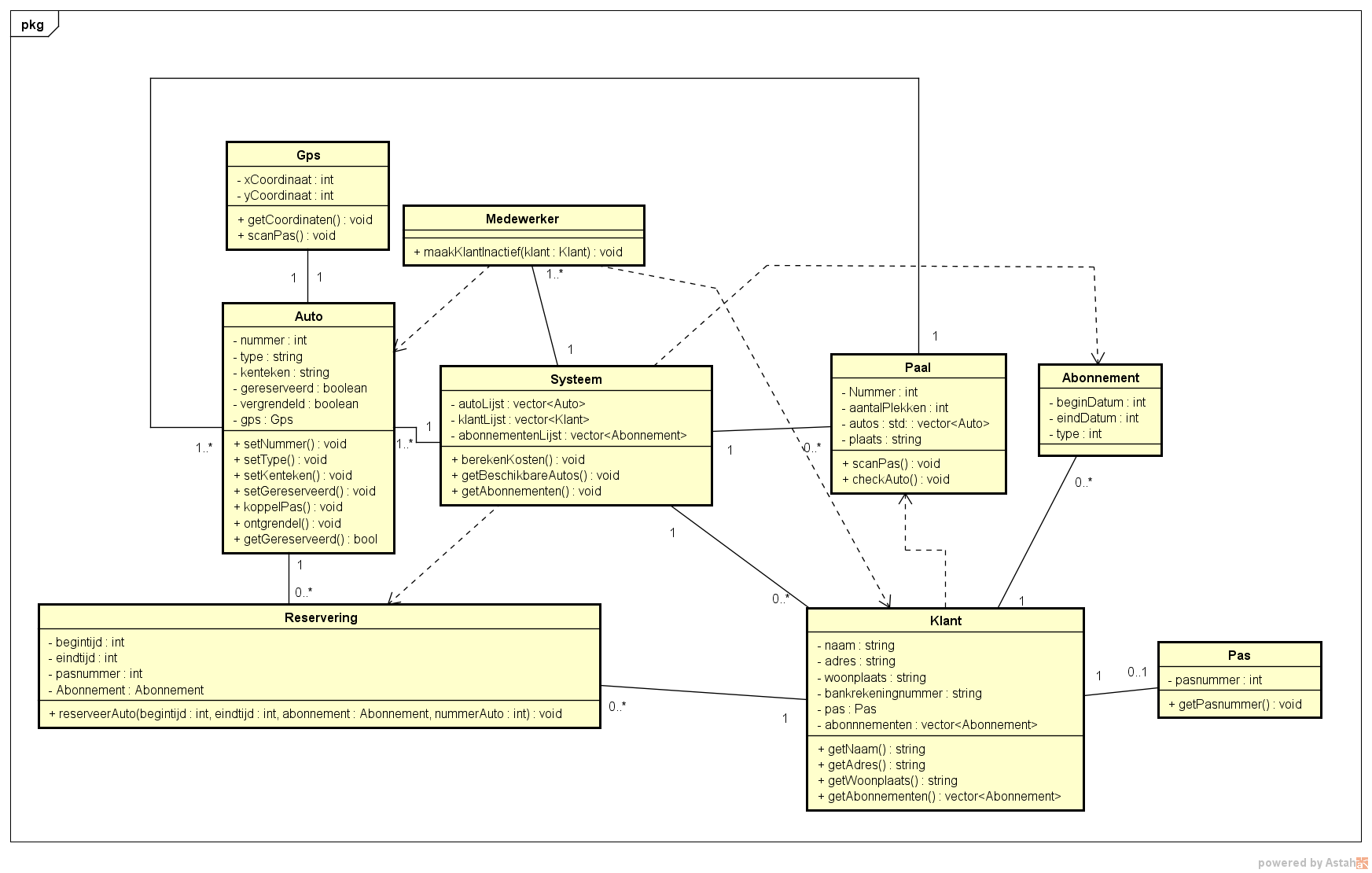


Hierboven staat de component sequence diagram voor het ophalen van een auto. Dit kan de klant doen door wanneer hij een auto gereserveerd heeft zijn pasje tegen de paal aan te houden. Hierdoor wordt de klant ingecheckt en wordt de pas aan de auto gekoppelt. Vervolgens moet de klant zijn pas bij de auto scannen waardoor de auto ontgrendeld wordt en kan de klant instappen en wegrijden.

Hierboven staat het component sequence diagram voor het terugbrengen van een auto. Hiervoor moet de klant de auto afsluiten, dit doet hij door de pas te gebruiken bij de deur van de auto te houden wanneer deze gesloten is. Vervolgens wordt er gecontroleerd of de pas daadwerkelijk aan de auto gekoppeld was. Vervolgens kijkt wordt de locatie van de auto opgehaald, zodat er gecontroleerd kan worden of de auto op een RedCars parkeerplaats. Vervolgens gaat de auto daadwerkelijk op slot. Hierna wordt het klantgebruik geupdatet zodat het systeem weet hoe lang de auto in gebruik is geweest. Vervolgens gaat de klant met de pas naar de paal en houdt de pas tegen de paal. Vervolgens wordt het gebruik gecheckt om te kijken of dit overeenkomt met het afgesloten abonnement. Hierna worden de kosten bepaalt.

# Design class diagram

Een klasse diagram beschrijft de structuur van een object georiënteerd systeem door de klasses in het systeem en de relaties ertussen te tonen. Ook laat het constraints, attributen en methoden zien.



# Applicatie