De ParkeerHeer:
Parkeren in de binnenstad

Wessel Heringa

24 september 2009

Voorwoord

Deze scriptie heb ik geschreven als afsluiting van mijn Master opleiding in Software Engineering aan de Vrije Universiteit van Amsterdam. Het is het resultaat van een acht maanden durende stage bij Logica, waar ik met veel plezier gezeten heb. De opdracht was aanvankelijk om "slimme algoritmen" te bedenken waarmee het parkeren kan worden vergemakkelijkt. Met deze omschrijving kun je alle kanten op en het heeft me veel moeite gekost om dit goed af te bakenen. Bij deze wil ik dan ook Eric ten Bos, Co Kooijman en Huub van Thienen bedanken voor de geboden hulp.

Hoewel de studie Software Engineering niet heel erg veel wiskunde bevat heb ik met deze afstudeeropdracht veel formules moeten uitwerken tot een model. Ik heb altijd affiniteit gehad met wiskundige formules en grafieken en heb dit gedeelte dan ook met veel plezier gemaakt. Sandjai Bhulai: bedankt voor je begeleiding, in het bijzonder voor je hulp bij het opstellen van het wiskundige model. Zonder jouw expertise was me dit nooit gelukt. Daarnaast wil ik Huub van Thienen en Bert Heringa bedanken voor het reviewen van en meedenken met de constructie van dit model.

Naast het wiskundige element heb ik geprobeerd een software architectuur te omschrijven die praktisch toepasbaar is. Ik wil hier Hans van Vliet bedanken voor zijn input op dit onderdeel.

Ik heb ervaren dat het schrijven van een scriptie een stressvolle bezigheid kan zijn. Daarom wil ik als laatste mijn lieve vriendin bedanken, Els Booij. Zonder haar geduld en luisterend oor had ik hier waarschijnlijk veel langer over gedaan.

Wessel Heringa,

Amsterdam, september 2009

Inhoudsopgave

Management Samenvatting

Uit een onderzoek van TNO blijkt dat 10-20% van het stedelijk verkeer op zoek is naar een parkeerplaats. De reeds bestaande *dynamisch parkeerverwijssystemen* trachten dit percentage te verminderen. Deze systemen maken gebruik van informatie van parkeergarages en beelden deze af op borden langs de weg. Op deze manier worden de bestuurders geïnformeerd over de huidige parkeersituatie. Dit onderzoek stelt een nieuw systeem voor, waarbij de informatie niet wordt afgebeeld op verkeersborden langs de weg maar in navigatiesystemen van de bestuurders. Dit systeem noemen we de ParkeerHeer.

De ParkeerHeer beeldt niet alleen informatie over de parkeersituatie af maar dirigeert de bestuurders actief naar de beste parkeergarage. Aangenomen wordt dat de bestuurders weten waar de parkeergarages zich bevinden, ze hebben immers een navigatiesysteem tot hun beschikking. Echter de situatie rond de parkeergarages (vol/niet vol) is ze niet bekend. De ParkeerHeer richt zich daarom op het verminderen van het aantal *weigeringen* van bestuurders bij parkeergarages omdat deze vol zijn.

De ParkeerHeer houdt rekening met alle aangesloten auto's en parkeergarages. Wanneer 50 auto's rijden richting een parkeergarage met 25 vrije plaatsen, zal het grofweg 25 auto's toewijzen en de andere 25 auto's een andere bestemming geven. Hierbij maakt het gebruik van het voorspellend vermogen van de *dynamische parkeerverwijssystemen*: het aantal vrije parkeerplaatsen in de toekomst wordt per parkeergarage geschat, op basis van resultaten in het verleden. Kom je dus over 30 minuten aan bij eindbestemming, dan zal de ParkeerHeer kijken of er *over 30 minuten* in de buurt een parkeerplaats vrij is.

De auto's en parkeergarages zullen op willekeurige momenten de benodigde informatie versturen naar (en opvragen van) de ParkeerHeer. Omdat bij elke toewijzing rekening moet worden gehouden met alle auto's en parkeergarages zal er periodiek een momentopname moeten worden gemaakt van alle auto's en parkeergarages. Met deze momentopname wordt vervolgens een oplossing berekend waarin de auto's worden verdeeld over de parkeergarages. Het gevolg is dat er een zekere vertraging in de oplossing zal komen: Het kan even duren voordat je een parkeerplaats toegewezen krijgt, omdat de momentopname periodiek (bijvoorbeeld elke 15 minuten) wordt gemaakt.

Om een oplossing te kunnen uitrekenen is een wiskundig model ontwikkeld. Het model definieert parkeerplaatsen in plaats van parkeergarages, omdat op deze manier parkeerplaatsen die niet bij een parkeergarage horen kunnen worden meegenomen in de berekening. Het probleem is gereduceerd tot een lineair programmeer probleem. De complexiteit van dit probleem is erg groot waardoor de optimale oplossing niet binnen afzienbare tijd kan worden berekend: worst-case van orde $O(n^m)$, best case van orde $O(m! - n!)$ bij n auto's en m parkeerplaatsen. Daarom wordt er gebruik gemaakt van bestaande software die met slimme algoritmen een sub-optimale oplossing uitrekent in aanzienlijk snellere tijd.

Daarnaast is het model dusdanig ontworpen dat het de functionaliteit van de

ParkeerHeer heeft geïsoleerd in één kostenfunctie c_{pa}: hoe lager c_{pa}, hoe wenselijker het is voor auto a om te parkeren bij parkeerplaats p. Deze kostenfunctie kan naar gelang worden aangepast. Zo kan bijvoorbeeld de afstand tussen eindbestemming en parkeerplaats worden gebruikt om c_{pa} te bepalen, of kunnen bestuurders die een duurder abonnement hebben voorrang krijgen.

Door de Parkeerheer op te delen in vier componenten zijn de verschillende taken geïsoleerd. Op deze manier kunnen verschillende ontwikkelteams werken aan de verschillende subcomponenten. Dit zijn:

- De Core, waarin configuratie data wordt opgeslagen, het systeem wordt opgestart. In de Core staan tevens een aantal hulpfuncties zoals een log,
- De Communicator, verantwoordelijk voor de communicatie van en naar de parkeergarages en auto's,
- De Preparator, die periodiek de opgeslagen data van de communicator gebruikt in combinatie met de kostenfunctie om zo de calculator voor te bereiden,
- De Calculator, die een oplossing uitrekent door gebruik te maken van de eerder besproken bestaande software pakketten

Omdat het streven is om zoveel mogelijk automobilisten aan te sluiten is schaalbaarheid een belangrijke niet-functionele eis. Momenteel is woensdag de drukste dag van de week. Op deze dag zijn 2 miljoen automobilisten op de weg. Het totaal aantal parkeerplaatsen bij parkeergarages ligt op circa 180.000. De ParkeerHeer ondersteunt dit grote aantal gebruikers door de volgende twee punten:

Ten eerste zal de ParkeerHeer met alle type navigatiesystemen overweg kunnen (TomTom, Mio, Garmin, Google Maps, etcetera). Aangenomen wordt dat het ontwikkelen van extra software op navigatiesystemen duurder is. Daarom is ervoor gekozen om zoveel mogelijk business logic te verplaatsen naar de ParkeerHeer. Voor elk type navigatiesysteem en parkeergarage kan er een aparte adapter in de communicator component van de ParkeerHeer geschreven worden die communiceert met desbetreffende client. De adapters kunnen gebruik maken van de API van de Communicator, waarin standaard functies staan zoals het versturen van de locatie van een auto, of het doorgeven van het aantal vrije parkeerplaatsen van de parkeergarage. Door gebruik te maken van deze structuur kunnen later nieuwe systemen worden aangesloten, zoals een website waarin bestuurders hun voorkeur kunnen opgeven, het uitlezen van een RSS feed, of een nieuw type navigatiesysteem.

Ten tweede is de ParkeerHeer opgedeeld in verschillende processen, die kunnen worden verdeeld over verschillende computers. Op deze manier kan bij een groot aantal auto's en parkeergarages de totaal benodigde rekenkracht worden verdeeld over verschillende processoren wat de performance ten goede komt. Elke adapter draait als eigen proces, er is één proces voor de calculator en preparator voor het periodiek uitrekenen van een oplossing, en er is een speciaal *loadbalancer* proces die de vele communicatieverzoeken verdeelt over de verschillende adapters.

1 Inleiding

In 2007 bracht TNO een onderzoek uit naar de voordelige effecten van ICT toepassingen in het verkeer [2]. Uit dit onderzoek kwam naar voren dat door het inzetten van ICT in het Nederlandse verkeer veel winst valt te behalen in de komende 10 tot 15 jaar: 50% minder files, 25% minder verkeersdoden, 10% minder uitstoot van CO_2 en 20% minder luchtvervuiling. Een deel van het TNO onderzoek behandelt de verloren reistijd door het niet kunnen vinden van een geschikte parkeerplaats: Het blijkt dat 10%-20% van het stedelijk verkeer in Nederland op zoek is naar een parkeerplek.

Een *dynamisch parkeerverwijssysteem* is een systeem waarmee autobestuurders worden geïnformeerd over de actuele situatie omtrent de beschikbare plaatsen van parkeergarages nabij hun huidige positie. De bestuurder kan hiermee nagaan waar er geparkeerd kan worden en of daar nog vrije parkeerplekken zijn. Een voorbeeld van een dynamisch parkeerverwijssysteem zijn de blauwe borden in de binnensteden, die aangeven waar geparkeerd kan worden. Soms worden deze ondersteund door een dynamische weergave van het aantal beschikbare parkeerplaatsen per parkeergarage. Met behulp van dynamische parkeerverwijssystemen kunnen de reistijdverliezen door verkeer op zoek naar parkeerplaatsen met 30% tot 50% afnemen [2].

Figuur 1: *Een voorbeeld van een dynamisch parkeerverwijssysteem. P Centrum Noord heeft nog 220 vrije parkeerplaatsen.*

In dit onderzoek wordt een variant van een dynamisch parkeerverwijssyteem ontwikkeld. Het doel is om de informatie die wordt weergegeven op de borden te interpreteren en door te sturen naar navigatiesystemen. Een belangrijk verschil met de systemen waarover in het TNO onderzoek wordt gesproken, is dat dit systeem de auto's naar de meest geschikte parkeerplaats dirigeert in plaats van slechts te informeren over het aantal vrije plaatsen bij de parkeergarages: Het maakt dus een beslissing voor de bestuurder. We noemen dit systeem de **ParkeerHeer**. Figuur 2 geeft een voorbeeld van het gebruik van de ParkeerHeer op een navigatiesysteem.

Figuur 2: Screenshot van de ParkeerTomTom: Een uitbreiding op het TomTom navigatiesysteem dat gebruik maakt van de diensten van de ParkeerHeer: De auto wordt naar de parkeergarage in de linkerbovenhoek gestuurd.

Het volgende hoofdstuk gaat dieper in op de probleemstelling. De structuur van dit document wordt uitgelegd in Hoofdstuk 3.

2 Doelstelling

Het voornaamste doel van de ParkeerHeer is het verkorten van de reistijd van bestuurders. Zoals eerder aangegeven is een groot deel van het stedelijk verkeer op zoek naar een parkeerplaats [2]. Dit verkeer noemen we vanaf nu het *zoekend verkeer*. De tijd dat één auto zoekt naar een parkeerplaats noemen we de *zoektijd*. De primaire onderzoeksvraag luidt:

Hoe kan de gemiddelde zoektijd van het zoekend verkeer worden verminderd?

We definiëren drie factoren die deze zoektijd beïnvloeden. Alle andere factoren kunnen in één van deze drie hoofdfactoren worden ondergebracht:

De bekendheid van de bestuurder met de stad

Iemand die weet welke route ze[1] het best kan nemen naar een parkeerplaats zal er eerder aankomen dan een bestuurder die nog nooit in de betreffende stad is geweest. Er zijn verschillende manieren om de bekendheid van de stad te vergroten. Enkele voorbeelden zijn de bewegwijzering en de navigatiesystemen in auto's.

Het aantal keer dat een bestuurder een bezette parkeerplaats aantreft

Idealiter rijdt een bestuurder in één keer naar een vrije parkeerplaats. Elke keer dat ze een bezette parkeerplaats aantreft zal ze haar weg moeten vervolgen. Dit betekent dat ze een omweg heeft gemaakt en dus niet meer de meest ideale route heeft genomen. Het aantreffen van een bezette parkeerplaats noemen we een *weigering*. Het aantal weigeringen wordt onder andere beïnvloed door het aantal vrije parkeerplaatsen in de buurt van de eindbestemming: Hoe minder vrije parkeerplaatsen er zijn, hoe groter de kans is dat de bestuurder niet in één keer een geschikte plaats kan vinden. Een andere factor is de kennis die de bestuurder heeft over de bezetting van parkeerplaatsen. Wanneer de bestuurder weet welke parkeerplaats vrij is, kan ze daar meteen heen rijden.

De eisen die de bestuurder stelt aan een parkeerplaats

Mensen willen niet vijf kilometer van parkeerplaats naar eindbestemming lopen. Ook willen ze niet van het kastje naar de muur gestuurd worden. De eisen die de bestuurder stelt aan een parkeerplaats beïnvloedt het aantal geschikte parkeerplaatsen. Hoe strenger de eisen, hoe minder geschikte parkeerplaatsen er zijn en hoe langer de bestuurder gemiddeld zal moeten zoeken naar een parkeerplaats. Daarom geldt: hoe strenger de eisen van de bestuurder, hoe langer de gemiddelde zoektijd.

2.1 Opdrachtomschrijving

In dit onderzoek wordt de onderzoeksvraag beantwoord door een applicatie te ontwikkelen die de gemiddelde zoektijd van auto's vermindert door één of meer van de hierboven genoemde factoren te beïnvloeden. Deze applicatie noemen

[1] In deze scriptie wordt voor onbepaalde personen de vrouwelijke vorm gebruikt. Dus 'ze' en 'zij' en niet 'hem' en 'hij'.

we de **ParkeerHeer**. De volgende doelstellingen zullen dienen als uitgangspunt bij de ontwikkeling van deze applicatie:

- **Verkorten van de zoektijd**: De bestuurders die gebruik maken van de diensten van de ParkeerHeer moeten gemiddeld een kortere zoektijd hebben dan bestuurders die er geen gebruik van maken.
- **Maximaliseren van het aantal aangesloten bestuurders**: De groep bestuurders die gebruik maakt van de diensten van de ParkeerHeer moet zo groot mogelijk zijn, om de totale gemiddelde zoektijd zo veel mogelijk te minimaliseren.
- **Maximaliseren van het aantal aangesloten parkeerplaatsen**: Het aantal parkeergarages dat gebruik maakt van de diensten van de ParkeerHeer moet zo groot mogelijk zijn om de keuze van de gebruikers zo groot mogelijk te maken en daardoor ook de totale gemiddelde zoektijd te minimaliseren.
- **Mogelijkheid tot uitbreiding van functionaliteit**: De mogelijkheden van een applicatie die auto's naar parkeerplaatsen dirigeert zijn zo groot dat ze niet in één keer gerealiseerd kunnen worden. Daarom moet de ParkeerHeer zo ingericht worden dat er gemakkelijk aanpassingen kunnen worden gedaan. Er kan gedacht worden aan het ondersteunen van gehandicaptenparkeerplaatsen, het zoeken naar restaurants in plaats van parkeerplaatsen, een ad-hoc parkeergarage bij een bijzonder evenement zoals pinkpop, en nog veel meer.

Deze doelstellingen staan aan de basis van de ontwikkeling van de ParkeerHeer. In de komende hoofdstukken zullen ze in meer detail worden uitgewerkt.

2.2 Belanghebbenden

Naast het maatschappelijk belang dat het verminderen van het zoekend verkeer heeft is er een aantal partijen die baat heeft bij dit onderzoek. Ten eerste wordt het onderzoek uitgevoerd in opdracht van Logica, een IT-detacheringsbedrijf dat zich voornamelijk bezighoudt met innovatieve projecten. Wanneer uit dit onderzoek blijkt dat de doelen van ParkeerHeer haalbaar zijn, zal de applicatie mogelijk worden doorontwikkeld en in gebruik worden genomen. Ten tweede dient dit document als Master Thesis voor de studie Software Engineering aan de Vrije Universiteit van Amsterdam.

3 Aanpak

Het probleem van het parkeren in de binnenstad kan op zoveel verschillende manieren worden opgelost dat het makkelijk is om het overzicht te verliezen. In dit hoofdstuk leggen we uit hoe we geprobeerd hebben zoveel mogelijk structuur aan te brengen in dit onderzoek.

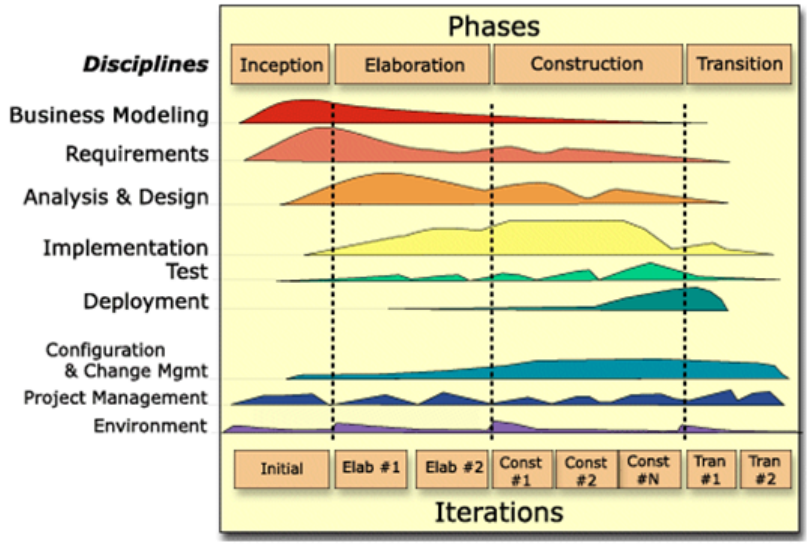
Het doel van dit document is om een *software architectuur* te presenteren waarmee het parkeren in de binnenstad kan worden vergemakkelijkt. We gebruiken hier de term architectuur omdat we alleen de meest belangrijke eisen en genomen ontwerpbeslissingen uitwerken. Dit resulteert in een bepaalde indeling van componenten van het systeem en een globaal overzicht van de manier waarop deze componenten met elkaar communiceren[2]. Wanneer we in dit document het woord *Design* of *Ontwerp* gebruiken bedoelen we hiermee het ontwerp van de software architectuur. De eisen worden in dit document ook wel aangeduid als *Requirements*.

3.1 Ontwikkelproces

In dit onderzoek hebben we een ontwikkelmethode gebruikt die is gebaseerd op het Rational Unified Process (RUP) [8]. In RUP wordt onderscheid gemaakt tussen *disciplines* en *ontwikkelfasen*. Een aantal voorbeelden van disciplines in de Sofware Engineering zijn *Requirements Engineering* en *Analysis and Design*. De RUP methode breekt met een oude manier van ontwikkelen die ook wel het *waterval* model wordt genoemd. In het waterval model wordt in elke ontwikkelfase één discipline helemaal uitgewerkt: zo worden eerst alle eisen (requirements) vergaard. Wanneer deze zijn uitgewerkt wordt begonnen aan het ontwerp.

Echter, de verschillende disciplines zijn sterk van elkaar afhankelijk. Zo kan een aantal eisen (Requirements discipline) pas worden gedefinieerd nadat er een aantal fundamentele ontwerpbeslissingen is gemaakt (Analysis & Design discipline). Dit is één van de redenen dat met RUP in elke ontwikkelfase aandacht wordt gegeven aan elke discipline. Dit wordt ook wel geïllustreerd in afbeelding 3 waar de ontwikkelfases horizontaal staan en de disciplines verticaal. Zo wordt bijvoorbeeld in de eerste fase (Inception) de meeste aandacht besteed aan de disciplines *Business Modeling* en *Requirements*, maar ook een beetje aan *Analysis & Design*. In de inceptie wordt dus het grootste deel van de requirements gemaakt, maar er is ook ruimte om tot een eerste globale versie van het ontwerp te komen. In dit document behandelen we twee iteraties: de *Inceptie* (Inception) en *Elaboratie* (Elaboration) fase.

[2]De term software architectuur komt van Bass et al. [9]. In dit boek wordt software architectuur omschreven als *"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them"*.



Figuur 3: De verschillende disciplines en iteraties van het Rational Unified Process: In elke iteratie wordt een bepaalde hoeveelheid aandacht besteed aan elke discipline. Uit: Philippe Kruchten - What is Rup [8]

Wanneer er iteratief wordt ontwikkeld zoals in RUP, is het gebruikelijk om voor elke discipline een apart document te maken. Enkele voorbeelden zijn een *Software Requirements Specification* waar de eisen in staan of een *System Design Document* waarin het ontwerp wordt uitgewerkt. In bijna elk document wordt verwezen naar andere documenten: De ene eis leidt tot het opdelen van een bepaald systeem in meerdere sub-systemen, wat weer leidt tot meer eisen of een bepaalde ontwerpbeslissing, die op haar beurt weer leidt tot nieuwe eisen. De lineariteit ontbreekt daarom in deze documenten.

Dit document streeft echter wel naar een zekere lineariteit: het zal van de eerste tot de laatste pagina gelezen worden. We delen het daarom op in twee ontwikkelfasen van RUP. Dit betekent ook dat de verschillende disciplines in elk deel zullen terugkomen en nooit helemaal volledig zullen zijn. Zo eindigt het hoofdstuk *Vereisten* (Hoofdstuk 5) niet met een volledige lijst van eisen, maar met een onvolledige lijst aangevuld met een overzicht van de onderdelen die nog moeten worden uitgewerkt.

In de komende paragrafen van dit hoofdstuk leggen we eerst uit welke disciplines we gebruiken en wat ze inhouden. Vervolgens geven we een overzicht van de verschillende ontwikkelfasen.

3.2 Disciplines

In dit onderzoek wordt onderscheid gemaakt tussen een aantal disciplines dat in dit gehele document zal terugkeren. De belangrijkste zijn:

Business Modeling Discipline

Deze discipline verkent de mogelijkheden: Is dit project binnen het beschikbare budget en tijd te realiseren? Kan er een technische oplossing voor worden gevonden?

Requirements Discipline

In deze discipline wordt bepaald *wat* de ParkeerHeer gaat doen. Dit wordt vertaald in eisen. Een requirement is hetzelfde als een eis. Een eis beschrijft een specifieke eigenschap die het systeem moet hebben. Een voorbeeld van een eis is *De auto's moeten binnen 15 seconden antwoord krijgen van de ParkeerHeer.* Een eis wordt geïdentificeerd door de letter R , gevolgd door een uniek nummer. Een eis heeft daarom de volgende opmaak:

R<nr van de eis>: <Korte omschrijving>

<Uitleg van de eis, en argumentatie>

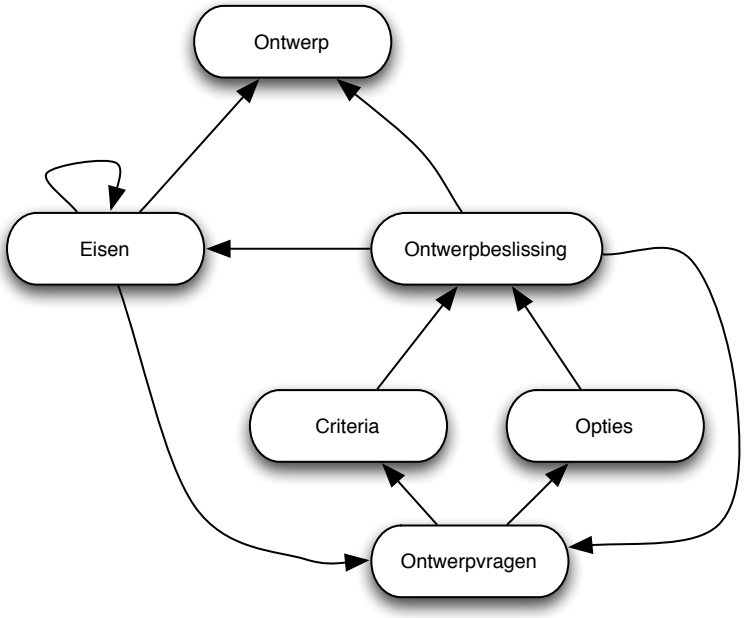
Analysis & Design Discipline

De Analysis & Design discipline geeft een antwoord op de vraag, hoe het systeem gerealiseerd gaat worden. In het Nederlands wordt voor *design* ook wel het woord *ontwerp* gebruikt. Het ontwerp van de ParkeerHeer bestaat uit een opdeling van het systeem in verschillende subcomponenten, een omschrijving van deze componenten en de manier waarop deze met elkaar samenwerken.

3.3 Relatie tussen de disciplines

In een aantal gevallen is een genomen ontwerpbeslissing niet vanzelfsprekend. In dit geval wordt deze met behulp van de QOC methode uitgewerkt [1]. QOC staat voor **Ontwerpvragen** (**Q**uestions), **Opties** (**O**ptions) en **Criteria** (**C**riteria): Een onduidelijk of onvolledig onderdeel van het systeem wordt benoemd door hier een bepaalde ontwerpvraag over te stellen. Vervolgens worden de mogelijke opties en afwegingen benoemd door de verschillende alternatieven (opties) te behandelen. Dit wordt ook wel de *ontwerpruimte* genoemd. Elke ontwerpvraag heeft hierbij een eigen ontwerpruimte. Uiteindelijk resulteert dit in een ontwerpbeslissing, wat weer leidt tot nieuwe eisen en/of ontwerpvragen.

Afbeelding 4 geeft een grafische weergave van de QOC methode en de relatie tussen de verschillende onderdelen. Hieronder verduidelijken we deze methode door elk onderdeel ervan apart te behandelen:



Figuur 4: De relatie tussen de in deze scriptie gebruikte termen. Pijlen geven aan dat het ene leidt tot het ander: Een nieuwe eis resulteert dus in nieuwe ontwerpvragen en/of een nieuw deel in het ontwerp.

Ontwerpvragen

Met een ontwerpvraag wordt een probleem aan de kaak gesteld. Het doel is om een oplossing te vinden voor dit probleem. Een voorbeeld van een vraag is: *Welke factoren moeten worden beïnvloed om de zoektijd te verkorten?* Daarnaast wordt de reden gegeven dat de vraag gesteld wordt. Een vraag kan resulteren uit een eerder genomen ontwerpbeslissing, uit een eis, of uit een van de uitgangspunten (paragraaf 2.1).

Om het overzicht te kunnen houden nummeren we de ontwerpvragen. Een ontwerpvraag identificeren we met de letter Q, gevolgd door het unieke nummer. Op de plaats waar de ontwerpvraag het eerst wordt gepresenteerd zetten we er tussen haakjes bij waar deze vraag wordt beantwoord. Dit kan een paragraaf of hoofdstuk zijn, maar ook een use-case. Een ontwerpvraag heeft daarom de volgende opmaak:

Q<nummer>: <De betreffende ontwerpvraag>(<Plaats in dit document waar deze vraag wordt beantwoord>)

Criteria

Met de criteria wordt aangegeven welke onderwerpen belangrijk zijn als het gaat om het maken van een ontwerpbeslissing. Een voorbeeld van een criterium is *In hoeverre is deze keuze van invloed op de performance van het systeem.* Veel gebruikte criteria zijn:

- *haalbaarheid* - In hoeverre is het haalbaar om een optie te realiseren
- *ontwikkelsnelheid* - in hoeverre kan een optie snel ontwikkeld worden
- *performance* - In hoeverre is een optie van invloed op de prestatie van de ParkeerHeer

Opties

Voor elk alternatief wordt uitgelegd wat het precies inhoudt en op welke manier er wordt voldaan aan de criteria. Voor elke optie geldt, dat het meer voldoet aan bepaalde criteria en minder aan andere. In elke optie zit dus een bepaalde afweging, waarbij er voor bepaalde criteria concessies gedaan worden ten gunste van andere(n).

Ontwerpbeslissing

Een ontwerpbeslissing is het maken van een keuze uit één van de voorgestelde alternatieven (opties). In dit onderdeel wordt de keuze beargumenteerd. Een ontwerpbeslissing kan resulteren in een nieuwe set van eisen, ontwerpvragen, en/of een verdieping in het ontwerp.

Door een keuze te maken stellen we dat bepaalde criteria belangrijker zijn dan andere. Sommige criteria kunnen we door de gemaakte keuze verder specificeren wat resulteert in eisen. Het komt ook voor dat er nog niet genoeg bekend is over een bepaald criterium om er een concrete eis van te maken. In dit geval wordt er een opsomming gemaakt van ontwerpvragen die beantwoord zullen worden voordat deze eis kan worden uitgewerkt. Er wordt dan teruggekomen op deze eis als deze vragen zijn beantwoord.

3.4 Ontwikkelfasen

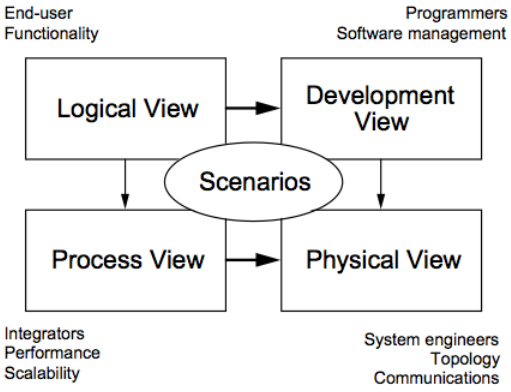
Zoals eerder aangegeven is er een verschil in de opbouw van dit document en het proces waarin de ParkeerHeer is ontwikkeld. Dit document is lineair van opzet, dat wil zeggen dat de lezer het van voor naar achter kan lezen. Bij de ontwikkeling is echter iteratief gewerkt, wat betekent dat zowel de eisen als het ontwerp in de loop van het ontwikkelproces zijn meegegroeid. De rest van dit document is opgedeeld in drie delen: Inceptie (in RUP: Inception), Elaboratie (in RUP: Elaboration), en Discussie/Conclusie. We bespreken deze kort in de komende paragrafen:

3.4.1 Inceptie (Inception)

In dit deel plegen we een vooronderzoek waarin we de bestaande systemen bekijken (business modeling discipline). Vervolgens gebruiken we deze kennis om tot een eerste set van ontwerpbeslissingen te komen, die resulteren in een reeks eisen en een globaal ontwerp. Niet alle eisen zullen zijn uitgewerkt. De niet uitgewerkte eisen zullen zijn voorzien van begeleidende ontwerpvragen die grotendeels worden beantwoord in de elaboratie. Aan het eind van dit deel is globaal bekend *wat* er precies gemaakt moet worden, en aan welke voorwaarden dit moet voldoen.

3.4.2 Elaboratie (Elaboration)

In dit deel worden de eisen uit het voorgaande deel uitgewerkt en wordt er een software architectuur voorgesteld die deze eisen dekt. Aan het eind van dit deel is globaal bekend *hoe* het systeem wordt gerealiseerd. Als leidraad nemen we het *4+1 Architectural Model* van Philippe Kruchten. Philippe Kruchten publiceerde omstreeks 1995 een methode waarin vier perspectieven, oftewel *Viewpoints* werden beschreven, en een vijfde met als doel de overige vier te testen [6]. Elk perspectief belicht de architectuur vanuit een andere invalshoek. Figuur 5 is een grafische representatie van het *4+1 Architectural Model*



Figuur 5: 4+1 Architectural Model, uit Kruchten [6]

Er zijn twee redenen waarom er is gekozen voor dit model. Ten eerste is het een veel gebruikte, bewezen methode. Ten tweede wordt het veel gebruikt bij opdrachtgever Logica. Wanneer de ParkeerHeer wordt doorontwikkeld door software engineers bij dit bedrijf zullen ze sneller de architectuur begrijpen en kunnen aanvullen. Hier moet een kleine kanttekening geplaatst worden: wij gebruiken het 4+1 model van Philippe Kruchten uit 1995 [6]. In 2003 is dit model in aangepaste vorm opgenomen in RUP [7]. Om verwarring bij ontwikkelaars te voorkomen beschrijven we nu kort de verschillende architectural viewpoints.

Logical Architecture View

Met de Logical Architecture wordt er een weergave gemaakt van de ParkeerHeer die laat zien hoe aan de functionele eisen wordt voldaan. Er wordt naar de ParkeerHeer gekeken vanuit het eindgebruikers perspectief: hoe gaat de functionaliteit verwezenlijkt worden? Hierbij wordt rekening gehouden met de eerder gestelde niet-functionele eisen (onder andere uitbreidbaarheid, performance, en veiligheid). Er wordt onder meer een wiskundig model gepresenteerd waarmee de auto's naar parkeerplaatsen gestuurd kunnen worden.

Process Architecture View

In de Process Architecture werpen we een blik op de ParkeerHeer vanuit de gelijktijdig uitgevoerde processen: Welke operaties moeten synchroon kunnen

worden uitgevoerd? En hoe zorgen we dat bij het uitvoeren van veel gelijktijdige operaties de performance eis gewaarborgd blijft?

Development Architecture View

De Development Architecture concentreert zich op de software en de pakketten waarin de software is ingedeeld. De software ontwikkelaar staat hier centraal. Het doel is om aan te geven op welke delen de ParkeerHeer kan worden uitgebreid: hoe verschillende typen navigatiesystemen kunnen worden aangesloten, en hoe de ParkeerHeer kan worden uitgebreid met nieuwe functionaliteit.

Physical Architecture View

Met de Physical Architecture wordt de ParkeerHeer gepresenteerd op hardware niveau: Welke computers en hardware systemen zijn er nodig voor de ParkeerHeer? Als we dit weten kunnen we kijken op welke manier de ParkeerHeer geschaald kan worden. Dit doen we samen met de kennis over gelijktijdig uitgevoerde processen die we in de Process Architecture View hebben verkregen. De belangen van de systeem ontwikkelaar, die gaat over schaalvergroting en de hardware waar de ParkeerHeer op draait, staan hier centraal.

Scenario's (Use Cases)

In figuur 5 is dit de middelste cirkel. De scenario's, ofwel use cases, overlappen de vier andere perspectieven. Ze geven een beeld van het systeem zoals het zou moeten werken. Bij de bespreking van de vier andere views kan worden gerefereerd naar de scenario's.

Het scenario hoofdstuk bevat alle use cases. De scenario view kan dus gezien worden als een gevolg van de voorgaande hoofdstukken, maar ook als een begin. Als eerste zijn de scenario's ontworpen en deze evolueerden mee met de architectuur.

Een use case wordt geïdentificeerd met de letters UC, gevolgd door een uniek nummer.

3.4.3 Discussie / Conclusie

Dit deel is een terugblik op de vorige delen. Er wordt gekeken in hoeverre het ontwerp voldoet aan de eisen en welke sterke en zwakke punten het heeft.

Deel I
Inceptie

Hoofdstuk 4: Vooronderzoek

Hoofdstuk 5: Vereisten

4 Vooronderzoek

In dit hoofdstuk wordt een aantal bestaande systemen die auto's de weg wijzen naar parkeergarages onderzocht. Daarnaast kijken we naar het totaal aantal bestuurders en parkeerplaatsen in Nederland.

4.1 Parkeersystemen

We behandelen verschillende parkeersystemen: De parkeerrouteinformatiesystemen en navigatiesystemen. Als derde kijken we naar de eigenschappen van de ParkeerTomTom, een systeem dat deze twee met elkaar combineert.

4.1.1 Parkeerrouteinformatiesysteem

Een veel gebruikt dynamisch parkeersysteem is het *Parkeerrouteinformatiesysteem*, ofwel *Parkeergeledeidingssysteem* en wordt vaak afgekort als PRIS [2]. PRIS zijn reeds in veel steden geïmplementeerd met behulp van informatieborden, waarop staat vermeld hoeveel vrije parkeerplaatsen een parkeergarage heeft. Het bord kan ook alleen een vol/vrij mededeling weergeven. De bestuurder krijgt dus voortijdig informatie aangeleverd waarmee ze een betere keuze kan maken uit de bekende parkeerplaatsen. Figuur 1 is een voorbeeld van een PRIS informatie parkeerbord. De informatieborden hebben onder andere de volgende eigenschappen [2]:

1. **Verschaffen informatie**: De informatieborden geven informatie over het aantal vrije parkeerplaatsen. De bestuurder moet aan de hand van deze informatie zelf beslissen waar ze naartoe rijdt.

2. **Voorspellend vermogen**: Het aantal vrije parkeerplaatsen dat op het bord wordt afgebeeld is een schatting in de toekomst. Een bord dat 15 minuten rijden van de parkeergarage ligt, beeldt informatie af van de parkeergarage, 15 minuten in de toekomst. Op deze manier wordt de kans dat de bestuurder geweigerd wordt bij aankomst kleiner en wordt de zoektijd verkort.

3. **Elektronische verbinding tussen parkeergarage en informatiebord**: De informatieborden geven informatie over de parkeergarages waarmee ze in verbinding staan. De parkeergarages maken gebruik van een systeem dat het aantal vrije parkeerplekken bijhoudt en daarmee het voorspellend vermogen realiseert.

4. **Gebruik van vooraf gedefinieerde routes**: De borden worden tevens gebruikt als wegwijsborden naar de parkeergarage. Zo'n vooraf gedefinieerde route wordt ook wel een P-route genoemd. Een P-route is een route in een stad waarbij meerdere parkeergarages worden aangedaan. Is de parkeergarage vol, dan kan de bestuurder de P-route verder volgen naar de volgende parkeergarage.

5. **Per gemeente geïmplementeerd**: De Parkeerrouteinformatiesystemen worden veelal door gemeenten geïmplementeerd en opereren niet op nationaal niveau: Elke gemeente of parkeergarage heeft haar eigen centrale punt waar het de parkeergegevens opslaat.

Met deze eigenschappen pogen de informatieborden de bekendheid van de bestuurder te vergroten (de eerste factor), en het aantal weigeringen te verkleinen (tweede factor). De p-routes zorgen ervoor dat de bestuurder na een weigering weet naar welke alternatieve parkeergarages ze kan rijden.

Een nadeel van dit systeem is dat het mensen informeert op basis van toekomstvoorspellingen. Echter wanneer de gebruikers reageren zoals bedoeld zal de toekomstvoorspelling niet meer correct zijn. Een voorbeeld: Het is druk bij een parkeergarage. De voorspelling is dat deze binnen een half uur vol zal zijn, met de huidige instroom. Informatieborden op meer dan 30 minuten afstand geven 'vol' aan. Hierdoor gaan mensen deze parkeergarage ontwijken, waardoor de instroom vermindert en de parkeergarage na 30 minuten nog niet vol is.

De PRIS informatiesystemen blijken zeer effectief: Het TNO rapport meldt dat het zoekverkeer met 30-50% kan afnemen onder invloed van een PRIS [2]. Daarnaast bleek uit een enquête [5] dat 71% van de bestuurders de PRIS informatiesystemen aanduidde als "gewenst tot zeer gewenst".

Op het moment van schrijven zijn de PRIS informatiesystemen nog niet beschikbaar voor derden. Ze zijn eigendom van de gemeenten en/of bedrijven die de parkeergarages beheren. Wel wordt er in de politiek gekeken of gemeenten en parkeergarages gemotiveerd kunnen worden om deze systemen open te stellen, zoals blijkt uit een artikel op tweakers.net op 15 april 2008[3].

4.1.2 Navigatiesystemen

Met behulp van een GPS navigatiesysteem wordt een bestuurder tijdens een autorit naar haar eindbestemming geleid. Met behulp van een ingebouwde kaart en GPS bepaling zoekt het systeem de kortste route naar eindbestemming. Het doel van een navigatiesysteem is dan ook het verkorten van de reistijd. Oorspronkelijk werden navigatiesystemen verkocht als een apart stuk hardware, met het navigatiesysteem als enige functionaliteit. Dit worden ook wel *dedicated* navigatiesystemen genoemd. Maar tegenwoordig wordt steeds meer navigatiesoftware beschikbaar gesteld op mobiele telefoons. Anders dan de dedicated navigatiesystemen draait deze software op generieke platforms, waardoor het gemakkelijker is deze uit te breiden met nieuwe functionaliteit. Zo heeft fabrikant TomTom een Software Developers Kit uitgebracht waarmee TomTom software op Windows Mobile kan worden uitgebreid. Hier wordt in de volgende paragraaf dieper op ingegaan.

Gebruik

Het gebruik van navigatiesystemen is sterk in opkomst. In Japan is de toepassing van navigatiesystemen gegroeid van 8000 voertuigen in 1993 naar 1 miljoen in 1996 naar 4 miljoen in 1999 [5]. Onderzoeksresultaten over Nederland zijn vooralsnog niet beschikbaar. De verwachting is dat het aantal gebruikers in de

[3]http://tweakers.net/nieuws/52968/staatssecretaris-navigatiesysteem-moet-weg-naar-parkeerplaats-wijzen.html (Pagina bestond nog op 24 september 2009. Alle links naar webpagina's in dit document bestonden nog op 24 september 2009)

toekomst sterk zal stijgen en dat uiteindelijk een groot deel van de bestuurders een navigatiesysteem zal gebruiken.

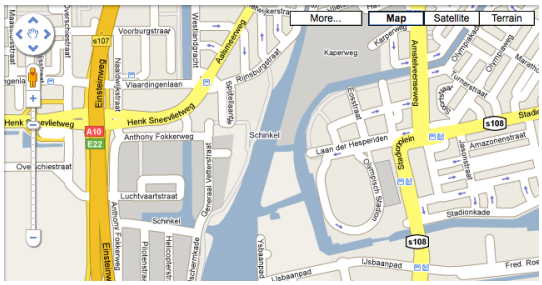
Markt

De markt voor navigatiesystemen is groot en er is veel concurrentie. Bekende leveranciers zijn onder andere Garmin, Tomtom en Mio, waarvan TomTom in Europa het grootste marktaandeel heeft (46%). Op het moment van schrijven zijn er meer dan 17 miljoen TomTom navigatiesystemen in omloop in Europa en Noord-Amerika[4]. Dat de concurrentie groot is, bleek wel in 1994 toen de ANWB aankondigde haar eigen navigatiesysteem uit de markt te halen omdat het onmogelijk bleek voldoende marktaandeel te behalen[5].

Daarnaast is er een tweede type navigatiesysteem in opkomst. Dit type staat continu via het internet in verbinding met een server, waar alle kaart en route-informatie op staat. De benodigde informatie wordt gedownload en tijdelijk opgeslagen op het navigatiesysteem. Voorbeelden hiervan zijn Google Maps[6] en Microsoft Virtual Earth[7]. Deze navigatiesystemen draaien louter op computers (desktop) of mobiele telefoons met internet. Ook hebben ze een beperkte functionaliteit als het gaat om het afbeelden van route-informatie. Google Maps en Microsoft Earth beelden de route bijvoorbeeld niet weer in *3d-view*, maar alleen van bovenaf.

Figuur 6: TomTom 3D view

[4]Uit een persbericht van TomTom op 20 januari 2009, investors.tomtom.com/releasedetail.cfm?ReleaseID=359889
[5]http://tweakers.net/nieuws/38150/anwb-navigatiepakket-engin-aan-de-kant-gezet.html
[6]http://maps.google.com
[7]http://virtualearth.com



Figuur 7: Google maps 2d view

Internet

De manier waarop navigatiesystemen de bestuurder naar haar eindbestemming brengt wordt steeds geavanceerder. De eerste versies gebruikten alleen een interne kaart en GPS-positiebepaling om de kortste route te vinden. Tegenwoordig komen er steeds meer systemen op de markt die gebruik maken van actuele verkeersinformatie. Zo bieden leveranciers TomTom en Garmin al systemen aan die actuele file-informatie afbeelden op het navigatiescherm. Een gevolg van deze ontwikkeling is dat steeds meer navigatiesystemen verbonden moeten zijn met het internet. Hoewel ook hierover geen onderzoeksresultaten beschikbaar zijn, zijn hier wel aanwijzingen over te vinden. Zo zei TomTom in november 2007 in een persbericht dat een substantieel aantal van de TomTom navigatiesystemen in de toekomst een verbinding zal hebben met het internet[8].

4.1.3 ParkeerTomTom

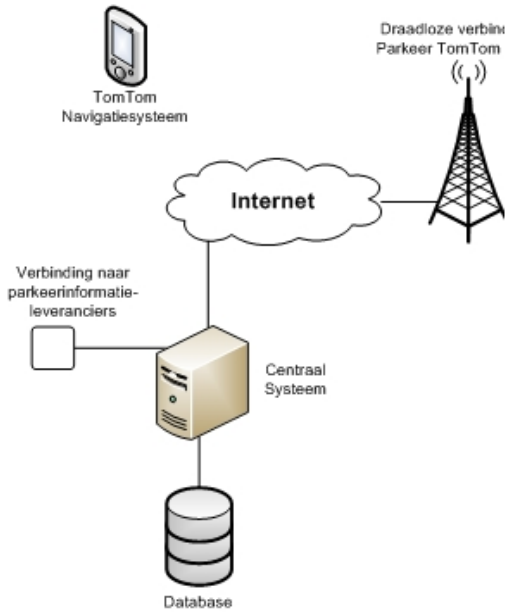
De ParkeerTomTom kan worden gezien als een rijdend PRIS informatiebord [10]. In het TomTom navigatiesysteem is een extra module geprogrammeerd die aan kan geven welke parkeergarages in de buurt van de eindbestemming vol zijn. De bestuurder ziet het resultaat op het scherm van haar navigatiesysteem in plaats van op borden op de weg. Figuur 2 is een screenshot van de ParkeerTomTom in actie.

De ParkeerTomTom werkt met het client-server model. Dit wil zeggen dat er één centrale server is waar alle gegevens worden verwerkt. De TomTom navigatiesystemen en de Parkeergarages (de clients) communiceren met dit centrale systeem (de server). Omdat de parkeergarages tot op heden geen service aanbieden waarmee ze de ParkeerTomTom kunnen voorzien van data is er bij Logica een eigen centrale server ontwikkeld die een Parkeergarage-server-systeem simuleert. Figuur 8 geeft een schematische weergave van de werking van de ParkeerTomTom. Geïnteresseerde lezers kunnen meer details vinden in het technisch ontwerp van de ParkeerTomtom van Michael Tong Sang [10].

De ParkeerTomTom pakt in één keer twee factoren aan die de zoektijd beïnvloeden: De bestuurder weet goed de weg te vinden in de stad door de basisfunctionaliteit van de TomTom (eerste factor). Daarnaast wordt de kans op weigering bij een

[8] http://www.nu.nl/internet/1286707/navigatiesystemen-steeds-vaker-online.html

parkeergarage aanzienlijk verkleind door de extra ParkeerTomTom module die informeert over de bezette parkeergarages (tweede factor).



Figuur 8: Schematisch overzicht ParkeerTomTom: De parkeerTomTom ("TomTom navigatiesysteem", linksboven) op de mobiele PDA van de eindgebruiker maakt verbinding met het internet en zendt haar huidige locatie en locatie van de eindbestemming naar het centrale systeem ("Centraal systeem met database", de computer in het midden). Het centrale systeem heeft op haar beurt een verbinding met de parkeergarages ("Parkeerinformatieleveranciers", linksonder) en kan op deze manier de auto een vrije parkeerplek toewijzen in de buurt van de eindbestemming. Het centrale systeem is een proof-of-concept en test voornamelijk de werking van de client.

4.2 Aantal auto's en parkeergarages

Omdat we het aantal bestuurders en parkeergarages dat meedoet willen maximaliseren (uitgangspunten 2 en 3), is het interessant om te weten hoeveel auto's en parkeergarages er in Nederland zijn. Uit krantenartikelen en onderzoeken maken we een schatting. Let op dat hier alleen rekening wordt gehouden met cijfers uit Nederland en niet uit andere landen.

4.2.1 Aantal auto's

Volgens het CPB telde Nederland eind 2008 meer dan zeven miljoen auto's [3]. In oktober 2003 schreef Nu.nl in een persbericht dat de woensdag de drukste dag was, waarbij gemiddeld twee miljoen Nederlanders tegelijk op de weg zijn[9]. Het aantal Nederlanders dat in 2008 onderweg is zal zijn toegenomen (omdat het aantal auto's is toegenomen), maar in dezelfde orde van grootte zijn gebleven. We gaan daarom uit van een piekbelasting op de weg van twee miljoen auto's.

[9]http://www.nu.nl/algemeen/224270/elke-woensdagochtend-2-miljoen-mensen-onderweg.html

Dit is een ruime schatting, het werkelijke aantal zal hier voorlopig niet boven komen.

4.2.2 Aantal parkeergarages

In het marktonderzoek "Marktschets parkeren 2008" wordt gesteld dat er totaal 8,9 miljoen openbare parkeerplaatsen zijn [4]. Slechts 2% ervan bevindt zich in parkeergarages. Dit betekent dat er 180.000 parkeerplaatsen in Nederland zijn die zich in parkeergarages bevinden.

5 Vereisten

Wanneer we de doelstelling met de informatie uit het vooronderzoek combineren, kunnen er een aantal fundamentele beslissingen worden genomen omtrent de werking van de ParkeerHeer. Deze beslissingen worden in dit hoofdstuk gemaakt. Het resultaat is een kleine set van de meest belangrijke functionele en niet-functionele eisen waar de ParkeerHeer aan moet voldoen. In Bass et al. [9] worden dit ook wel de *Architectural Drivers* genoemd: Het zijn de eisen die bepalend zijn voor de architectuur. Wanneer de *Architectural Drivers* bekend zijn, kan de architectuur verder worden uitgewerkt. In hoofdstuk 3 spraken we over het gebruik van de QOC methode, deze passen we in dit hoofdstuk grondig toe. We beginnen bij de uitgangspunten die we in de doelstelling (hoofdstuk 2) hebben opgesteld.

5.1 Uitgangspunten

In de doelstelling worden vier uitgangspunten genoemd waar aan moet worden voldaan bij de bouw van de Parkeerheer. Deze behandelen we nu één voor één.

5.1.1 Verkleinen van de zoektijd

De hoofdfunctionaliteit van de ParkeerHeer zal zijn, dat ze de gemiddelde zoektijd van de auto's verkleint vergeleken met situaties waar de ParkeerHeer niet gebruikt wordt. Dit kunnen we als volgt formuleren in een eis:

R1: De ParkeerHeer zal de zoektijd verkleinen

In de doelstelling (hoofdstuk 2) schreven we dat er drie factoren zijn die deze zoektijd kunnen beïnvloeden. De vraag is nu welke van deze factoren gaan worden gemanipuleerd. Ook willen we weten hoevéél de zoektijd moet worden verkleind. Kortom: Deze eis is in dit stadium nog niet specifiek genoeg. We laten daarom de volgende ontwerpvragen open:

- *Q1: Welke factoren gaan worden beïnvloed om de zoektijd te verkleinen?* (5.2)
- *Q2: Hoeveel moet de ParkeerHeer de zoektijd verkleinen?* (5.3)

5.1.2 Maximaliseren van het aantal bestuurders & parkeerplaatsen

Het tweede en derde uitgangspunt geven aan dat de ParkeerHeer *schaalbaar* moet zijn: Het moet een groot aantal bestuurders en parkeergarages kunnen bedienen en er moet een manier bedacht worden om een grote groei van gebruikers te ondersteunen. In het vooronderzoek kwam naar voren dat er maximaal 2 miljoen auto's tegelijk op de weg zijn. Het aantal openbare parkeerplaatsen ligt op 8,9 miljoen, waarvan 2% in parkeergarages. Dit leidt tot R2 en R3, die hieronder zijn uitgewerkt.

Meer gebruikers betekent automatisch dat er meer rekenkracht nodig is en dat het langer duurt om een oplossing uit te rekenen. Het is daarom belangrijk dat we een uitspraak doen over de *performance* van de ParkeerHeer. Met performance bedoelen we dat de tijd die de ParkeerHeer nodig heeft om een oplossing

uit te rekenen niet te lang duurt. Een bestuurder wil niet twee uur wachten tot ze een parkeerplaats toegewezen krijgt. Dit wordt als volgt geformuleerd:

- *Q3: Hoe snel moeten de auto's aan vrije parkeerplaatsen worden toegewezen?* (5.6)

R2: Het aantal bestuurders dat is aangesloten op de ParkeerHeer zal zo groot mogelijk zijn.

De ParkeerHeer moet een zo groot mogelijk aantal bestuurders kunnen bedienen. Dit delen we op in de volgende twee eisen:

R2-a: De ParkeerHeer zal tot 20.000 auto's kunnen bedienen.

Dit is 1% van het maximaal aantal weggebruikers tijdens spitsuur. We nemen dit percentage omdat in eerste instantie niet alle verkeersdeelnemers de ParkeerHeer zullen gebruiken. Het gekozen percentage is daarom een ruwe schatting.

R2-b: De ParkeerHeer zal dusdanig worden ingericht dat het door hardware-uitbreiding tot 2.000.000 auto's kan bedienen.

Dit is 100% van het huidige maximaal aantal weggebruikers tijdens spitsuur. Met hardware-uitbreiding kan worden gedacht aan het repliceren van de ParkeerHeer door het op meerdere computers te laten draaien. Hierdoor is er meer processorkracht beschikbaar. De precieze invulling kunnen we op dit moment nog niet beantwoorden, dat doen we in het volgende deel. We laten daarom de volgende ontwerpvraag open:

- *Q4: Hoe kan de ParkeerHeer geschaald worden zodat het meer auto's kan ondersteunen?* (UC12)

R3: Het aantal parkeerplaatsen dat is aangesloten op de ParkeerHeer zal zo groot mogelijk zijn.

De ParkeerHeer moet kennis hebben van zoveel mogelijk parkeerplaatsen. Deze eis kunnen we opdelen in het volgende:

R3-a: De ParkeerHeer zal tot 180.000 parkeerplaatsen kunnen bedienen.

Dit is ongeveer 2% van het totaal aan parkeerplaatsen in Nederland, gelijk aan het aantal parkeerplaatsen in parkeergarages. We nemen dit percentage omdat in eerste instantie niet alle parkeergarages aangesloten zullen worden op de ParkeerHeer. Het gekozen percentage is daarom een ruwe schatting.

R3-b: De ParkeerHeer zal dusdanig worden ingericht dat het door hardware-uitbreiding tot 8,9 miljoen parkeerplaatsen kan bedienen.

Dit is 100% van het totaal aantal parkeerplaatsen. In het ideale geval zijn ze allemaal aangesloten op de ParkeerHeer. Ook deze eis zal in het volgende deel verder worden uitgewerkt. We laten daarom de volgende ontwerpvraag open:

- *Q5: Hoe kan de ParkeerHeer geschaald worden zodat het meer parkeerplaatsen kan ondersteunen?* (UC12)

5.1.3 Uitbreiden van functionaliteit

Het vierde uitgangspunt zegt iets over de uitbreidbaarheid van de ParkeerHeer. In de toekomst willen wellicht gehandicapten er ook gebruik van gaan maken, of willen we extra parkeerplaatsen in kunnen bouwen bij speciale evenementen.

R4: De ParkeerHeer zal gemakkelijk aangepast kunnen worden om extra functionaliteit te ondersteunen

Hoe we de functionaliteit precies willen aanpakken, en op welke manier, specificeren we in het volgende deel. We laten daarom de volgende ontwerpvraag open:

- *Q6: Welke delen van de ParkeerHeer kunnen worden aangepast en welke niet?* (UC9 & UC10)

5.2 Verkleinen van de zoektijd

Nu de uitgangspunten zijn vertaald in eisen en ontwerpvragen kunnen deze één voor één worden uitgewerkt. We beginnen bij de eerste, die volgt uit R1:

Q1: Welke factoren gaan worden beïnvloed om de zoektijd te verkorten?

We doelen hierbij op de drie factoren die we in de doelstelling hebben benoemd (hoofdstuk 2).

5.2.1 Criteria

We definiëren de volgende drie criteria:

- **Bereik**: In hoeverre kan een groot aantal mensen er gebruik van maken en sluit het aan bij R2, R3 en R4?
- **Impact**: In hoeverre is het iets nieuws, wat voor extra functionaliteit zorgt en sluit het aan bij het eerste uitgangspunt uit paragraaf 2.1 en R1?
- **Haalbaarheid:** In hoeverre is het praktisch haalbaar om deze factor te beïnvloeden met een software applicatie?

5.2.2 Opties

Zoals eerder aangegeven in de doelstelling zijn er drie manieren om de zoektijd te verkorten:

- Bekendheid van de bestuurder met de stad vergroten.
- Het aantal keer dat een bestuurder een bezette parkeerplaats aantreft (het aantal weigeringen) verminderen.
- De eisen die de bestuurder stelt aan een parkeerplaats versoepelen.

Wanneer we eerste factor gaan beïnvloeden zal de impact niet erg groot zijn, omdat de bekendheid van de bestuurder voor een groot gedeelte al wordt vergroot door navigatiesystemen en informatieborden (paragraaf 4.1.2). De derde factor is praktisch moeilijk haalbaar, omdat hiervoor een cultuuromslag plaats moet vinden: mensen moeten genoegen nemen met een parkeerplaats waarbij ze verder moeten lopen. Hierbij zal de oplossing niet liggen in het maken van een applicatie, maar meer in bijvoorbeeld een bewustwordingscampagne op radio en tv. Het verminderen van het aantal weigeringen (de tweede factor) is hierbij de beste oplossing: Omdat er nog niet veel applicaties zijn die dit doen zal de impact groot zijn. De navigatiesystemen kunnen gebruikt worden als medium om het aantal weigeringen te verkleinen. In dit geval is het bereik groot, aangezien een hoog percentage van de bestuurders een navigatiesysteem gebruikt. Daarnaast is dit een haalbare oplossing. Dit is aangetoond door de ParkeerTomTom en de PRIS informatieborden.

5.2.3 Ontwerpbeslissing: Verminderen van het aantal weigeringen

We gaan het aantal weigeringen verminderen. Ter verduidelijking zetten we de criteria tegenover de drie opties in een tabel:

	Bereik	Impact	Haalbaarheid
Verbeteren bekendheid	Groot	Laag	Goed
Verminderen aantal weigeringen	Groot	Hoog	Goed
Aanpassen eisen bestuurder	Groot	Hoog	Laag

5.2.4 Eisen

We kunnen de genomen beslissing bijna letterlijk overnemen als eis. Wel specificeren we de eis iets meer, om te voorkomen dat de eis op verschillende manieren kan worden geïnterpreteerd:

R1-a: De ParkeerHeer zal het aantal keer dat auto's geweigerd worden bij een parkeergarage omdat deze geen vrije plaatsen heeft, verminderen.

Aan deze keuze zullen we een aantal voorwaarden moeten verbinden. Zo bedoelen we met verminderen, dat het aantal weigeringen afneemt ten opzichte van auto's die de applicatie niet gebruiken. Daarnaast gaan we er hier vanuit dat er daadwerkelijk tijdswinst te behalen valt. Het kan nou eenmaal voorkomen dat er geen parkeergarages zijn die nog vrije plekken hebben.

5.2.5 Vervolgvragen

De gestelde eis is nog niet helemaal goed uitgewerkt. Zo weten we nog niet op welke manier het aantal weigeringen gaat worden verminderd en hoe de informatie wordt aangeleverd en verwerkt. Daarom stellen we de volgende ontwerpvragen:

- *Q7: Op welke manier gaan het aantal weigeringen worden verminderd?* (5.3)
- *Q8: Hoe gaan de bestuurders de benodigde informatie aanleveren en aangeleverd krijgen?* (UC5 & UC6)
- *Q9: Hoe gaan de parkeerplaatsen de benodigde informatie aanleveren en aangeleverd krijgen?* (UC7)

5.3 Verminderen van het aantal weigeringen

Nu er besloten is om het aantal weigeringen te verminderen moet worden bepaald hoe dit gaat gebeuren. De vraag die hier wordt beantwoord is:

Q7: Op welke manier gaat het aantal weigeringen worden verminderd?

5.3.1 Criteria

De opties worden op de volgende drie criteria beoordeeld:

- **Foutgevoeligheid**: In hoeverre kunnen we met zekerheid zeggen dat deze optie een juiste oplossing berekent?
- **Functionaliteit**: In hoeverre voldoet deze optie aan de functionaliteit die we willen bereiken, namelijk het verminderen van het zoekend verkeer?
- **Haalbaarheid:** Is deze optie praktisch haalbaar?

5.3.2 Opties

De volgende drie opties worden overwogen:

Auto's informeren over parkeergarages

Zoals bij de ParkeerTomTom en de PRIS informatieborden kunnen de bestuurders geïnformeerd worden over de status van de garages. Een groot nadeel hiervan is dat het systeem zichzelf tegenwerkt: De borden geven een schatting in de toekomst, maar diezelfde toekomst beïnvloeden ze door deze schatting door te geven aan de bestuurders. Hierdoor komt het voor dat veel bestuurders een bepaalde parkeergarage ontwijken waardoor er toch weer plaatsen vrijkomen.

Daarnaast wordt er met deze methode geen rekening gehouden met het dynamische aspect van de auto's en parkeergarages: Stel dat er 100 auto's willen rijden naar een parkeergarage waar nog 50 plaatsen vrij zijn. Wanneer de ParkeerHeer alleen informeert over het aantal beschikbare plaatsen zullen alle 100 auto's naar deze parkeergarage rijden, en worden er dus 50 afgewezen.

Een voordeel van deze methode is dat de gebruiker zelf kan beslissen wat ze doet. In deze context kan worden beargumenteerd dat de foutgevoeligheid erg klein is, omdat het nog altijd de gebruiker is die de keuze maakt. Het enige dat de ParkeerHeer doet wanneer voor deze optie gekozen wordt, is informeren over het aantal parkeerplaatsen.

Omdat het informeren reeds is gerealiseerd in de PRIS informatieborden en de ParkeerTomTom kan gesteld worden dat deze optie praktisch haalbaar is.

Auto's toewijzen aan parkeergarages en hier plekken reserveren

De ParkeerHeer kan de auto's toewijzen aan een parkeergarage en deze plek bij de parkeergarage reserveren. Omdat de plaatsen worden gereserveerd kan er een zeer nauwkeurige schatting worden gegeven over het aantal vrije plaatsen bij een parkeergarage. Het is echter moeilijk haalbaar om bestuurders bij de poort te *authenticeren* als de gebruikers die de plaats hebben gereserveerd. Met authenticeren wordt bedoeld: Het aantonen van de bestuurder dat *zij* degene is geweest die de parkeerplaats heeft gereserveerd en dat ze zich dus niet als diegene voordoet om gemakkelijk een parkeerplaats te kunnen krijgen.

Daarnaast zijn er parkeergarages met slechts één ingang wat het moeilijk maakt om gereserveerde auto's voor te laten gaan als er een rij staat.

Een derde beperking van deze optie is, dat de keuzevrijheid van de eindgebruiker wordt beperkt. Wanneer de informatie waarop de ParkeerHeer haar beslissing neemt incorrect is, zal ze de bestuurder naar een verkeerde parkeergarage worden gestuurd, zonder dat de bestuurder daar iets aan kan doen.

Auto's toewijzen aan parkeergarages en deze plekken niet reserveren

Bij de derde optie worden de auto's wel toegewezen aan parkeergarages, maar worden deze plekken niet gereserveerd. Het kan dus voorkomen dat een auto is toegewezen aan een parkeergarage die uiteindelijk toch vol blijkt te zijn. Dit betekent dus dat deze optie een zekere foutgevoeligheid heeft. Om dit te verminderen kunnen dezelfde toekomst voorspellende algoritmen van de PRIS systemen worden gebruikt, met als toevoeging dat de extra informatie van de auto's die op het systeem zijn aangesloten wordt meegerekend. Hierdoor wordt het probleem bij informeren waarbij het systeem zichzelf tegenwerkt grotendeels verholpen. Daarnaast kan een bepaalde foutmarge worden gehanteerd om het aantal weigeringen te minimaliseren. Kortom: Er zijn tal van manieren om de foutgevoeligheid aan te pakken. Wanneer we voor deze optie kiezen zullen we dit nader moeten onderzoeken.

5.3.3 Ontwerpbeslissing: Toewijzen van auto's aan parkeergarages, dirigeren zonder reserveren

We kiezen voor de derde optie: Toewijzen van auto's aan parkeergarages, zonder de plekken te reserveren. Ter verduidelijking staan de keuzes nog een keer op een rij in tabel 5.3.3

	Foutgevoeligheid	Functionaliteit	Haalbaarheid
Informeren	Klein	Laag	Goed
Dirigeren & Reserveren	Klein	Hoog	Slecht
Dirigeren & Niet reserveren	Groot	Hoog	Goed

5.3.4 Eisen

Als gevolg van deze beslissing stellen we de volgende eisen aan de ParkeerHeer:

R1-b: De ParkeerHeer zal auto's naar vrije parkeerplaatsen dirigeren.

R1-c: De ParkeerHeer zal geen vrije plaatsen reserveren bij parkeergarages.

R1-d: De ParkeerHeer zal rekening houden met alle aangesloten auto's en parkeergarages.

Dit betekent dat er geen 50 auto's worden gestuurd naar een parkeergarage met 40 vrije plaatsen.

R1-e: De ParkeerHeer zal een schatting maken van het aantal vrije parkeerplaatsen in de toekomst.

De bestuurder zal nog een bepaalde tijd moeten reizen voor ze bij de parkeergarage arriveert. Wanneer de toewijzing gebeurt op basis van het aantal vrije parkeerplaatsen op het moment dat de bestuurder haar verzoek verstuurt, is de kans groter dat de parkeergarage alsnog bezet is als ze arriveert. Daarom zal de ParkeerHeer de bestuurder toewijzen op basis van toekomstvoorspellingen: Arriveert de bestuurder in 30 minuten bij de eindbestemming, dan zal de ParkeerHeer kijken of er over 30 minuten een plekje vrij is in de buurt van haar eindbestemming.

5.3.5 Vervolgvragen

De genomen ontwerpbeslissing heeft nog een aantal andere gevolgen: Bij de bespreking van de optie werd een aantal mogelijkheden genoemd om de foutgevoeligheid te verkleinen. De vervolgvragen die we hierbij stellen is:

- *Q10: Hoe kan de foutgevoeligheid worden verminderd?* (5.6)
- *Q11: Welke eisen stelt de bestuurder aan het toewijzen?* (5.6)

Daarnaast kan er nu gekeken worden of het mogelijk is de reeds bestaande systemen te gebruiken voor de toewijzing en in welk deel van de ParkeerHeer een oplossing berekend gaat worden. Hiervoor stellen we de volgende vragen:

- *Q12: Welke bestaande systemen kunnen gebruikt worden om de toewijzing tot stand te laten komen?* (5.4)
- *Q13: Welk subsysteem gaat de oplossing berekenen?* (5.4)

5.4 Hergebruik van bestaande systemen

Nu we besloten hebben dat de zoektijd verkleind gaat worden door het aantal weigeringen te verminderen, kunnen we kijken of we reeds bestaande systemen kunnen gebruiken bij de ontwikkeling van de ParkeerHeer. In deze paragraaf wordt gekeken of de bestaande systemen die besproken zijn in het vooronderzoek (hoofdstuk 4) kunnen worden gebruikt bij de ontwikkeling van de ParkeerHeer. Hierdoor worden we meteen gedwongen een aantal keuzes te maken over de te gebruiken architectuur. Met andere woorden, de ontwerpvragen *Q12* en *Q13* uit de vorige paragraaf (5.3) worden hier beantwoord.

5.4.1 Criteria

De criteria die we hanteren voor het maken van een keuze zijn:

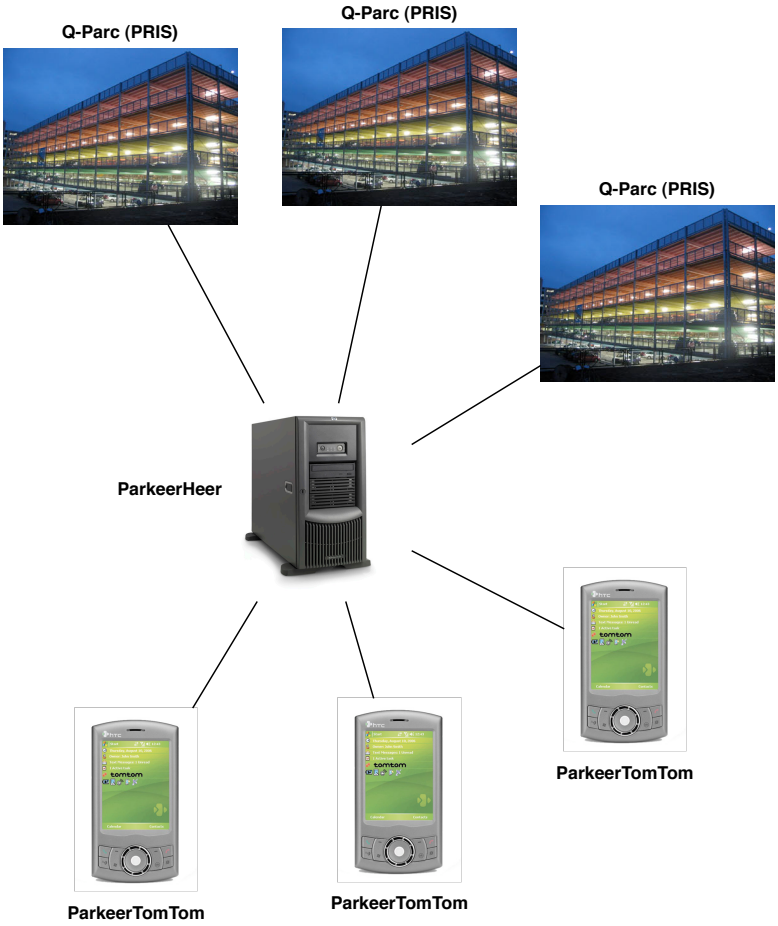
- **Bereik**: In hoeverre is het mogelijk om een groot aantal bestuurders en parkeergarages aan te sluiten (R2, R3 en R4)?
- **Uitbreidbaarheid**: In hoeverre is het mogelijk om nieuwe functionaliteit toe te voegen (R5)?
- **Haalbaarheid**: In hoeverre is deze keuze praktisch haalbaar?
- **Ontwikkelsnelheid**: In hoeverre verkleint het gebruik van deze systemen de tijd die nodig is om de ParkeerHeer te ontwikkelen?
- **Veiligheid**: In hoeverre is deze optie veilig: Wordt de privacy van de eindgebruikers gewaarborgd en kunnen kwade partijen de ParkeerHeer misbruiken voor de verkeerde doelen? Een voorbeeld is het sturen van extra auto's naar een bepaalde parkeergarage, zodat deze meer geld verdient.

5.4.2 Opties

We bespreken de volgende opties:

Ontwikkel een server specifiek voor ParkeerTomTom en PRIS informatiesysteem

Met deze optie wordt gekozen voor een *client-server* architectuur, waarbij de ParkeerHeer de server is en de PRIS garages en ParkeerTomTom systemen worden gezien als clients. De clients geven de benodigde informatie door aan de ParkeerHeer. Vervolgens kan de ParkeerHeer hiermee een oplossing uitrekenen en terugsturen naar de clients. De ParkeerHeer communiceert hierbij alleen met de besproken bestaande systemen (ParkeerTomTom en PRIS garages). Overige systemen worden geheel buiten beschouwing gelaten. Figuur 9 is een voorbeeld van een dergelijke client-server opstelling.



Figuur 9: Schematisch overzicht Specifieke ParkeerHeer server: De ParkeerHeer fungeert als server voor alleen de ParkeerTomTom en PRIS navigatiesystemen

Deze opstelling heeft de volgende voordelen:

- De ontwikkeltijd is redelijk kort, omdat de PRIS informatiesystemen en ParkeerTomTom al bestaan. Er hoeven geen nieuwe *protocollen* ontwikkeld te worden. Een protocol is een afspraak tussen de client en de server over de manier waarop deze met elkaar communiceren. De nadruk van de ontwikkeling zal dan ook liggen in het bedenken van de juiste algoritmen en/of heuristieken die de auto's aan de vrije parkeerplaatsen toewijzen.
- De specifieke functionaliteit van de PRIS en ParkeerTomTom systemen kunnen worden gebruikt. Er hoeft geen rekening te worden gehouden met andere systemen, waardoor bijvoorbeeld de voorgedefiniëerde routes van TomTom kunnen worden gebruikt om de functionaliteit te vergroten[10].
- Omdat de ParkeerHeer een centrale server is die alle verzoeken van parkeergarages en auto's afhandelt, kan deze ook dienen als veiligheidsmecha-

[10]Zie ook het vooronderzoek, hoofdstuk 4.

nisme om de privacy van de gebruikers te waarborgen. De ParkeerHeer kan bijvoorbeeld op een centrale plek worden gezet, waar parkeergarages en bestuurders geen fysieke toegang toe hebben. Zo kunnen deze de informatie niet manipuleren door bestanden te wijzigen.

Daarnaast kent deze opstelling de volgende nadelen:

- Uit het vooronderzoek bleek dat er veel verschillende navigatiesystemen zijn. Omdat in deze optie alleen TomTom navigatiesystemen kunnen voorzien van een parkeer uitbreiding, is het bereik van deze optie zeer beperkt.
- Deze optie gaat ervan uit dat het protocol van de PRIS garages bekend is en dat deze partijen bereid zijn mee te werken aan de ontwikkeling. Dit is helaas niet het geval[11]. Daarom is de haalbaarheid van deze optie op het moment van schrijven gering.
- Ondanks de beveiligingsmechanismen kunnen de parkeergarages verkeerde informatie versturen naar de ParkeerHeer met als doel meer auto's aan te trekken en dus meer inkomsten te genereren.

Ontwikkel een generieke server voor alle navigatiesystemen en parkeergarages

Met deze optie is de ParkeerHeer ook een centrale server die data verzamelt van de clients, maar dit keer kunnen de clients meerdere typen systemen zijn. Onderstaand figuur 10 illustreert dit.

[11] Zie ook hoofdstuk 4, het vooronderzoek

Figuur 10: Schematisch overzicht Generieke ParkeerHeer server: De ParkeerHeer fungeert als server voor meerdere typen navigatiesystemen en parkeergarages.

Met deze opstelling wordt er een generiek protocol ontwikkeld dat met de parkeergarages en navigatiesystemen communiceert. Op deze manier wordt het aansluiten van nieuwe systemen vergemakkelijkt. Hoe dit protocol er precies uit ziet en hoe het geïmplementeerd gaat worden zal verder moeten worden uitgezocht.

De ParkeerHeer server kan los ontwikkeld worden van de PRIS en ParkeerTomTom. Later kunnen deze applicaties alsnog worden aangesloten. Hiermee kunnen we de *scope* van dit onderzoek vernauwen en ons richten op de algoritmen en/of heuristieken die de auto's aan parkeergarages toewijzen, en de architectuur van de server in het algemeen. Hoe de software op de parkeergarages of navigatiesystemen eruit gaat zien laten we achterwege. Om de ParkeerHeer te testen kan er een test-parkeergarage worden geschreven die gebruik maakt van het protocol en de ParkeerHeer voorziet van data. Hetzelfde geldt voor de auto's. Deze optie heeft de volgende voordelen:

- Groot potentieel bereik: Alle navigatiesystemen en parkeergarages kunnen

worden aangesloten.

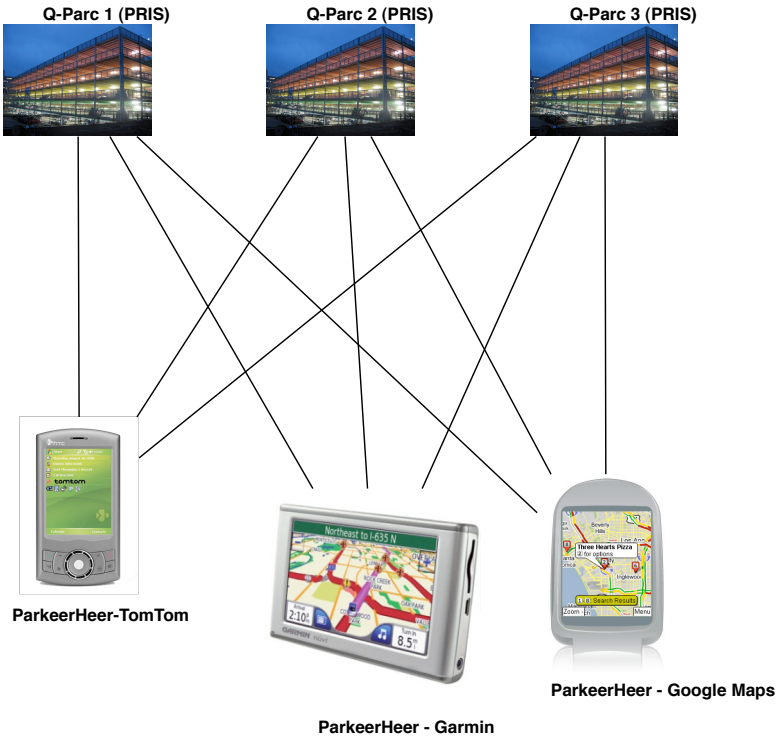
- Omdat de ParkeerHeer een centrale server is die alle verzoeken van parkeergarages en auto's afhandelt, kan deze ook dienen als veiligheidsmechanisme om de privacy van de gebruikers te waarborgen. De ParkeerHeer kan bijvoorbeeld op een centrale plek worden gezet, waar parkeergarages en bestuurders geen fysieke toegang toe hebben. Zo kunnen deze de informatie niet manipuleren door bestanden te wijzigen.
- De scope kan worden vernauwd. Met scope bedoelen we de onderdelen die wel onderzocht en uitgewerkt gaan worden en ook vooral de delen die we niet verder uitwerken. Door de beperkte beschikbare tijd zal namelijk niet alles uitgewerkt kunnen worden. Wanneer voor deze optie gekozen wordt, kan besloten worden dat alleen de ParkeerHeer server wordt ontwikkeld, alsmede het protocol waarmee de ParkeerHeer communiceert met de clients. De werkelijke aansluiting van de clients zal in een volgende fase moeten worden uitgewerkt.
- Met deze generieke oplossing kan later gekeken worden of de communicatie ook via andere middelen kan plaatsvinden, bijvoorbeeld via SMS berichten.

En de volgende nadelen:

- Na de ontwikkeling van de ParkeerHeer zal er nog een extra ontwikkelfase moeten komen waarin minimaal één parkeergaragesysteem en één navigatiesysteem wordt aangesloten voordat de ParkeerHeer in gebruik kan worden genomen.
- Wanneer het protocol eenmaal is gemaakt, en meerdere typen parkeergarages en navigatiesystemen zijn aangesloten, zal het moeilijk worden dit aan te passen. Hier zal een oplossing voor gevonden moeten worden.

Garages als servers, auto's als clients

Voor de laatste optie laten we de garages dienen als servers, en de auto's als clients. De navigatiesystemen in de auto's zoeken de beste parkeergarage in de buurt van eindbestemming en maken hier contact mee. Vervolgens doen ze bij deze parkeergarage een verzoek om te parkeren. De ParkeerTomTom kan in dit geval niet gebruikt worden, omdat deze uitgaat van een centrale server. Wel kunnen de PRIS informatiesystemen gebruikt kunnen worden. De ParkeerHeer is in deze optie de client, die contact maakt met de parkeergarages. Onderstaand figuur 11 illustreert dit.



Figuur 11: Schematisch overzicht Generieke ParkeerHeer server: De ParkeerHeer zijn de clients die met de parkeergarages communiceren

Deze optie heeft de volgende voordelen:

- De taken worden gelijkmatig verdeeld over de parkeergarages, wat de performance en schaalbaarheid verhoogt. In tegenstelling tot de vorige twee opties is er niet één server die alle informatie afhandelt.

En de volgende nadelen:

- Voor elk type navigatiesysteem moet een eigen ParkeerHeer applicatie ontworpen worden.
- Nieuwe functionaliteit (uitgangspunt 4) moet voor elke ParkeerHeer instantie worden ontwikkeld.
- Parkeergarages met een ander protocol worden niet ondersteund, dit beperkt het bereik.
- De ParkeerHeer staat op de navigatiesystemen en op de parkeergarages: Deze twee partijen zouden de software dusdanig zelf kunnen aanpassen dat ze voorrang krijgen of meer auto's toegewezen krijgen.

5.4.3 Ontwerpbeslissing: Ontwikkel een generieke server

We gaan een generieke server ontwikkelen (optie 2). Hoofdargument hiervoor is, dat deze optie goed aansluit op bestaande systemen en de mogelijkheid biedt tot

relatief simpele uitbreiding in zowel type clients als in functionaliteit. Hieronder zetten we de voor en nadelen nogmaals in een tabel:

Criterium	Specifieke Server	Generieke Server	Garages als Server
Bereik	Laag	Groot	Laag
Uitbreidbaarheid	Moeilijk	Mogelijk	Moeilijk
Haalbaarheid	Moeilijk	Goed	Slecht
Ontwikkeltijd	Snel	Snel	Middel
Veiligheid	Voldoende	Voldoende	Slecht

Belangrijke criteria zijn de haalbaarheid en het bereik: Deze oplossing kunnen we bouwen zonder afhankelijk te zijn van de PRIS informatiesystemen. Tegelijkertijd kunnen we een systeem maken dat in de toekomst alle parkeergarages en auto's met een navigatiesysteem kan bedienen.

We vernauwen hier de scope zoals beschreven: We ontwikkelen alleen de ParkeerHeer server, en ontwerpen hierbij een protocol waarmee clients kunnen worden aangesloten. Het aansluiten van de clients zal in een volgende ontwikkelfase plaatsvinden.

5.4.4 Eisen

Deze beslissing resulteert in de volgende eisen:

R5: De ParkeerHeer zal een centrale server worden die haar diensten aan verschillende typen clients beschikbaar stelt.

R5-a: De ParkeerHeer zal met de bestuurder communiceren via een navigatiesysteem.

De ParkeerHeer gaat communiceren met navigatiesystemen. Hierbij kan gebruik worden gemaakt van interne kaart van de navigatiesystemen.

R5-b: De ParkeerHeer zal met de parkeergaragesystemen communiceren.

Door te communiceren met de parkeergaragesystemen kan gebruik worden gemaakt van het voorspellend vermogen van de parkeergarages.

R5-c: De communicatie tussen de ParkeerHeer en haar clients zal via het internet verlopen.

De parkeergarages en navigatiesystemen zullen via het internet met de ParkeerHeer moeten communiceren. Dit is de simpelste optie. Een eerder genoemd alternatief, de communicatie via SMS-berichten, laten we hierbij achterwege. Later zal dit alsnog kunnen worden toegevoegd. Dit zal de architectuur van de ParkeerHeer moeten kunnen ondersteunen. Hier komen we later op terug (Hoofdstuk 8).

R5-d: De ParkeerHeer zal op een veilige manier met haar clients communiceren.

Met veiligheid wordt bedoeld: Het waarborgen van de privacy van de opgeslagen gebruikersgegevens en het voorkomen van misbruik van de ParkeerHeer. Door te stellen dat de ParkeerHeer een centrale server wordt, garanderen we al dat de Parkeergarages en navigatiessystemen geen directe toegang hebben tot alle gebruikers, en dus ook niet veel informatie kunnen misbruiken. De ParkeerHeer zal altijd afhankelijk zijn van de aangeleverde informatie van auto's en parkeergarages, maar door de communicatie te beveiligen kunnen we in ieder geval voorkomen dat iemand zich voordoet als iemand anders, of meeluistert.

Vervolgvragen

De gemaakte ontwerpbeslissing brengt een aantal onduidelijkheden met zich mee: Hoe gaat de ParkeerHeer precies communiceren met de clients, welke gegevens moeten er precies verzonden worden en hoe gaat de communicatie precies beveiligd worden? Dit vertalen we in de volgende ontwerpvragen:

- *Q14: Welke gegevens moeten er verzonden worden van en naar de parkeergarages?* (5.5 en UC7)
- *Q15: Welke gegevens moeten er verzonden worden van en naar de navigatiesystemen?* (5.5, UC5, UC6)
- *Q16: Hoe kan de communicatie tussen ParkeerHeer en client worden beveiligd?* (5.5, UC5, UC7)

5.5 Communicatie tussen client en server

We hebben nu besloten dat de ParkeerHeer een centrale server wordt die diensten aanbiedt aan parkeergarages en bestuurders (via navigatiesystemen). Daarnaast hebben we de vraag gesteld, hoe de communicatie tot stand moet komen (Q14, Q15), en hoe deze het beste kan worden beveiligd (Q16). Deze vragen beantwoorden we deels in deze paragraaf. We treden hier nog niet in detail, maar nemen een ontwerpbeslissing over de manier waarop de informatie wordt uitgewisseld (synchroon of asynchroon) en de plek waar de meeste berekeningen worden uitgevoerd.

5.5.1 Criteria

De volgende criteria zijn van belang:

- **Performance**: In hoeverre wordt de performance door deze keuze beïnvloed?
- **Aanpasbaarheid**: In hoeverre is het mogelijk nieuwe functionaliteit toe te voegen?
- **Bereik**: In hoeverre kunnen in de toekomst meer navigatiesystemen en parkeergarages worden aangesloten met deze optie?
- **Veiligheid**: In hoeverre kan voorkomen worden dat parkeergarages of bestuurders frauderen met gegevens?

5.5.2 Opties

Er zijn twee dilemma's: Het gebruik van synchrone of asynchrone communicatie en het gebruik van thin, rich of thick clients:

Asynchrone of synchrone communicatie

Voor *asynchrone* communicatie geldt, dat de verbinding tussen client en server niet blijvend is. De client verstuurt een bericht en wacht niet op antwoord. Dit komt de performance ten goede, omdat de server geen verbindingen open hoeft te houden. *Synchrone* communicatie doet dit wel. Een voordeel hiervan is, dat de auto kan wachten tot ze een parkeerplaats toegewezen heeft gekregen. Maar met een potentieel van 2 miljoen aangesloten bestuurders (zie vooronderzoek) is dit niet goed haalbaar: De performance van de ParkeerHeer zal drastisch dalen wanneer ze voor elk van deze bestuurder een verbinding open moet houden. Er moet dus gekozen worden voor asynchrone communicatie.

Thin, rich of thick clients

Deze drie opties gaan over de hoeveelheid *business logic* die op de clients gaat worden uitgevoerd. Met business logic worden de algoritmen bedoeld waarmee de verzamelde informatie wordt gevormd tot bruikbare data. De locatie die verantwoordelijk is voor de meeste business logic zal dan ook de meeste rekenkracht nodig hebben.

Het verschil tussen thin, rich en thick clients verduidelijken we met een concreet voorbeeld: Stel dat de clients hun locatie moeten versturen naar de ParkeerHeer. Om een oplossing uit te rekenen, zal eerst de stad van de locatie moeten worden gevonden. Deze zal moeten worden berekend uit de (lat,lon) coördinaten van de GPS gegevens van de navigatiesystemen.

Bij *thin* clients staat er bijna geen business logic op de clients: Deze sturen alle benodigde informatie naar de Parkeerheer en deze rekent alles uit. De locatie zal dus in meest onaangepaste vorm worden verstuurd, bijvoorbeeld als (lat,lon) coördinaat. De ParkeerHeer zal vervolgens de stad moeten opzoeken waar deze coördinaten bij horen.

Thick clients zijn het omgekeerde van thin clients: Deze rekenen zoveel mogelijk zelf uit. De (lat,lon) coördinaten worden op de clients omgezet naar een stad, en deze wordt naar de ParkeerHeer gestuurd.

Rich clients zijn een combinatie van de twee, waarbij een deel van de business logic op de clients wordt uitgerekend, en een deel op de server.

In het algemeen geldt dat software beter aanpasbaar is wanneer het op één centraal punt staat. In dit geval op de ParkeerHeer server. Daarom streven we naar thin clients.

5.5.3 Ontwerpbeslissing: Asynchroon, thin clients

Om de performance te maximaliseren en de uitbreidbaarheid zo gemakkelijk mogelijk te maken kiezen we voor asynchrone, thin clients. Dit resulteert in de volgende eisen:

R4-a: De Clients van de ParkeerHeer zullen zo min mogelijk business logic bevatten.

R5-e: De ParkeerHeer zal communiceren met haar clients via asynchrone communicatie.

5.6 Wensen van de bestuurder

In deze paragraaf gaan we dieper in op de wensen van de bestuurder. Immers, als we hier geen rekening mee houden, zullen er niet veel mensen zijn die de ParkeerHeer willen gebruiken. Een aantal eerder gestelde ontwerpvragen hebben te maken met de wensen van de bestuurder. Deze beantwoorden we hier stuk voor stuk.

5.6.1 Q10: Hoe kan de foutgevoeligheid worden verminderd?

Door de eerder genomen ontwerpbeslissing om niet te reserveren, en wel te dirigeren, zal de ParkeerHeer een zekere mate van foutgevoeligheid hebben (paragraaf 5.3). Daarom stellen we voor om een foutmarge te hanteren:

R1-f: De ParkeerHeer zal een foutmarge hanteren omtrent het geschatte aantal vrije parkeerplaatsen in de toekomst.

Omdat een toekomstvoorspelling nooit 100% perfect kan zijn, zal er een zekere foutmarge gehanteerd moeten worden. Het doel is om het aantal weigeringen te verminderen, en daarom zal bij een bepaald minimum al gesteld worden dat de parkeergarage bezet is. Het volgende scenario schetst dit in het kort:

Plien komt over 30 minuten aan bij haar eindbestemming, dichtbij parkeergarage A. De schatting is dat A over 30 minuten nog 1 vrije plaats heeft. De ParkeerHeer stuurt Plien niet naar A, omdat de kans te groot is dat de schatting fout is.

5.6.2 Q11: Welke eisen stelt de bestuurder aan het toewijzen?

R1-g: Wanneer er voor een auto geen parkeergarage gevonden kan worden zal de ParkeerHeer deze naar haar eindbestemming dirigeren.

Het kan altijd voorkomen dat er geen parkeergarages met vrije plaatsen in de buurt van eindbestemming zijn. In dit geval moet de bestuurder naar haar eindbestemming gedirigeerd worden. Zo kan ze zelf een vrije parkeerplaats zoeken, bijvoorbeeld bij een parkeergarage die niet is aangesloten op de ParkeerHeer.

R1-h: Aangesloten bestuurders zullen worden toegewezen aan parkeergarages die zo dicht mogelijk bij hun eindbestemming liggen.

We gaan er van uit dat hoe minder ver de bestuurder hoeft te lopen, hoe blijer ze is.

R1-i: De afstand tussen eindbestemming en toegewezen parkeergarage mag niet groter zijn dan een voorafbepaald minimum.

Vanaf een bepaalde loopafstand is het niet meer nuttig om een auto toe te wijzen aan een parkeerplaats. Deze eis voorkomt dat we bestuurders sturen naar een vrije parkeerplaats in Utrecht terwijl de eindbestemming in Groningen ligt.

R1-j: Er zullen niet meer auto's worden toegewezen aan een parkeergarage dan er vrije plaatsen zijn.

Heeft een parkeergarage op een gegeven moment in de toekomst 10 vrije plaatsen, dan mogen er maximaal 10 auto's met dezelelfde aankomsttijd heen worden gestuurd.

5.6.3 Q3: Hoe snel moeten de auto's aan de parkeergarages worden toegewezen?

R1-k: De ParkeerHeer zal een auto aan een parkeergarage hebben toegewezen voordat de afstand van de auto tot haar eindbestemming kleiner is dan een vooraf bepaald minimum.

De bestuurder wil vanaf een bepaald punt een parkeerplaats toegewezen krijgen. Voorbij dit punt wordt aangenomen dat de bestuurder zelf een beslissing neemt en de ParkeerHeer gaat negeren. Ook kan het voorkomen dat de bestuurder een afslag heeft gemist waardoor de toegewezen parkeerplaats niet meer het meest efficiënt is. Omdat de infrastructuur per stad verschilt moet dit punt per eindbestemming in te stellen zijn. De rekeneenheid voor dit punt wordt vastgesteld op geschatte reistijd. Dit punt noemen we vanaf nu ook wel de **Toewijs-grens**.

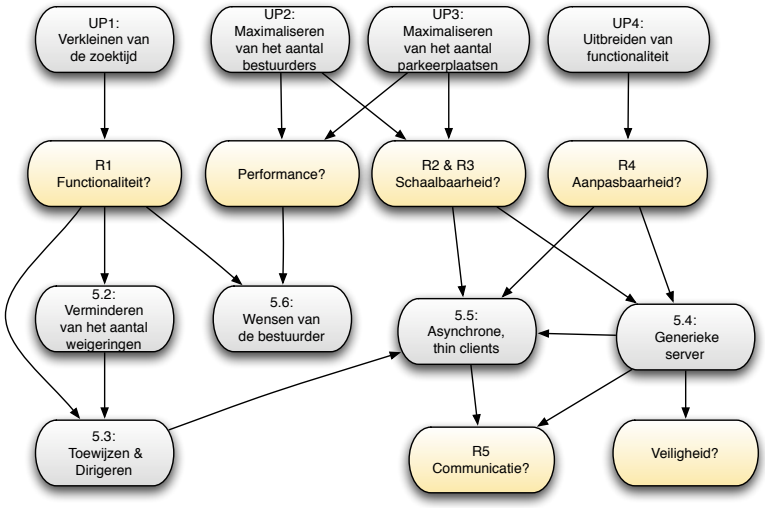
5.7 Samenvattend

We hebben nu de eerste ontwerpbeslissingen genomen, die resulteren in een reeks van eisen en een eerste opdeling van het systeem in subcomponenten. De eisen zijn nog niet volledig en compleet. Wel hebben we de belangrijkste functionele en niet-functionele componenten benoemd. Deze worden ingedeeld in de volgende categoriën:

- R1: Functionaliteit. Deze eisen en ontwerpvragen gaan over de functionaliteit van het systeem. Use cases 1 t/m 8 werken de functionele eisen verder uit.
- R2 & R3: Schaalbaarheid. Deze eisen en ontwerpvragen gaan over de manier waarop de ParkeerHeer geschaald kan worden met betrekking tot het aantal auto's (R2) en parkeerplaatsen (R3). Use case 12 verwerkt de schaalbaarheid van de ParkeerHeer.

- R4: Aanpasbaarheid. Deze eisen en ontwerpvragen hebben betrekking op de manier waarop de ParkeerHeer gaat worden ingericht, zodat ze gemakkelijk aangepast kunnen worden. Use cases 9, 10 en 11 geven inzicht in de manier waarop aanpasbaarheid van de ParkeerHeer zal worden uitgewerkt.
- R5: Communicatie. Dit is een subset van de functionele vragen en eisen, en gaan specifiek over de manier waarop de ParkeerHeer met haar clients communiceert. Use cases 5,6 en 7 verduidelijken de communicatie met betrekking tot de auto's en parkeergarages.
- Performance. Deze vragen en eisen gaan over de performance van de ParkeerHeer. De performance hangt af van meerdere zaken. We scharen de eisen met betrekking tot performance bij de functionaliteit (R1), omdat dit de performance het meest beïnvloedt. Use case 12 werkt de performance uit.
- Veiligheid: De vragen en eisen met betrekking tot de veiligheid gaan voornamelijk over de communicatie, daarom is dit onderwerp ondergebracht onder het kopje Communicatie (R5). Use cases 5,6 en 7 verwerken deels de veiligheid van de ParkeerHeer met betrekking tot de communicatie.

Ter verduidelijking en om het overzicht te kunnen behouden geven we hieronder een schematisch overzicht van de genomen ontwerpbeslissingen in dit hoofdstuk. In de appendix is de volledige lijst van ontwerpvragen en eisen opgenomen.



Figuur 12: Weergave van de genomen ontwerpbeslissingen. Een pijl tussen twee blokken geeft aan dat de oorsprong leidde tot het doel van de pijl. De bovenste vier blokken zijn de uitgangspunten. Een geel blok geeft een functionele of niet-functionele vraag weer die verder moet worden uitgediept. Zo kwam bij de ontwerpbeslissing om een generieke server te gaan gebruiken naar boven, dat veiligheid een belangrijke niet-functionele eis is.

Deel II
Elaboratie

6 Scenarios

De scenario's ofwel use cases[12] komen voort uit de functionele en niet-functionele eisen die in het eerste deel aan bod zijn gekomen. Daarnaast zijn ze met de ontwikkeling van de ParkeerHeer doorontwikkeld.

6.1 Overzicht

Figuur 13 geeft een grafische weergave van alle use cases en de actoren. De use cases worden opgedeeld in twee delen:

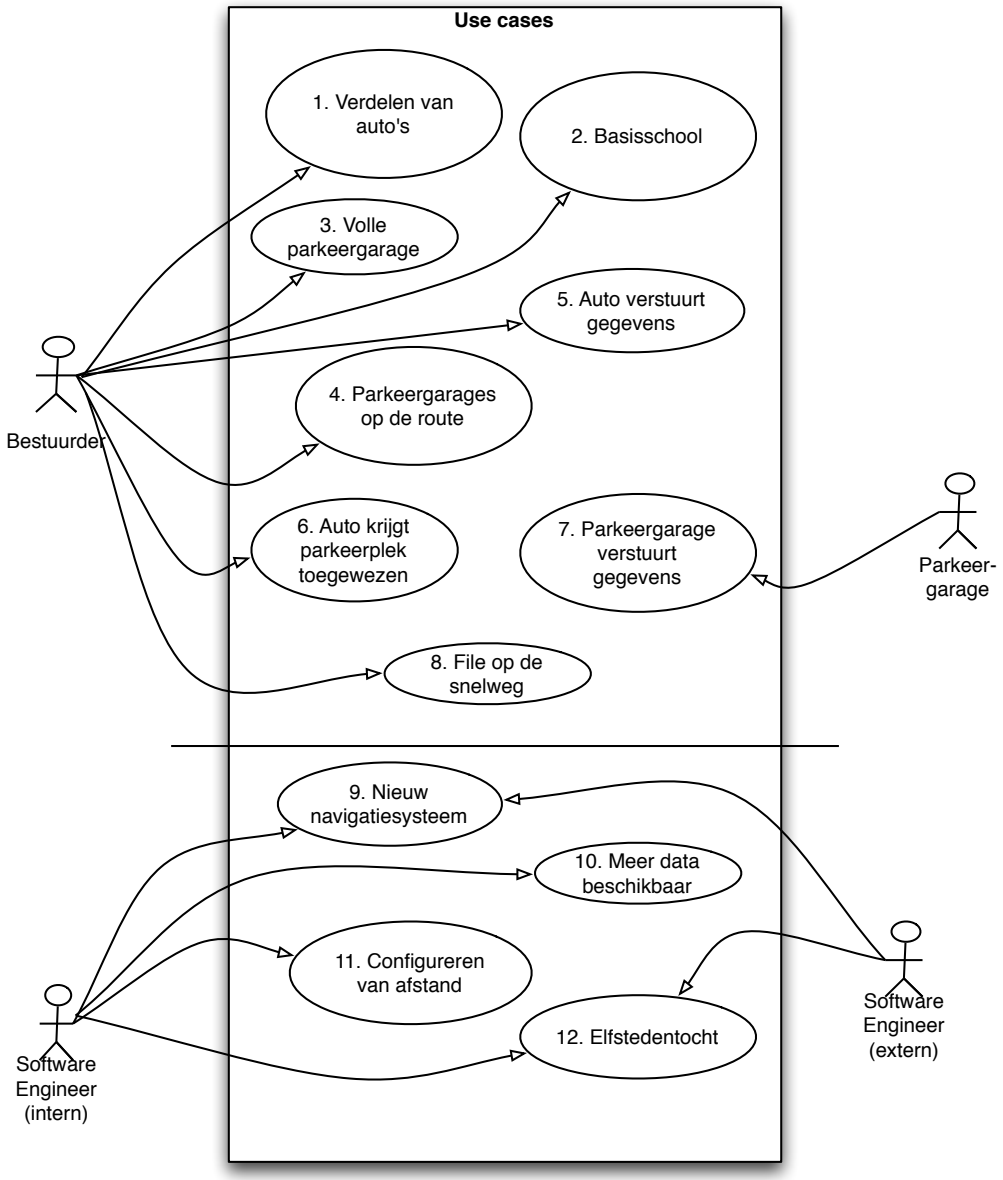
Functionele use cases

Deze use cases belichten voornamelijk de wensen en eisen van de bestuurder en beantwoorden dan ook voornamelijk de eisen van categorie R1 en R5. De Logical Architecture View besteedt de meeste aandacht aan de realisering van deze use cases. De functionele use cases staan boven de horizontale streep in figuur 13.

Niet-functionele use cases

Deze use cases zijn gemaakt om de niet-functionele eisen te kunnen testen. Anders dan de functionele use cases hebben de niet-functionele use cases geen einde. Ze definiëren niet hoe het systeem moet gaan werken, maar stellen een mogelijk niet-functioneel probleem bloot. Ze hebben dan ook geen eenduidige uitkomst. In de vier andere Architecture Views wordt aandacht besteed aan de niet-functionele eisen en wordt een antwoord op deze use cases gegeven. De niet-functionele use cases staan onder de horizontale streep in figuur 13.

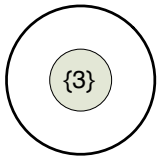
[12] De termen 'scenario' en 'use case' worden door elkaar heen gebruikt en betekenen in dit onderzoek hetzelfde.



Figuur 13: Use case diagram van de ParkeerHeer

6.1.1 Use Case Template

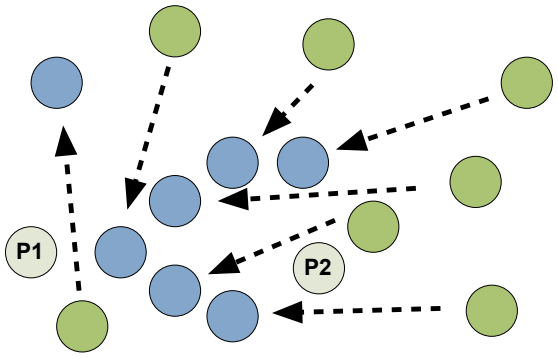
In tabel 1 wordt een template weergegeven van een use case zoals deze in dit document wordt gebruikt.

Naam	<Naam>
ID	<Uniek id>
Primair doel	<Het primaire doel van de use case>
Inceptie	<Waar in het inceptie deel worden de onderdelen van deze use cases behandeld?>
Elaboratie	<Waar in het elaboratie deel worden de onderdelen van deze use cases behandeld?>
Afbeelding	[optioneel] Afbeelding ter verduidelijking van de situatie, met de volgende symbolen:  auto A onderweg naar eindbestemming Z  parkeergarage met 3 vrije plaatsen. De cirkel eromheen geeft de maximale afstand aan tot een eindbestemming
Hoofdscenario Resultaat	<Beschrijving van de use case> <Het resultaat van de use case>
Alternatief Resultaat	<Een alternatief, wanneer een of meer voorwaarde(n) worden aangepast> <Resultaat van het alternatief>

Tabel 1: Template van een use case

6.2 Functionele Use Cases

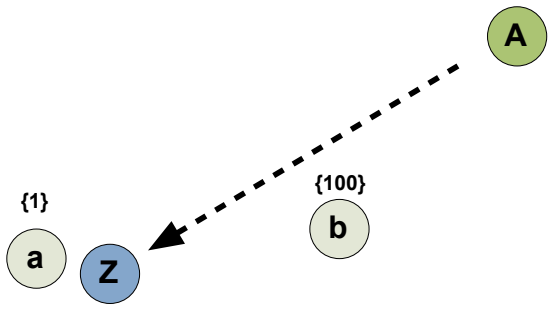
6.2.1 Verdelen van auto's

Naam	Verdelen van auto's
ID	UC 1
Primair doel	Aangeven hoe de ParkeerHeer rekening houdt met de bezetting van de parkeergarages.
Inceptie	Verkleinen van de zoektijd (5.2) Dirigeren (5.3) Wensen van de bestuurder (5.6)
Elaboratie	Logical Architecture View (7)
Afbeelding	
Hoofdscenario	7 auto's willen allemaal naar een eindbestemming in de buurt van parkeergarage $P1$ en $P2$. Parkeergarage $P1$ heeft nog 5 parkeerplaatsen over. Parkeergarage $P2$ heeft nog 4 parkeerplaatsen over. De foutmarge is 0 voor beide parkeergarages $P1$ en $P2$.
Resultaat	De auto's worden dusdanig verdeeld over $P1$ en $P2$ dat er niet meer dan 5 auto's naar $P1$ worden gestuurd en niet meer dan 4 naar $P2$
Alternatief	Het aantal vrije parkeerplaatsen van $P1$ verandert naar 2. Het aantal vrije parkeerplaatsen van $P2$ verandert naar 1.
Resultaat	De auto's worden dusdanig verdeeld over $P1$ en $P2$ dat niet meer dan 2 auto's naar $P1$ worden gestuurd en niet meer dan 1 naar $P2$. De overige auto's worden direct naar hun eindbestemming gedirigeerd.

6.2.2 Basisschool

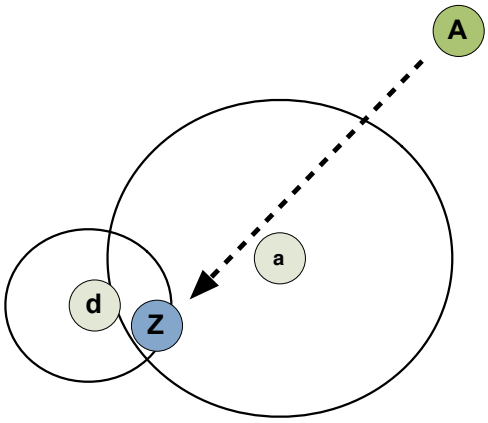
Naam	Basisschool
ID	UC 2
Primair doel	Het verduidelijken van het voorspellend vermogen.
Inceptie	Vooronderzoek: Voorspellend vermogen (4.1.1) Vereisten: Generieke server (5.4)
Elaboratie	Logical Architecture View: Verzamelen van data (7.2)
Hoofdscenario	Parkeergarage $P1$ staat naast basisschool *De Werkschuit* in Zwolle. Parkeergarage $P2$ staat ook naast *De Werkschuit*, maar verder dan $P1$. Elke doordeweekse ochtend rond half 9 is parkeergarage P1 helemaal vol, omdat de ouders hun kinderen wegbrengen naar school. Het is zo'n doordeweekse ochtend omstreeks 8 uur. Bestuurder A heeft een eindbestemming dichtbij *de Werkschuit*. De eindbestemming van A ligt dichter bij $P1$ dan $P2$.
Resultaat	De ParkeerHeer stuurt A naar parkeergarage $P2$.
Alternatief	Het is weekend, of na half 9 's ochtends doordeweeks.
Resultaat	De ParkeerHeer stuurt A naar Parkeergarage $P1$.

6.2.3 Volle parkeergarage

Naam	Volle parkeergarage
ID	UC 3
Primaire doelen	Werking van de foutmarge van de voorspellingen verduidelijken. Uiteenzetten wat er gebeurt als er geen geschikte parkeerplaats kan worden gevonden. Aangeven dat de auto's met een bepaalde rangorde worden toegewezen aan parkeergarages.
Inceptie	Vooronderzoek: Voorspellend vermogen (4.1.1) Vereisten: Verkleinen van de zoektijd (5.2) Vereisten: Generieke server (5.4) Vereisten: Eisen van de bestuurder (5.6)
Elaboratie	Logical Architecture View (7)
Afbeelding	

Naam	Volle parkeergarage (Vervolg)
Hoofdscenario	Auto A wil naar eindbestemming Z.
	Parkeergarage a ligt dichterbij de eindbestemming dan parkeerplaats b.
	Auto A is 15 min. verwijderd van eindbestemming Z.
	Parkeergarage a heeft over 15 minuten nog 1 vrije parkeerplek.
	Parkeergarage b heeft over 15 minuten nog 100 vrije parkeerplekken.
	De foutmarge van parkeerplaats a is 5.
	De foutmarge van parkeerplaats b is 5.
Resultaat	Auto A wordt naar parkeergarage b gestuurd.
Alternatief 1	Parkeergarage b heeft over 15 minuten nog 4 vrije parkeerplaatsen.
Resultaat	Auto A wordt naar Z gestuurd.
Alternatief 2	Parkeergarage a heeft over 15 minuten nog 6 vrije parkeerplekken.
	Er is een tweede auto A' die ook naar parkeergarage a wil.
Resultaat	A' wordt naar haar eigen eindbestemming gestuurd.
	A wordt naar Z gestuurd

6.2.4 Prioritisering van auto's

Naam	Prioritering van auto's
ID	UC 4
Primair doel	Prioritering van auto's aangeven.
Inceptie	Vereisten: Eisen van de bestuurder (5.6)
Elaboratie	Logical Architecture View (7)
	
Hoofdscenario	Auto A wil naar eindbestemming Z. Parkeergarage a ligt verder van eindbestemming Z dan parkeergarage d. Parkeergarage a ligt dichterbij auto A dan parkeergarage d. Parkeergarage a heeft nog genoeg parkeerplaatsen en ligt op loopafstand van Z. Parkeergarage d heeft nog genoeg parkeerplaatsen en ligt op loopafstand van Z.
Resultaat	Auto A wordt naar parkeergarage d gestuurd.
Alternatief 1 Resultaat	a en d liggen niet op loopafstand van Z. Auto A wordt naar Z gestuurd.

6.2.5 Auto verstuurt gegevens

Naam	Auto verstuurt gegevens
ID	UC 5
Primair doel	Aangeven op welke manier de gegevensoverdracht plaatsvindt van auto naar de ParkeerHeer.
Inceptie	Samenwerken met navigatiesoftware (5.4) Wensen van de bestuurder (5.6) Performance (5.6) Veiligheid (5.4)
Elaboratie	Logical Architecture View: Periodiek een oplossing uitrekenen (7.1) Logical Architecture View: Verzamelen van data (7.2) Process Architecture View (9) Physical Architecture View (10)
Hoofdscenario	Een bestuurder maakt gebruik van de ParkeerHeer services via haar navigatiesysteem. De bestuurder zet haar navigatiesysteem aan. De bestuurder selecteert een eindbestemming. De bestuurder zet de Parkeeroptie aan en voert haar gebruikersnaam en wachtwoord in.
Resultaat	Het navigatiesysteem zendt periodiek, automatisch, de benodigde gegevens naar de ParkeerHeer. De bestuurder hoeft hierna geen handelingen meer te verrichten.
Alternatief	De ingevoerde gebruikersnaam-wachtwoord combinatie is niet correct.
Resultaat	De bestuurder krijgt hier een melding van en voert opnieuw gebruikersnaam en wachtwoord in.

6.2.6 Auto krijgt parkeergarage toegewezen

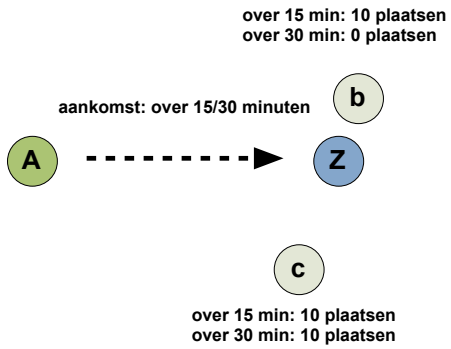
Naam	Auto krijgt parkeergarage toegewezen
ID	UC 6
Primaire doelen	Uiteenzetten hoe een auto een vrije parkeerplek krijgt toegewezen. Aangeven dat de bestuurder binnen een bepaalde tijd een parkeergarage toegewezen wil hebben gekregen.
Inceptie	Dirigeren (5.3) Performance (5.6)
Elaboratie	Physical Architecture View (10) Logical Architecture View (7)
Hoofdscenario	Een auto maakt gebruik van de ParkeerHeer services via haar navigatiesysteem (UC 5). De bestuurder heeft de parkeeroptie ingeschakeld en is 4 min. van haar eindbestemming verwijderd. De toewijs-grens ligt op 5 minuten van de eindbestemming van de bestuurder. De bestuurder heeft reeds een parkeergarage toegewezen gekregen.
Resultaat	De parkeergarage verschijnt als eindbestemming op het scherm van het navigatiesysteem. De bestuurder rijdt naar de parkeergarage en parkeert daar haar auto.
Alternatief	Er is geen geschikte parkeergarage in de buurt van eindbestemming
Resultaat	De bestuurder wordt direct naar de eindbestemming gestuurd.

Naam	Auto krijgt parkeergarage toegewezen (Vervolg)
Alternatief 2	De bestuurder rijdt naar de parkeergarage, maar bij aankomst zijn alle plekken bezet.
Resultaat	De bestuurder activeert de parkeer optie opnieuw en wacht tot ze een nieuwe parkeergarage krijgt toegewezen.
Alternatief 3	De bestuurder rijdt naar de parkeergarage, maar tijdens het rijden verandert de situatie van de parkeergarage waar ze aan is toegewezen: deze blijkt toch vol te zijn.
Resultaat	De bestuurder wordt automatisch naar een andere parkeergarage gestuurd die niet vol is.
Alternatief 4	Het is gebleken dat de toewijs-grens te hoog is. Het voorspellend vermogen van de ParkeerHeer is niet voldoende om vanaf deze afstand een goede schatting te maken. Hierdoor is de kans groot dat de bestuurder bij een volle parkeergarage komt te staan.
Resultaat	De toewijs-grens wordt verkleind naar 3 minuten. De bestuurder krijgt mogelijk later een parkeerplaats toegewezen, maar voordat ze deze grens is gepasseerd.
Alternatief 5	De bestuurder wijzigt tijdens de rit haar eindbestemming.
Resultaat	De auto krijgt na verloop van tijd een nieuwe parkeergarage toegewezen.
Alternatief 6	De bestuurder heeft via het interface van het navigatiesysteem een voorkeur van parkeergarage opgegeven.
Resultaat	Mits deze parkeergarage genoeg vrije parkeerplaatsen heeft, wordt de bestuurder hierheen gestuurd.

6.2.7 Parkeergarage verstuurt gegevens

Naam	Parkeergarage verstuurt gegevens
ID	UC 7
Primair doel	Uiteenzetten op welke manier parkeergarages hun gegevens versturen.
Inceptie	Samenwerken met parkeergaragesystemen (5.4) Performance (5.6) Veiligheid (5.4)
Elaboratie	Logical Architecture View: Periodiek een oplossing uitrekenen (7.1) Logical Architecture View: Verzamelen van data (7.2) Process Architecture View (9) Physical Architecture View (10)
Hoofdscenario	Een parkeergarage is aangesloten op de ParkeerHeer. De parkeergarage heeft een unieke gebruikersnaam en wachtwoord gekregen. De parkeergarage heeft meer of minder vrije parkeerplaatsen dan een moment eerder.
Resultaat	De software van de parkeergarage verzendt de mutatie automatisch, inclusief gebruikersnaam en wachtwoord.
Alternatief 1	De ParkeerHeer houdt niet elektronisch bij hoeveel parkeerplaatsen er beschikbaar zijn.
Resultaat	De parkeergarage wordt niet aangesloten op de ParkeerHeer.
Alternatief 2	De verzonden gebruikersnaam en wachtwoord combinatie is niet correct. De gegevens van de parkeergarage worden niet opgeslagen.
Resultaat	De parkeergarage krijgt hier een melding van.

6.2.8 File op de snelweg

Naam	File op de snelweg
ID	UC 8
Primair doel	Aangeven op welke manier de informatie van de navigatie-software wordt gebruikt om een parkeerplaats te vinden.
Inceptie	Samenwerken met navigatiesoftware (5.4) Verkleinen van de zoektijd (5.2) Dirigeren (5.3) Wensen van de bestuurder (5.6) Voorspellend vermogen (5.4)
Elaboratie	Logical Architecture View (7) Process Architecture View (9) Physical Architecture View (10)
	

Naam	File op de snelweg (Vervolg)
Hoofdscenario	Auto A rijdt op de snelweg op weg naar eindbestemming Z. Parkeerplaatsen b en c liggen dichtbij Z. b ligt dichter bij Z dan c. b heeft over 15 minuten nog vrije plaatsen maar zit over 30 minuten vol. c heeft het komende uur nog voldoende vrije plaatsen. Auto A komt over 15 minuten aan bij Z. De foutmarge van b en c is 0.
Resultaat	De ParkeerHeer stuurt A naar parkeergarage b.
Alternatief 1	Het navigatiesysteem van A maakt gebruik van een extra functionaliteit waarmee het files kan ontwijken. Er staat een file op de snelweg waardoor de aankomsttijd 15 minuten wordt verlaat.
Resultaat	Het navigatiesysteem geeft de aangepaste aankomsttijd door aan de ParkeerHeer. De ParkeerHeer wijst parkeergarage c toe aan de auto. De auto neemt de eerste afslag en ontwijkt hiermee de files.

6.3 Niet-Functionele Use Cases

6.3.1 Nieuw systeem aansluiten

Naam	Nieuw systeem aansluiten
ID	UC 9
Primair doel	Uitbreidbaarheid van communicatiekanalen aantonen.
Inceptie	Generieke Server (5.4)
Elaboratie	Development Architecture View (8)
Hoofdscenario	Er is een nieuw type navigatiesysteem op de markt: De *acigoL*. Het ontwikkelteam van de *acigoL* wil graag gebruik maken van de ParkeerHeer om zo de gebruikers van extra functionaliteit te voorzien.
Alternatief	De *acigoL* is geen navigatiesoftware, maar software die informatie over parkeerplaatsen geeft.

6.3.2 Meer data beschikbaar

Naam	Meer data beschikbaar
ID	UC 10
Primair doel	Uitbreidbaarheid in functionaliteit aantonen.
Inceptie	Generieke Server (5.4)
Elaboratie	Development Architecture View (8)
Hoofdscenario	Een aantal parkeergarages brengt naast reguliere parkeerplaatsen, ook de gehandicapten parkeerplaatsen beschikbaar. Deze plaatsen mogen slechts door een beperkte groep bestuurders bezet worden.

6.3.3 Configureren van instellingen

Naam	Configureren van instellingen
ID	UC 11
Primair doel	Configuratiemogelijkheden van de parkeerheer aantonen.
Inceptie	-
Elaboratie	Development Architecture View (8)
Hoofdscenario	De minimale afstand van parkeergarage tot eindbestemming moet in Utrecht voor een bepaalde tijd worden aangepast van 500 meter naar 1000 meter.

6.3.4 Elfstedentocht

Naam	Elfstedentocht
ID	UC 12
Primair doel	Performance en schaalbaarheid eisen blootleggen.
Inceptie	Communicatie client en server (5.5)
Elaboratie	Process Architecture View (9) Physical Architecture View (10)
Hoofdscenario	Na meer dan 30 jaar is het eindelijk zover: De Friese elfstedentocht kan weer worden gereden. Heel Nederland rukt uit om in de elf steden de schaatsers aan te moedigen. Er zijn speciale parkeerplaatsen ingeschakeld. De status van deze parkeerplaatsen (bezet/niet bezet) wordt door een nieuw systeem elektronisch bijgehouden en verzonden naar de ParkeerHeer Er moet rekening worden gehouden met grote drukte op de weg, 50.000 auto's die, verdeeld over de elf steden, een parkeerplaats zoeken.

7 Logical Architecture

Het doel van de Logical Architecture View is het realiseren van de functionele eisen van de ParkeerHeer. Als eerste wordt de communicatie tussen de ParkeerHeer en de clients uitgewerkt. Vervolgens wordt er een model gepresenteerd waarmee een oplossing kan worden berekend. Als laatste wordt een deel van dit model, de kostenfunctie, nader bekeken.

7.1 Het moment dat er een oplossing wordt berekend

Zoals besloten in R2 en R3 moet de ParkeerHeer rekening houden met een groot aantal auto's en parkeergarages: De oplossing moet berekend worden op basis van alle gegevens van de auto's en parkeergarages (R1-d). Maar deze gegevens veranderen voortdurend: De schatting van het aantal vrije plekken van de parkeergarages van een bepaald moment zal steeds nauwkeuriger worden omdat dit moment steeds dichter bij zal komen. De auto's veranderen van locatie, komen dichter bij hun eindbestemming. De locatie en verwachtte aankomsttijd zal dan ook steeds nauwkeuriger worden en worden bijgesteld. We stellen hierbij de vraag:

Q17: Op welk moment moet er een oplossing worden berekend?

Bij het beantwoorden van deze vraag worden de volgende criteria gehanteerd:

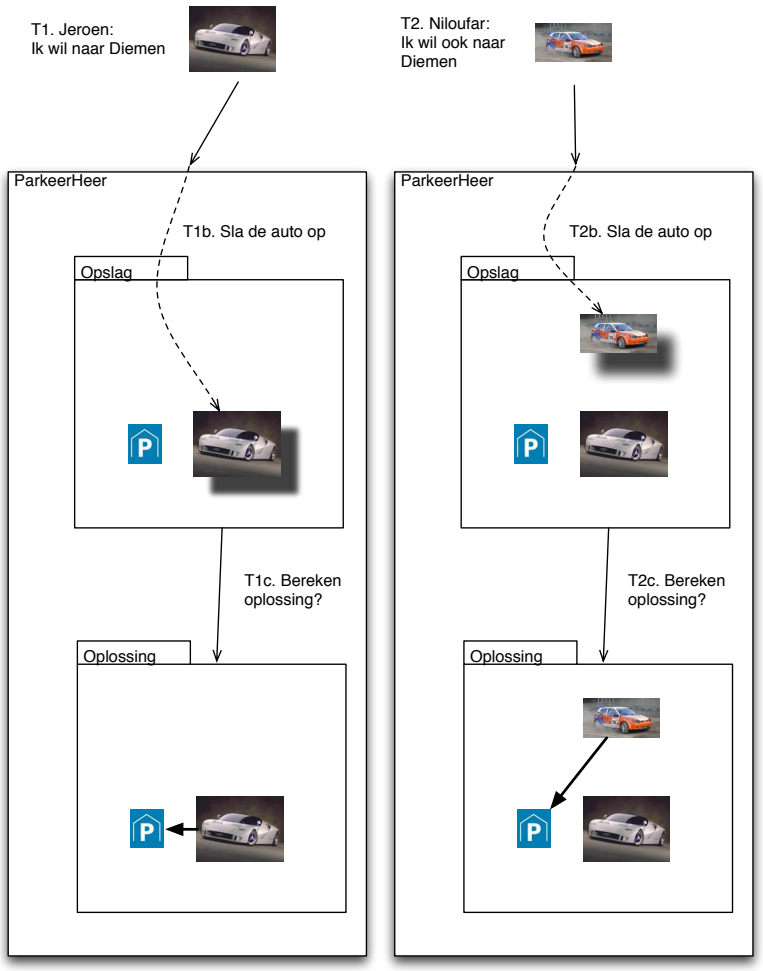
- Performance: In hoeverre worden er overbodige berekeningen uitgevoerd?
- Aanpasbaarheid: In hoeverre kan deze optie worden aangepast, is het flexibel?
- Gebruiksvriendelijkheid: In hoeverre is deze optie gebruiksvriendelijk voor de eindgebruiker?

Optie: Meteen oplossing uitrekenen

Bij de eerste optie rekenen we meteen een oplossing uit: Het navigatiesysteem stuurt een verzoek naar de ParkeerHeer en krijgt direct een parkeergarage toegewezen. Deze optie heeft de volgende eigenschappen:

- Gebruiksvriendelijk: De bestuurder krijgt direct antwoord.
- Na elk verzoek moet er een oplossing worden uitgerekend

Door het tweede punt is dit geen goede oplossing. We verduidelijken dit met een voorbeeld, zie ook figuur 14:



Figuur 14: Dynamisch of statisch?

In bovenstaande afbeelding willen Jeroen en Niloufar[13] alle twee naar Diemen. Er is één parkeergarage met precies één vrije parkeerplaats en foutmarge 0. Op tijd t_1 meldt Jeroen zich aan en geeft door aan de ParkeerHeer dat hij naar Diemen wil (T1 op de afbeelding). De ParkeerHeer ontvangt het bericht en slaat Jeroen op (T1b). Vervolgens rekent de ParkeerHeer een oplossing uit.

Precies op dat moment meldt Niloufar zich aan: ze wil ook naar Diemen, ze heeft nagenoeg dezelfde eindbestemming als Jeroen. Haar eindbestemming is alleen iets dichterbij de parkeergarage. De ParkeerHeer slaat haar gegevens op, en rekent een nieuwe oplossing uit. Deze oplossing stelt dat niet Jeroen, maar Niloufar naar de parkeergarage moet worden gestuurd. Op dit moment heeft Jeroen net te horen gekregen dat hij naar de parkeergarage kan rijden (de oplossing van T1c). Een fractie van een seconde later is de oplossing van Niloufar

[13]Spreek uit: Ni-loe-far

uitgerekend. Jeroen mag dus eigenlijk niet meer naar de parkeergarage rijden, die is namelijk voor Niloufar bedoeld. We merken hier een aantal dingen op:

- Deze situatie schendt R1-j door twee bestuurders naar 1 vrije parkeerplaats te sturen
- De ParkeerHeer doet een overbodige berekening, namelijk de eerste.
- Jeroen zal op een later tijdstip te horen krijgen dat hij toch niet naar de parkeergarage mag rijden.

Trekken we deze situatie door voor Corry, Els, Hans, en 19.995 meer bestuurders[14], dan kunnen we het volgende stellen:

- De ParkeerHeer doet een hoop overbodige berekeningen
- Door de overbodige berekeningen wordt er erg veel gewisseld: Bestuurders worden dan wel, en dan weer niet naar een bepaalde garage gestuurd.
- Er is een redelijk grote kans dat R1-j meerdere malen geschonden wordt.

Optie: Periodiek oplossing uitrekenen

In deze optie reduceren we het aantal overbodige berekeningen door periodiek een oplossing uit te rekenen, bijvoorbeeld elke 5 minuten. In afbeelding 14 moeten daarbij de pijlen T1c en T2c worden weggehaald. Deze oplossing heeft de volgende eigenschappen:

- Hoe korter de periode, hoe sneller de bestuurder een antwoord krijgt.
- Hoe korter de periode, hoe groter het aantal overbodige berekeningen wordt.

Daarnaast is deze optie goed aanpasbaar omdat het systeem dusdanig kan worden ingericht dat de lengte van een periode variabel is.

Ontwerpbeslissing: Periodiek oplossingen uitrekenen

We kiezen voor de laatste optie. Vanaf nu noemen we een periode ook wel een *epoch*. Let op dat we de eerste optie (meteen een oplossing uitrekenen) kunnen benaderen door de lengte van een epoch heel erg klein te maken. In tabel 7.1 zetten we de overwegingen nogmaals tegenover elkaar.

Criterium	Meteen	Periodiek
# Overbodige Berekeningen (groter is slechter)	Groot	Kleiner
Aanpasbaarheid	Slecht	Beter
# Schommelingen? (groter is slechter)	Groot	Kleiner

[14] Zie R2 en R3

We moeten hierbij noteren dat het probleem van de schommelingen niet is opgelost. Daarom stellen we de volgende vragen:

Q18: Hoe kan het aantal schommelingen worden verminderd? (Onbeantwoord, zie Discussie (12))

Q19: Hoe lang moet een periode duren? (Onbeantwoord, zie Discussie (12))

Q20: Hoe moet de data verzameld worden? (7.2)

7.2 Verzamelen van data

Nu besloten is dat er periodiek een oplossing uitgerekend gaat worden, moet worden bedacht hoe de benodigde data gaat worden verzameld. We behandelen hier de volgende vraag, waarbij de oplossing eigenlijk direct voortvloeit uit eerder genomen beslissingen:

Q20: Hoe moet de data verzameld worden?

7.2.1 Voorspellen van de toekomst

Aan het begin van elke epoch wordt een nieuwe berekening gemaakt met behulp van de meest recente data. Maar voor welk tijdstip geldt deze nieuwe oplossing? We stellen dat we aan het begin van Epoch n oplossingen gaan berekenen voor j epochs in de toekomst, dus voor epochs $e_{n+1}, \ldots, e_{n+j}$. e_{n+1} berekenen we met behulp van de data die we hebben van voorgaande epochs.

In epoch e_n berekenen we de oplossing voor epoch e_{n+1} als volgt:

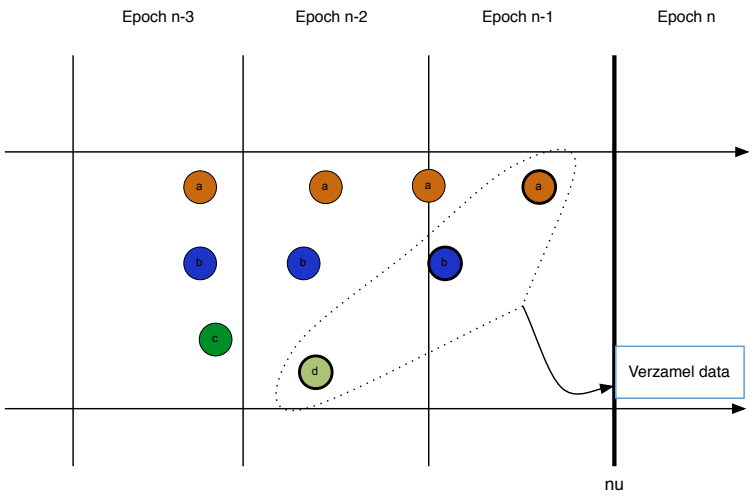
1. Haal alle auto's op die in epoch e_{n+1} aankomen.
2. Schat voor alle parkeergarages hoeveel vrije plaatsen ze zullen hebben in epoch e_{n+1}

Op deze manier worden de auto's en de status van de parkeergarages gerangschikt per epoch: Elke epoch heeft haar eigen set van auto's en parkeergarages, en dus ook een eigen oplossing.

7.2.2 Ophalen van de auto's die in epoch e_{n+1} arriveren

De auto's zullen het tijdstip dat ze arriveren moeten doorgeven. Dit tijdstip zal voortdurend veranderen, door omleidingen, files, of andere onvoorziene omstandigheden. Daarom zal dit tijdstip op regelmatig terugkerende intervallen opnieuw moeten worden gestuurd. We stellen dat de ParkeerHeer alleen de laatst opgestuurde waarde meeneemt in de berekening. Heeft een auto na een bepaalde tijd helemaal niets meer gestuurd, dan zal de auto uit het systeem verwijderd worden. Het kan zijn dat de bestuurder haar navigatiesysteem heeft uitgezet, of reeds een parkeerplek heeft gevonden. Samenvattend kunnen we stellen dat de data van de auto's als volgt wordt opgehaald (zie ook figuur 15):

Haal de meest recente data op van alle auto's die niet ouder is dan w epochs.



Figuur 15: Hoe de data van de auto's wordt opgehaald. Er zijn vier auto's (a, b, c en d), die regelmatig hun status updaten. Elke cirkel stelt zo'n status-update voor. Auto c heeft voor het laatst iets van zich laten horen in epoch $n-3$. Dit is te lang geleden, daarom gaan we ervan uit dat c niet meer meedoet. Van auto's a, b en d halen we de meest recente status-update op.

7.2.3 Schatten van aantal parkeerplaatsen per garage in epoch e_{n+1}

Er kunnen allerlei toekomstvoorspellende modellen gemaakt worden waarmee het aantal parkeerplaatsen in de toekomst kan worden geschat. Dit laten we in deze fase nog even achterwege, omdat dit een erg complex probleem is en er reeds systemen zijn die dit doen: De PRIS informatiesystemen. Aanvankelijk zullen we de ParkeerHeer hier op aansluiten. Hierdoor kan de aandacht meer worden gericht op andere onderdelen. We gaan er dus van uit dat de data omtrent het aantal vrije parkeerplaatsen wordt aangeleverd door de parkeergarages.

7.3 Model

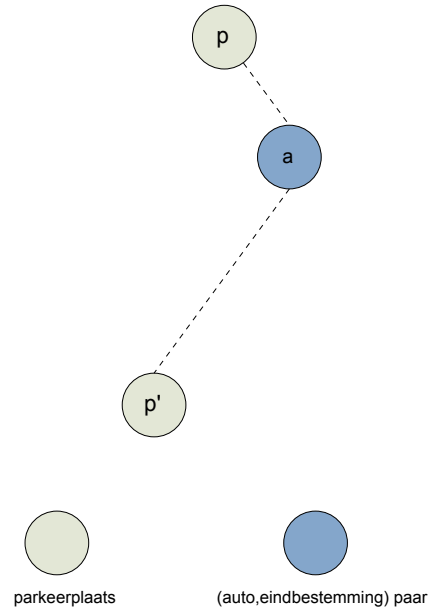
Het doel van dit hoofdstuk is om het concrete probleem van het toewijzen van auto's aan parkeerplaatsen om te zetten in een abstract model waar een computer mee kan rekenen. Als eerste worden de auto's, eindbestemmingen en parkeergarages omgezet naar een graaf en wordt er een definitie gegeven van een bezette parkeerplek. Dit leidt tot een optimalisatieprobleem dat met bestaande software pakketten kan worden opgelost. Vervolgens definiëren we een kostenfunctie, die aangeeft in hoeverre een auto ergens mag of wil parkeren. De uitkomst van de kostenfunctie resulteert in een representatie van de eerder gedefinieerde graaf. Vervolgens stellen we een aantal voorwaarden aan deze kostenfunctie, en bekijken we op welke manier deze kostenfunctie het optimalisatieprobleem beïnvloedt.

7.3.1 De basis

We stellen ons een situatie voor van rijdende auto's. Deze auto's willen allemaal parkeren in de buurt van hun eindbestemming. We maken een ongerichte gewogen graaf G van de situatie *op één specifiek moment*. De graaf heeft twee type knopen:

- P **knopen**: Waar P staat voor een parkeergarage. Parkeergarages worden gerepresenteerd als een verzameling parkeerplekken met dezelfde locatie. In dit model worden alleen de vrije parkeerplaatsen meegenomen. Dit noteren we als $P = \{p_0, \ldots, p_m\}$, dus P is de set van vrije parkeerplaatsen $p_0, \ldots, p_m$.
- (A, E) **knopen**: Waar A staat voor auto en E voor eindbestemming. Deze knoop representeert dan ook één auto en haar bijbehorende eindbestemming. Elke (A, E) knoop is direct verbonden met *elke* parkeerplek. De eindbestemming van een auto a noteren we als $dest_a$. De verzameling knopen noteren we als $A = \{a_0, \ldots, a_n\}$, waarbij $a_i = (\text{auto } i, dest_i)$.

De volgende figuur geeft een voorbeeld van G, met één auto en twee vrije parkeerplaatsen:



Figuur 16: Voorbeeld van graaf G met één auto en twee vrije parkeerplaatsen. De afstand tussen de knopen geeft de relatieve reisduur tussen de parkeerplaatsen en eindbestemmingen aan. Het is dus het gunstigste om naar p te rijden.

De knopen van de graaf worden aanvankelijk zo gepositioneerd dat de afstand tussen de knopen overeenkomt met de relatieve reisduur van de parkeerplaatsen (voor de P knopen) en de eindbestemmingen (voor de (A, E) knopen). Later zullen we zien dat de afstand niet de enige variabele is die moet worden meegewogen en zullen we de graaf hierop moeten aanpassen. In de loop van dit hoofdstuk zal deze graaf dan ook steeds meer gaan afwijken van de fysieke positionering van de parkeerplaatsen en eindbestemmingen. Hiermee geven we gehoor aan R4 en Q6: (*Welke delen van de ParkeerHeer kunnen worden aangepast?*)

x_{pa} wordt gebruikt om aan te geven of een auto aan een parkeerplaats is toegewezen:

$$x_{pa} = \begin{cases} 1, & \text{als auto } a \text{ is toegewezen aan parkeerplaats } p, \\ 0, & \text{anders.} \end{cases}$$

In het hiervoor genoemd voorbeeld kunnen we stellen dat de beste oplossing is om auto a naar p te sturen, dus $(x_{pa} = 1 \ \wedge \ x_{p'a} = 0)$.

7.3.2 Oplossingsruimte

Een situatie waarin elke x een waarde heeft gekregen noemen we een configuratie. Het eerdergenoemde voorbeeld in figuur 16 heeft vier mogelijke configuraties. Ze zijn afgebeeld in tabel 2:

	Conf. 1	Conf. 2	Conf. 3	Conf. 4
x_{pa}	0	1	0	1
$x_{p'a}$	0	0	1	1

Tabel 2: Mogelijke configuraties horende bij figuur 16.

Het valt hier direct op dat sommige configuraties geen oplossingen zijn. Zo kan auto a niet naar twee parkeerplaatsen tegelijk worden gestuurd. Configuratie 4 is dus geen geldige oplossing. Configuratie 2 en 3 zijn dat wel. De eerste configuratie is een bijzonder geval waar we later op terugkomen. De set van alle oplossingen noemen we de oplossingsruimte. We schrijven $S = \{s_0, \ldots, s_k\}$ voor de totale set S van alle mogelijke configuraties $\{s_0, \ldots, s_k\}$. W is de subset van C waarvoor geldt, dat alle configuraties in W geldige oplossingen zijn:

$$W = \{s \in S \mid \text{s is een geldige oplossing}\}$$

Voor de volledigheid noteren we de volgende eisen:

R1-l: Een auto zal naar precies 1 parkeerplaats worden gestuurd, of direct naar eindbestemming

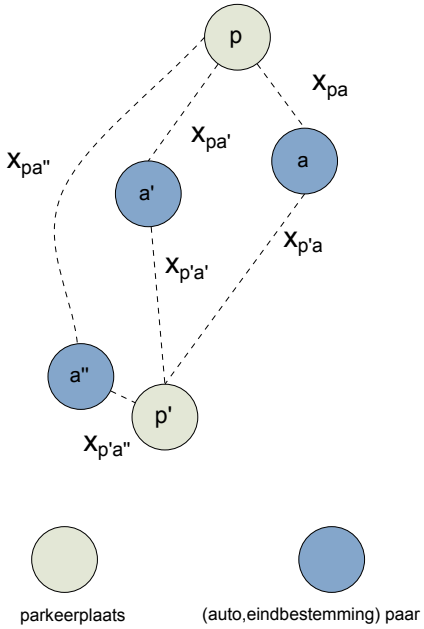
Deze eis volgt uit bovenstaande en uit het feit dat dit voorkomt dat er auto's worden geweigerd bij parkeergarages omdat ze allemaal naar dezelfde parkeergarage zijn gestuurd. Dit is dus een verdere uitdieping van R1-j. Wanneer de auto aan 0 parkeerplaatsen wordt toegewezen, zal deze naar de eindbestemming rijden zoals bepaald in R1-g.

R1-m: Een parkeerplaats zal door 0 of 1 auto's worden bezet

Ook hiervoor geldt dat dit uit bovenstaande volgt, en dat het een verdere uitdieping is van R1-j.

7.3.3 Restricties en ongeldige configuraties

Om de grenzen van de oplossingsruimte verder te definiëren, breiden het voorbeeld uit met twee extra auto's en zetten we de toewijs-functie bij de kanten van de graaf, zie figuur 17:



Figuur 17: Drie auto's en twee vrije parkeerplaatsen. Eén auto kan niet worden toegewezen aan een parkeerplaats. Maar welke?

Deze grafiek zetten we om naar een tabel, om een beter beeld te krijgen van de situatie (Tabel 3).

	Auto a	Auto a'	Auto a''
Parkeerplaats p	x_{pa}	$x_{pa'}$	$x_{pa''}$
Parkeerplaats p'	$x_{p'a}$	$x_{p'a'}$	$x_{p'a''}$

Tabel 3: Toewijs-functies per kant, horende bij figuur 17 in tabelvorm.

Dit voorbeeld heeft drie auto's, en maar twee parkeerplaatsen. Er zal dus minimaal één auto niet aan een parkeerplaats worden toegewezen. We voegen een speciale parkeerplaats toe aan de verzameling parkeerplaatsen: $P \cup \{p_\epsilon\}$, zodat we kunnen zeggen dat $p_\epsilon \in P$. Voor p_ϵ geldt, dat er oneindig veel auto's op mogen parkeren. R1-m geldt dus niet voor deze parkeerplaats. Deze parkeerplaats kan worden gezien als de eindbestemming van de auto, waar hij heen moet worden gestuurd indien er geen geschikte parkeerplaatsen zijn.

Uit R-l volgt:

$$[\mathbf{1}] \quad \sum_{p \in P} x_{pa} = 1, \qquad \forall a \in A$$

En uit R1-m volgt:

$$[\mathbf{2}] \quad \sum_{a \in A} x_{pa} \leq 1, \qquad \forall p \in \{P \setminus p_\epsilon\}$$

Daarnaast moet er rekening worden gehouden met de wensen van de bestuurder, zoals besloten in 5.6. Een bestuurder mag maar aan één parkeerplaats worden toegewezen, maar het kan voorkomen dat meer dan één plek geschikt is. Daarom introduceren we de set M_p, waar alle auto's in zitten die mogen parkeren bij parkeerplaats p.

[**3**] $a \in M_p \leftrightarrow \{a \text{ mag toegewezen worden aan parkeerplaats } p\}$,

$\forall p \in P,\ \forall a \in A$

Op dit punt geldt, dat M_p wordt bepaald door R1-i: De afstand tussen eindbestemming en toegewezen parkeergarage mag niet groter zijn dan een vooraf bepaalde afstand.

Wanneer een configuratie niet aan [**1**] , [**2**] of [**3**] voldoet, is het geen geldige oplossing, dus:

[**4**] $\{s \in S \ \wedge\ (\neg[1] \ \vee\ \neg[2] \ \vee\ \neg[3])\} \rightarrow \{s \notin W\}$

7.3.4 Kostenfunctie

De oplossingsruimte is nu gedefinieerd, dus het is mogelijk om de set van mogelijke oplossingen uit te rekenen. Er is altijd minimaal één oplossing: Stuur alle auto's naar parkeerplaats p_ϵ. Deze oplossing is het minst wenselijk, omdat de ParkeerHeer in dit geval helemaal niets bijdraagt aan het verkeer en haar bestuurders. Ze zouden immers zonder de applicatie ook wel naar hun eindbestemming gereden zijn.

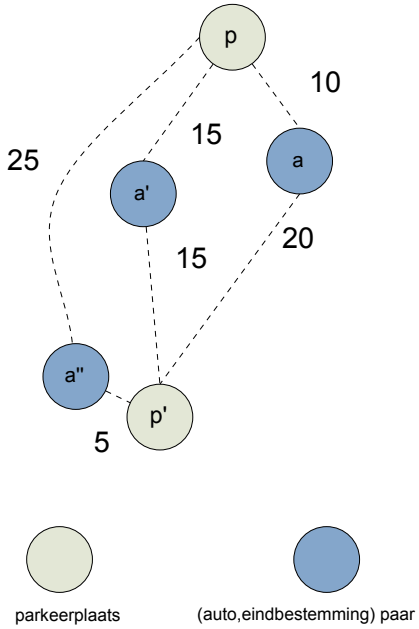
Daarom introduceren we een kostenfunctie c_{pa}. Deze functie geeft de kosten aan voor auto a om naar parkeerplaats p te rijden. Hoe groter c, hoe minder wenselijk het is voor auto a om naar vrije parkeerplaats p te rijden.

7.3.5 R1-n: De kostenfunctie bepaalt in hoeverre een auto wil parkeren bij een parkeerplaats.

Met de huidige opgestelde eisen kunnen we dit direct koppelen aan de afstand tussen een parkeerplaats en eindbestemming. De kostenfunctie wordt dan:

$$c_{pa} = \{\text{afstand}(dest_a, p)\}$$

Waarbij afstand(x, y) de reisduur is tussen x en y. Later gaan we de kostenfunctie aanpassen en zo de functionaliteit uitbreiden. In figuur 18 wordt een voorbeeld gegeven van deze kostenfunctie. De kosten staan aangegeven naast de kanten van de graaf. De afstand tussen de knopen geeft de relatieve reistijd weer tussen de eindbestemmingen en parkeerplaatsen.



Figuur 18: Drie auto's en twee vrije parkeerplaatsen. De kosten staan naast de kanten

In dit voorbeeld is de beste oplossing om a'' naar p' te sturen, en a naar p. a' valt hiermee buiten de boot, en zal direct naar haar eindbestemming gedirigeerd moeten worden.

De kosten kunnen ook in een tabel worden gezet, zoals we eerder gedaan hebben met de toewijs-functie. Er wordt hier ook een extra rij toegevoegd voor de speciale parkeerplaats p_ϵ. We stellen dat de kosten van een willekeurige auto a naar p_ϵ altijd gelijk zijn aan een constante, c_ϵ. Zie ook onderstaande tabel 4:

	Auto a	Auto a'	Auto a''
Parkeerplaats p	10	15	25
Parkeerplaats p'	20	15	5
Parkeerplaats p_ϵ	c_ϵ	c_ϵ	c_ϵ

Tabel 4: Toewijs-functie per kant, horende bij figuur 18, in tabelvorm. De laatste rij is de rij van de *speciale parkeerplaats* p_ϵ

Wanneer we in een configuratie elke x_{pa} vermenigvuldigen met de bijbehorende kosten c_{pa} kunnen we de totale kosten berekenen. Zo kunnen we zeggen dat de voor de beste mogelijke configuratie de totale kosten $(10 + c_\epsilon + 5)$ zijn. Op deze manier kunnen we de configuraties rangschikken: Hoe kleiner de totale kosten, hoe dichter de configuratie zit bij een ideale oplossing. We vertalen dit in de volgende eis:

R1-o: De auto's worden dusdanig toegewezen aan parkeerplaatsen dat de totale kosten (berekend door middel van de kostenfunctie) minimaal zijn.

In het wiskundige model wordt dit als volgt genoteerd:

Minimaliseer

$$\sum_{a \in A, p \in P} c_{pa} x_{pa}$$

Onder

$$x_{pa} \in \{0,1\}, \forall p \in P, \forall a \in A,$$

$$[\mathbf{1}] \quad \sum_{p \in P} x_{pa} = 1, \forall a \in A,$$

$$[\mathbf{2}] \quad \sum_{a \in A} x_{pa} \leq 1, \forall p \in \{P \setminus p_\epsilon\}$$

Tabel 5: De minimalisatiefunctie

Deze minimalisatiefunctie houdt rekening met **[1]** en **[2]**. Omdat er ook rekening gehouden moet worden met **[3]** en **[4]** stellen we de volgende restrictie aan de kostenfunctie:

$$[\mathbf{5}] \; a \notin M_p \rightarrow c_{ap} = \infty, \quad a \in A, \; p \in P$$

De speciale parkeerplaats p_ϵ is altijd een geschikte parkeerplaats, dus:

$$[\mathbf{6}] \; a \in M_p, \quad p = p_\epsilon, \; \forall a \in A$$

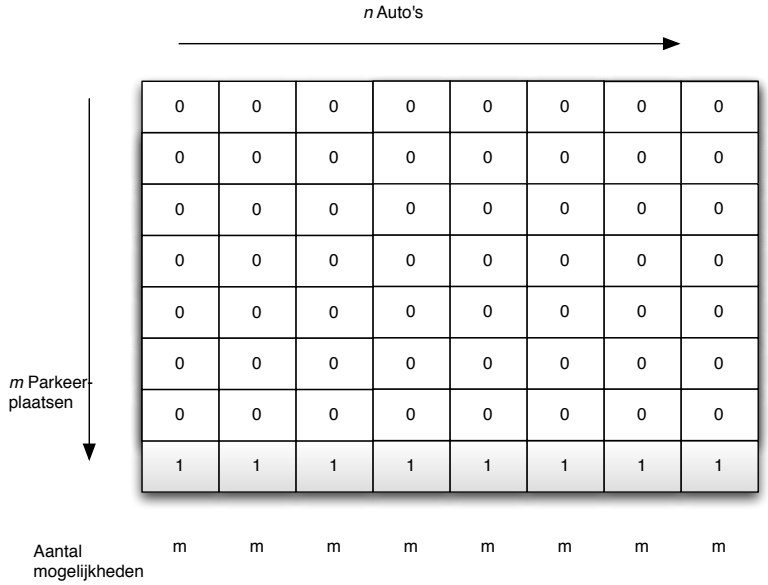
Echter, p_ϵ moet minder geschikt zijn dan elke andere parkeerplaats in M, dus:

$$[\mathbf{7}] \; a \in M_p \rightarrow c_{pa} < c_\epsilon, \quad \forall p \in P, \; \forall a \in A$$

Dit betekent dat in de kostenmatrix van tabel 4, c_ϵ groter moet zijn dan 25.

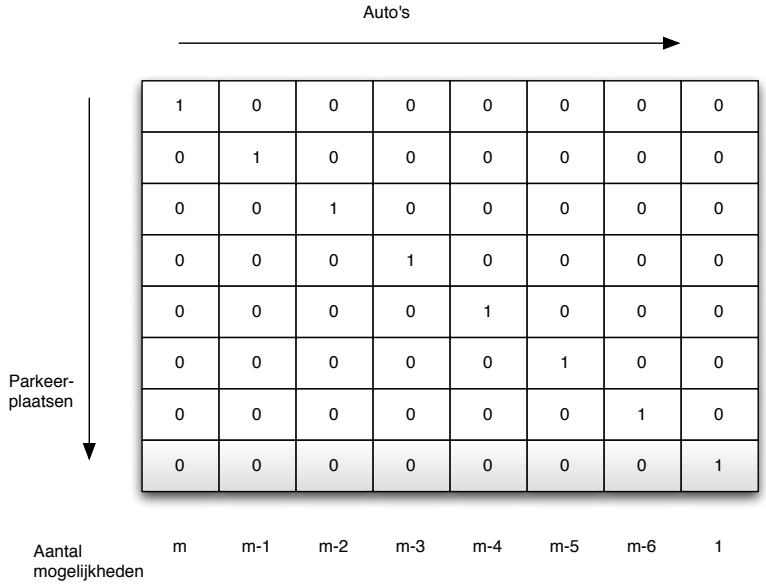
7.4 Complexiteit

Het aantal waarden in de kostenmatrix is afhankelijk van het totaal aantal parkeergarages en auto's. Bij n auto's en m parkeerplaatsen maken we een kostenmatrix van $m+1$ rijen en n kolommen, waarmee het totaal aantal variabelen op $n(m+1)$ komt. Voor elke waarde c_{pa} is er een toewijs-waarde x_{pa} die aangeeft of de auto naar de parkeerplaats wel ($x_{pa} = 1$) of niet ($x_{pa} = 0$) wordt gestuurd. Er zijn dus totaal $2^{(m+1)n}$ mogelijke configuraties. Door **[1]** en **[2]** toe te passen wordt het aantal geldige oplossingen beperkt. In het slechtste geval zijn dit er $O(m^n)$, namelijk als alle auto's naar hun eindbestemming worden gestuurd omdat er geen geldige parkeerplaatsen in de buurt zijn. Zie ook figuur 19.



Figuur 19: Worst-case complexiteit: Alle auto's worden direct naar hun eindbestemming gestuurd omdat er geen geldige parkeerplaatsen in de buurt van eindbestemming zijn.

In het beste geval zijn dit er $(O(m! - n!))$, namelijk wanneer alle auto's aan een parkeerplek worden gestuurd. Dit wordt verduidelijkt in figuur 20



Figuur 20: Best-case complexiteit: De auto's worden allemaal naar een parkeerplaats gestuurd, voor zover er parkeerplaatsen zijn.

Dit is een enorme complexiteit die met het aantal voorgestelde auto's en parkeerplaatsen niet snel kan worden opgelost. Daarom wordt de oplossing berekend door middel van een heuristiek. Dit is een methode die een sub-optimale oplossing uitrekent. Niet alle mogelijkheden afgegaan, waardoor het programma eerder eindigt. Een nadeel is dat de oplossing niet altijd optimaal is. De heuristiek wordt geïmplenteerd met behulp van *lineair programmeren*. De formule uit figuur 5 wordt ingevoerd in een bestaand software pakket dat werkt met lineair programmeren. Vervolgens rekent dit pakket een sub-optimale oplossing uit.

7.5 Inrichten kostenfunctie

In deze paragraaf gaan we dieper in op de kostenfunctie. Deze is flexibel opgesteld dat er in deze fase van de ontwikkeling nog niet diep nagedacht hoeft te worden over de *precieze* invulling ervan. In het voorbeeld hebben we de absolute afstand tussen parkeerplaats en eindbestemming genomen als kosten, maar er zijn nog veel meer factoren die we kunnen laten meetellen. We kunnen bijvoorbeeld:

- Gehandicapten parkeerplaatsen toevoegen en gehandicapte bestuurders hierheen dirigeren.
- Dynamisch parkeerplaatsen toevoegen, bijvoorbeeld bij evenementen.
- Rode auto's voorrang geven boven witte.
- Bestuurders die meer betalen voorrang geven.

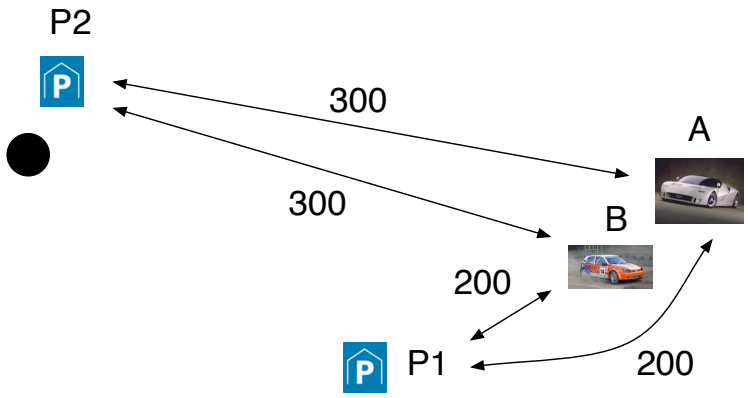
Om de flexibiliteit van deze methode te illustreren, geven we hier een voorbeeld van een iets andere toepassing van het model. We laten de eisen even los en gaan uit van de volgende situatie. Zie ook figuur 21 ter illustratie:

Voor de zoveelste keer staat er een file op de A1 tussen Muiden en Diemen. De ParkeerHeer is inmiddels een volgroeid product en wordt door onder andere twee bestuurders gebruikt. Deze twee rijden net Almere uit, richting Diemen.

Figuur 21: Twee auto's willen naar Diemen, maar tussen Muiden en Diemen staat een verschrikkelijk lange file. De zwarte stip is de eindbestemming van de twee auto's.

De ParkeerHeer is aangesloten op het NS treinverkeer en heeft uitgerekend dat het voor de bestuurders sneller is om naar station Weesp te rijden en de reis per trein te vervolgen, dan in de file te staan tot aan eindbestemming. Op het station in Weesp is nog genoeg plek om te parkeren.

Om de auto's naar het station in Weesp te sturen, kan de ParkeerHeer de volgende kostenmatrix maken:



Figuur 22: Kostenmatrix afgebeeld in de graaf

Omdat de kosten geminimaliseerd zullen worden, zullen beide de auto's naar het station in Weesp worden gestuurd, alwaar de bestuurders de trein kunnen nemen.

De kostenfunctie kan dus op verschillende manieren worden ingericht. Hierdoor hoeven er nog geen beslissingen genomen te worden over de criteria waarmee auto's worden toegewezen. Dit kan later tijdens de implementatie gedaan worden. Beter nog, kan tijdens de uitvoering de kostenfunctie worden aangepast om zo de functionaliteit te vergroten wanneer het systeem online is. Dit heeft echter wel de volgende beperkingen:

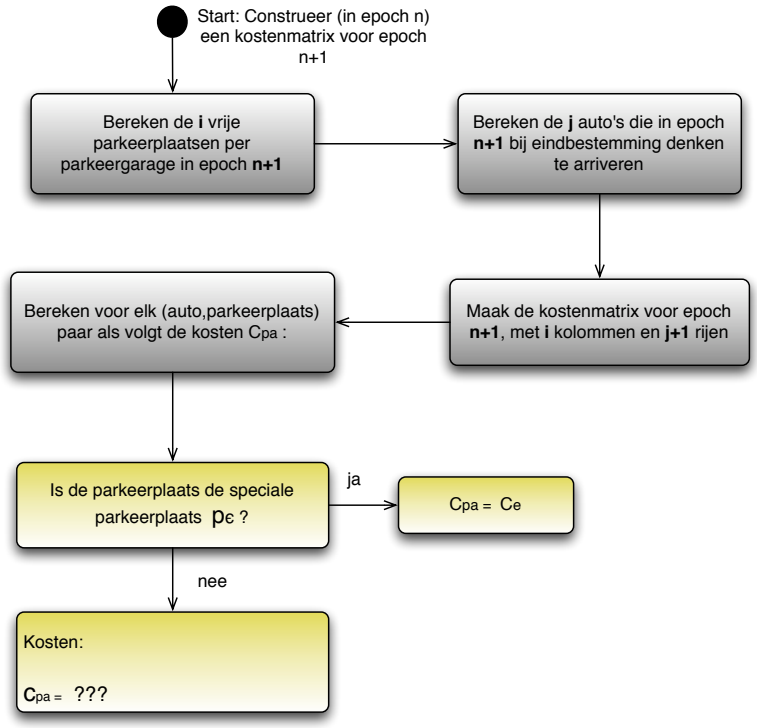
- De aankomsttijd van de auto's moet bekend zijn. Anders kunnen deze niet in de juiste epoch ingedeeld worden.
- Voor extra functionaliteit, zal in sommige gevallen extra data nodig zijn. Om gehandicapten te kunnen toewijzen aan parkeerplaatsen voor gehandicapten, zal een navigatiesysteem moeten kunnen aangeven dat de bestuurder gehandicapt is, en zal het parkeergaragesysteem het aantal vrije parkeerplaatsen voor gehandicapten door moeten geven, naast het aantal vrije gewone parkeerplaatsen.

7.6 Samenvattend

We hebben nu een model van de auto's, parkeergarages en eindbestemmingen. Ook is duidelijk hoe de benodigde data naar de ParkeerHeer kan worden gestuurd. De kostenmatrix speelt een belangrijke rol in het toewijzingsproces: Hiermee kan de functionaliteit van de ParkeerHeer eenvoudig worden aangepast. Voorwaarde voor het aanpassen van functionaliteit, is dat de benodigde data wordt aangeleverd door de parkeergarages en de bestuurders.

7.6.1 Beslisboom

Wanneer we het model en de kostenfunctie combineren met de methode waarmee de data wordt opgehaald, kunnen we de volgende beslisboom maken:



Figuur 23: Beslisboom van het construeren van een kostenmatrix. De grijze (in zwart-wit:grijs van onder) blokken stellen acties voor die van invloed zijn op de gehele kostenmatrix. De gele (in zwart-wit: grijs van boven) stellen acties voor die van invloed zijn op elk (auto,parkeerplaats) paar.

De exacte formule voor de kostenfunctie laten we hierbij open (De onderste gele rechthoek in afbeelding 23) omdat we deze later zullen invullen, zoals besproken.

7.6.2 Scenario's

Op dit punt kunnen een aantal scenario's grotendeels worden verwezenlijkt, met name door de invoering van de kostenmatrix en de reeds genomen beslissingen uit hoofdstuk 5:

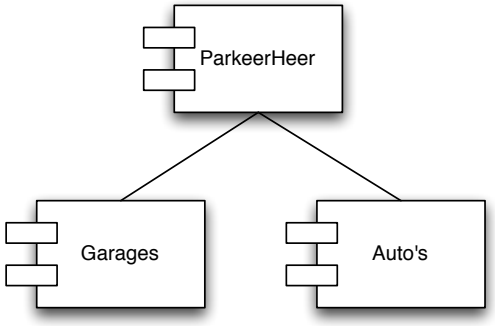
- Het verdelen van de auto's [UC1],
- Niet meenemen van volle parkeergarages bij de berekening [UC3],
- Prioritizering van auto's [UC4],
- De foutmarge in parkeergarages [UC3],
- De toewijs-grens [UC6]
- Het voorspellend vermogen [UC2]

8 Development Architecture

Met de aanpasbaarheidseisen in het achterhoofd wordt het systeem in dit hoofdstuk opgedeeld in verschillende subsystemen. Door deze indeling kunnen meerdere ontwikkelaars werken aan de verschillende delen zonder dat ze geheel afhankelijk van elkaar zijn. Dit is een veel gebruikte methode die ook wel *divide and conquer* wordt genoemd.

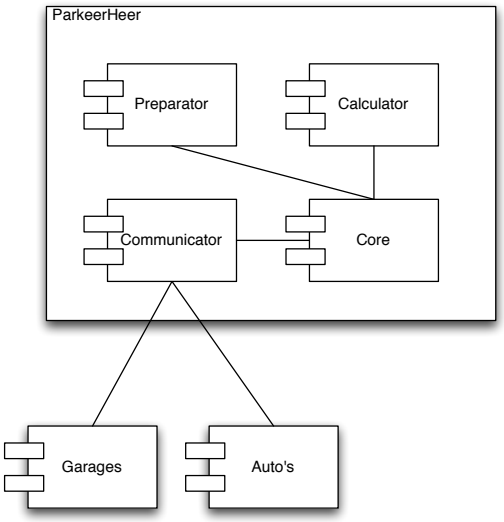
8.1 Drie subcomponenten

In hoofdstuk 5 is besloten om een generieke server te maken (zie ook figuur 10). Onderstaand figuur is een versimpeling hiervan. Het beeldt de drie componenten af: De ParkeerHeer, de auto's en de parkeergarages:



Figuur 24: Architectuur - Basis. Dit is een andere representatie maar geeft hetzelfde weer als figuur 10. De blokken zijn componenten, de lijnen tussen de blokken geven aan dat de twee componenten met elkaar in contact staan ofwel afhankelijk van elkaar zijn. De auto's staan dus in contact met de ParkeerHeer, evenals de Parkeergarages.

In de Logical Architecture is er een methode bedacht waarmee een oplossing kan worden gevonden. Zoals eerder besproken verzamelt de ParkeerHeer de inkomende data continu, maakt hiervan periodiek een kostenmatrix en rekent aan de hand daarvan een oplossing uit. Dit zijn drie acties die onder een eigen subcomponent worden ingedeeld. Daarnaast is er nog een component nodig die als een regelaar dient tussen alle andere componenten en verantwoordelijk is voor algemene opgeslagen configuraties en data. De indeling is in Figuur 25 afgebeeld:



Figuur 25: Architectuur - View componenten. De ParkeerHeer wordt opgedeeld in vier subcomponenten: De Communicator, verantwoordelijk voor de communicatie van en naar parkeergarages en auto's, De Preparator, verantwoordelijk voor het nemen van een snapshot, de Calculator, verantwoordelijk voor het uitrekenen van een oplossing met behulp van het model en de Core, voor een aantal algemene functies als een grafische interface, loggen en configuratie data. De verbindingen tussen de componenten geven afhankelijkheden weer.

Met deze indeling wordt meteen de voorwaarde geschept om aan de schaalbaarheid en performance eisen te voldoen: Zo zouden de drie componenten kunnen worden geschaald door ze op fysiek andere computers te laten draaien. Hier wordt in de Process View (Hoofdstuk 9) en de Physical View (Hoofdstuk 10) dieper op ingegaan. We bespreken nu kort de verschillende componenten, en argumenteren waarom deze als apart subsysteem worden ingedeeld:

De Communicator

Deze component is verantwoordelijk voor de communicatie van en naar alle parkeergarages en auto's. Het slaat ingekomen informatie op van de parkeergarages en auto's en verstuurt de oplossingen naar de auto's. Omdat de ParkeerHeer rekening moet houden met een groot aantal auto's en parkeergarages die bovendien ook nog op willekeurige momenten communiceren met de ParkeerHeer zullen er mechanismen moeten worden geïmplementeerd die hiermee overweg kunnen. Het opslaan en verzenden van grote hoeveelheden informatie is een potentieel *bottleneck*. Een bottleneck is een deel van het systeem wat de performance omlaag haalt. Hier kan een analogie gemaakt worden met een ketting: De zwakste schakel bepaalt de sterkte van de gehele ketting. Door de communicatie met de auto's en parkeergarages onder te brengen in een apart subcomponent wordt dit probleem geïsoleerd, waardoor het gemakkelijker kan worden opgelost.

De Preparator

Verwerkt periodiek de data die via de communicator is binnengekomen tot kostenmatrices zoals besproken in paragraaf 7.3. De functionaliteit van de ParkeerHeer moet kunnen worden aangepast, en de meeste functionaliteit zit in de

manier waarop de kostenmatrix is opgebouwd. Door de kostenmatrix in een apart subcomponent te plaatsen kan deze gemakkelijker worden doorontwikkeld, zonder rekening te hoeven houden met communicatie en het vinden van een oplossing.

De Calculator

Deze component rekent oplossingen uit met behulp van door de preparator vervaardigde kostenmatrices, zoals besproken in paragraaf 7.3. Voor het vinden van een oplossing wordt gebruik gemaakt van reeds bestaande software. Door de Calculator als apart subcomponent in te stellen kan deze bestaande software worden verbeterd. Ook is het mogelijk deze te vervangen door andere wellicht efficiëntere software, zonder de communicatie of kostenmatrix functionaliteit te hoeven veranderen.

De Core

De Core wordt gebruikt om het systeem te starten en biedt tevens een kleine set van functies die door het gehele systeem gebruikt kunnen worden. Hier valt te denken aan een log en debugging functionaliteit en een GUI om het systeem te stoppen of te starten. Ook is de Core verantwoordelijk voor het beheren en gebruiken van configuratie instellingen zoals de lengte van een epoch, eventuele gebruikersdata zoals voorkeuren of login en wachtwoord paren.

8.2 Communicator en de auto's en parkeergarages

Er moet rekening gehouden worden met een groot aantal verschillende systemen die aangesloten moeten kunnen worden op de ParkeerHeer. Al deze systemen hebben potentieel een andere manier van communiceren. In deze paragraaf gaan we dan ook de volgende vraag beantwoorden:

Q21: Hoe gaat de ParkeerHeer met alle verschillende typen navigatiesystemen en parkeergarages communiceren?

Deze vraag is deels al beantwoord: De Communicator is hiervoor verantwoordelijk. De volgende criteria worden gehanteerd bij het oplossen van deze vraag:

- Implementatiegemak voor clients: Hoe gemakkelijk is deze oplossing te implementeren bij clients (parkeergarages en navigatiesystemen)
- Implementatiegemak voor de server: Hoe gemakkelijk is deze oplossing te implementeren bij de server (De ParkeerHeer)
- Aanpasbaarheid: In hoeverre is het gemakkelijk in de toekomst extra functionaliteit toe te voegen?

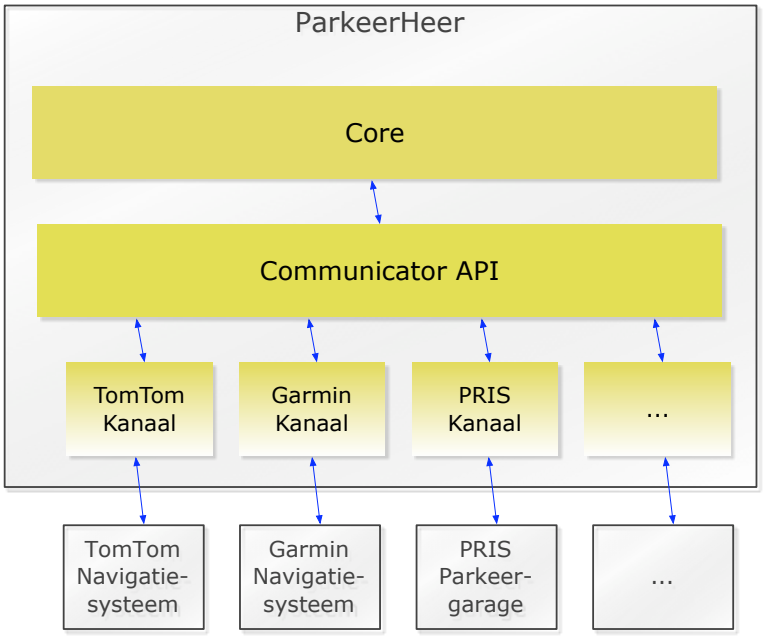
8.2.1 Optie: één generiek protocol

Met deze optie wordt er één generiek protocol ontworpen, waar alle parkeergarages en navigatiesystemen aan zullen moeten voldoen. Voordeel van deze optie is, dat het aan de kant van de ParkeerHeer gemakkelijk te ontwikkelen is. Omdat er maar één protocol gebruikt wordt, is het uitwerken van de Communicator is redelijk simpel. De navigatiesystemen en parkeergarages zullen

dit protocol moeten ondersteunen, wat betekent dat hier extra software voor ontworpen moet worden. Uit het vooronderzoek is gebleken dat er al systemen bestaan met een dergelijke functionaliteit. Een optie is om het protocol direct van deze systemen over te nemen. Het protocol van de PRIS systemen kan daarvoor gebruikt worden en het protocol van de ParkeerTomtom. Deze optie is niet heel erg aanpasbaar, zeker wanneer het eenmaal geplaatst is en in gebruik is genomen.

8.2.2 Optie: Communicator API

Met deze optie wordt voor elk type client een eigen protocol ontworpen. De Communicator wordt hierbij opgedeeld in meerdere subcomponenten die ook wel *Adapters* worden genoemd. De Communicator voorziet hierbij in een Application Programming Interface (API), die door de verschillende adapters wordt gebruikt. Een API is een generieke set van functies die door verschillende componenten (in dit geval de adapters) kan worden gebruikt. Voorbeelden in deze context zijn het aanmelden van een auto, of het doorgeven van de locatie van een auto. Let op dat we hierbij de eerste optie kunnen implementeren door één generieke adapter te maken. Zie ook figuur 26:



Figuur 26: Verschillende lagen in de Communicator component van de ParkeerHeer: Voor elk type navigatie- en parkeergaragesysteem kan een apart kanaal geschreven worden.

Met deze opstelling kan de ParkeerHeer met elk protocol omgaan, mits het de minimale eisen van de dataverstrekking naleeft die voorgeschreven wordt in de API van de Communicator.

Een nadeel van deze optie is, dat het programmeerwerk wordt verschoven van de clients naar de ParkeerHeer. In het geval er al een protocol wordt gebruikt, zoals

in de PRIS systemen en de ParkeerTomTom is dit een voordeel: Deze systemen zullen niet veranderd hoeven worden. In andere gevallen zal een team moeten werken aan zowel de client, als aan de server. Dit kan overigens ook als voordeel worden gezien: Eerder besloten we al dat de clients zo min mogelijk business logic moesten bevatten (paragraaf 5.5). Dit wordt door deze optie gerealiseerd.

Ontwerpbeslissing: Communicator API

We kiezen voor de tweede optie. We zetten de voor- en nadelen nogmaals op een rij:

	één generiek protocol	communicator API
Implementatiegemak clients	Groter	Kleiner
Implementatiegemak server	Klein	Groter
Aanpasbaarheid	Klein	Zeer groot

We kiezen voor deze optie omdat er reeds een parkeergaragesysteem en een navigatiesysteem met een protocol bestaan. Hierdoor hoeft voor deze systemen alleen nog de adapter te worden ontwikkeld die met de Communicator API communiceert. Dit verkort de ontwikkeltijd, terwijl de aanpasbaarheid groot blijft.

Daarnaast kan door deze keuze de scope beperkt worden: In dit document wordt alleen de ParkeerHeer besproken exclusief de implementatie van de verschillende adapters. Met deze ontwerpbeslissing hebben we use case 9 grotendeels beantwoord.

8.3 Samenvattend

In dit hoofdstuk is de ParkeerHeer ingedeeld in vier verschillende subcomponenten. De ontwerpbeslissingen die daarbij zijn gemaakt, zijn vooral gedreven door de grote mate van aanpasbaarheid en uitbreidbaarheid die de ParkeerHeer zal moeten hebben. Door deze indeling zal het voor toekomstige ontwikkelaars gemakkelijker worden om de ParkeerHeer aan te passen, omdat de belangrijkste taken zijn geïsoleerd in deze subcomponenten.

In Use cases 9 en 10 worden de mogelijke uitbreidingen besproken, deze zijn in dit hoofdstuk uitgewerkt. In Use case 11 wordt gesproken over configuratie mogelijkheden. Deze functionaliteit zal in het Core component moeten worden ingebouwd.

9 Process Architecture

In de Process Architecture View wordt de ParkeerHeer besproken in termen van gelijktijdig uitgevoerde taken. Zo'n taak wordt ook wel een *process* genoemd. Er is altijd een beperkte hoeveel rekenkracht en geheugen beschikbaar. Daarom is het belangrijk om de processen zo in te richten dat ze elkaar niet in de weg zitten.

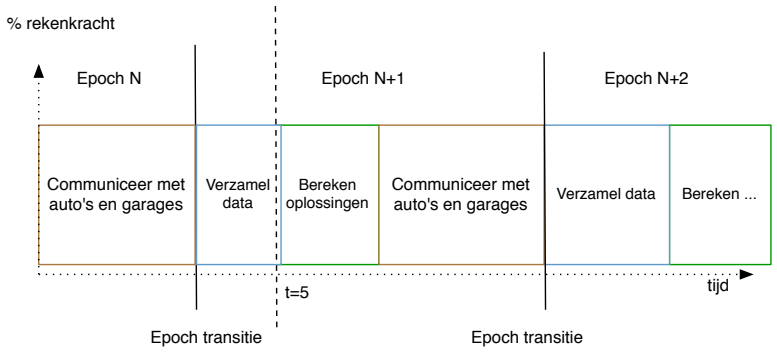
Daarnaast kunnen we de schaalbaarheid vergroten door de processen door verschillende processoren te laten uitvoeren. Hiervoor is een bepaalde rangschikking nodig.

9.1 Sequentieel of gelijktijdig uitgevoerde taken?

De taken van de ParkeerHeer zijn grofweg:

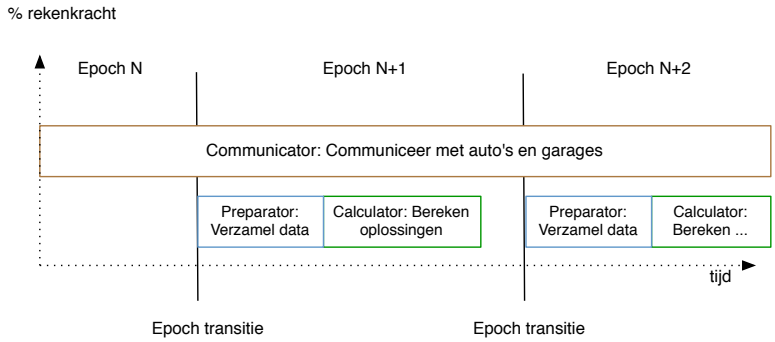
- (Continu) Informatie ontvangen van auto's en parkeergarages, en verzenden naar auto's
- (Periodiek) een oplossing uitrekenen

Deze twee taken moeten tegelijk worden uitgevoerd: Stopt de ParkeerHeer met informatie ontvangen wanneer het een oplossing gaat uitrekenen, dan betekent dit dat alle informatie die wordt verzonden door de auto's en parkeergarages, verloren gaat. Deze onwenselijke situatie wordt afgebeeld in figuur 27. In deze figuur is de tijd uitgezet tegen de totale hoeveelheid rekenkracht. De ParkeerHeer doet in deze situatie alles sequentieel: Dat betekent dat het met één taak tegelijk bezig is.



Figuur 27: Verkeerde implementatie: De ParkeerHeer stopt met communiceren met de auto's en parkeergarages op het moment dat het een oplossing gaat berekenen. Stuurt een parkeergarage een update op bijvoorbeeld tijd $t = 5$, dan zal deze niet door de ParkeerHeer verwerkt worden.

De oplossing is om ten minste twee processen tegelijk te laten draaien, zoals in figuur 28:



Figuur 28: De verschillende processen in de tijd: Aan het begin van elke epoch wordt er een nieuwe oplossing uitgerekend. De Communicator moet ten alle tijden kunnen communiceren met de auto's en parkeergarages. De Preparator en Calculator moeten periodiek een oplossing kunnen uitrekenen.

In deze figuur is de communicatie en het vinden van een oplossing verdeeld in twee onafhankelijke processen die gelijktijdig worden uitgevoerd.

In de volgende paragraaf wordt besproken hoe de ParkeerHeer verder kan worden opgedeeld in verschillende processen. De grafiek in het bovenstaande figuur 28 wordt gebruikt om de verschillende alternatieven te illustreren. Vooralsnog gaan we ervan uit dat het uitrekenen van een nieuwe oplossing evenveel rekenkracht kost als de communicatie met de auto's en parkeergarages. Hoe dit in werkelijkheid is verdeeld is moeilijk te zeggen. We doen hier dan ook geen uitspraken over. Wanneer blijkt dat een process meer rekenkracht vereist dan anderen, kan ervoor worden gekozen dit process op een aparte computer te laten draaien, zoals besproken in de physical architecture view (hoofdstuk 10).

9.2 Gelijktijdig uitgevoerde taken

We beantwoorden nu de volgende vraag:

Q22: Hoe gaat de ParkeerHeer de verschillende taken tegelijk uitvoeren?

We laten ons hier leiden door de volgende criteria:

- Performance: In hoeverre kan deze optie de afgesproken hoeveelheid auto's en parkeergarages bedienen en tegelijkertijd een oplossing uitrekenen?
- Uitbreidbaarheid: Hoe gemakkelijk is het om de verschillende processen op andere computers en/of processoren te laten draaien, zodat er meer bestuurders en parkeergarages bediend kunnen worden?
- Aanpasbaarheid: Hoe gemakkelijk is het om extra functionaliteit toe te voegen, of extra type clients?
- Eenvoud: Hoe complex is deze oplossing? Is hij gemakkelijk te realiseren?

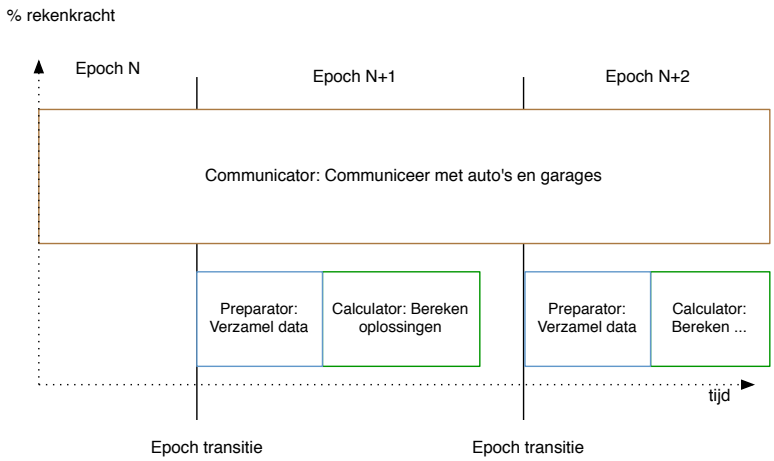
Optie: Twee processen: één voor ontvangen en verzenden van data, één voor uitrekenen oplossingen

Deze optie is reeds afgebeeld in figuur 28. Een nadeel van deze optie is, dat het moeilijk schaalbaar is. Een process kan moeilijk draaien op meerdere processo-

ren, waardoor de rekenkracht ook moeilijker kan worden vergroot. Daarnaast zal het communicator process voortdurend moeten wisselen van protocol: In de development architecture view is gesproken over de verschillende protocollen die gebruikt gaan worden. Wanneer de communicator informatie binnen krijgt van een client zal deze eerst moeten uitzoeken welk type het is en het daarbij horende protocol moeten gebruiken. Gezien het grote aantal verzoeken dat de ParkeerHeer zal ontvangen, zal dit de performance niet ten goede komen.

Optie: één process voor elke adapter

Met deze optie wordt voor elke adapter een apart process gedraaid. Door de grote hoeveelheid gelijktijdige processen kan deze oplossing gemakkelijk worden geschaald naar meerdere computers; elk proces kan een eigen processor toegewezen krijgen. De situatie kan zich voordoen waarbij één type client veel dataverkeer genereert, meer dan de andere. In deze optie kan hier niets aan gedaan worden. In dat geval is het erg moeilijk om de capaciteit van de ParkeerHeer omhoog te schalen door meerdere processen te verdelen over meerdere processoren, omdat een type client maar één bijhorend proces heeft.



Figuur 29: N + 1 processen

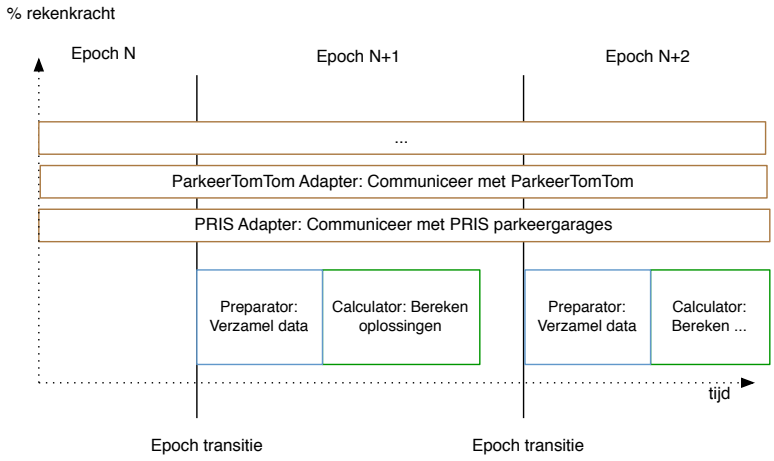
Optie: één proces voor elke adapter, plus een loadbalancer

In deze optie wordt er voor elke adapter een apart process gedraaid. Daarbij wordt één extra process gestart die de communicatie van en naar de clients regelt: Dit process wordt ook wel een loadbalancer genoemd.

Met deze optie kan de loadbalancer efficiënt informatie doorsluizen en het juiste protocol bepalen. In geval van schaling naar meerdere fysieke computers kan de loadbalancer op de Centrale Server van de ParkeerHeer blijven staan, terwijl elke adapter een eigen computer krijgt aangewezen.

Ook kan met deze optie meerdere adapter processen worden ingesteld voor hetzelfde type client. De loadbalancer kan vervolgens bepalen naar welke adapter

het verzoek wordt gestuurd. Hierdoor is in het geval één type client veel dataverkeer genereert mogelijk de capaciteit omhoog te schalen.



Figuur 30: N + 2 processen

Keuze: één proces voor elke adapter, plus een loadbalancer

We kiezen voor de laatste optie. Hoewel dit niet de meest eenvoudige oplossing is, biedt het wel de meeste perspectief met betrekking tot de schaalbaarheid. In onderstaande tabel zijn alle voor en nadelen van de besproken opties nogmaals opgenomen.

Criterium	2 processen	n +1 processen	n + 2 processen
Performance	Laag	Goed	Goed
Uitbreidbaarheid	Moeilijk	Gemiddeld	Goed
Aanpasbaarheid	Moeilijk	Gemiddeld	Goed
Eenvoud	Goed	Gemiddeld	Complex

9.3 Samenvattend

Door het gebruik van verschillende processen in de Communicator kan de ParkeerHeer ten alle tijden voldoen aan verzoeken van Parkeergarages en auto's (Use cases 5, 6 en 7). Wanneer een bestuurder haar eindbestemming wijzigt zal het maximaal 1 epoch duren voor de ParkeerHeer een nieuwe parkeergarage heeft gevonden (Use case 6).

10 Physical Architecture

In de Development Architecture View is de ParkeerHeer opgedeeld in software pakketten. In de Process Architecture View is de ParkeerHeer opgedeeld in processen. In dit deel wordt er gekeken hoe deze pakketten en processen fysiek op verschillende computers en processoren kunnen draaien. Door hier een beeld van te geven, wordt duidelijk op welke manier de ParkeerHeer geschaald kan worden. We benoemen hierbij eerst de grootste performance bottlenecks, waarna er een methode wordt gegeven waarmee deze bottlenecks verkleind kunnen worden.

10.1 Performance Bottlenecks

We benoemen drie bottlenecks: De opslag van data, de communicatie van en naar de loadbalancer, en het uitrekenen van een oplossing. Deze behandelen we nu één voor één.

10.1.1 Opslag van data

Er wordt gewerkt met een grote hoeveelheid variabelen. Een kostenmatrix bijvoorbeeld, herbergt in het ergste geval n keer $m + 1$ variabelen (waarbij n het aantal auto's is, en m het aantal parkeerplaatsen). Het is wenselijk deze gegevens op te slaan, omdat de ParkeerHeer waarschijnlijk doorontwikkeld zal worden waarbij er toekomstvoorspellingen gedaan gaan worden over het aantal vrije parkeerplaatsen. Met deze gegevens kan dit toekomstvoorspellingsmodel gemaakt en getest.

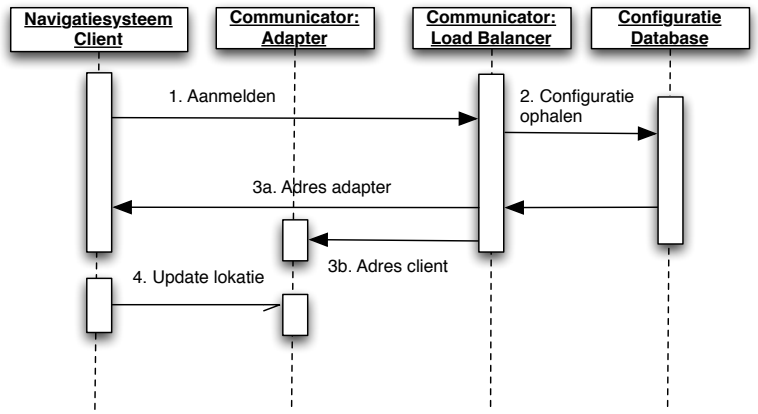
Er wordt onderscheid gemaakt tussen twee typen data: De data die nodig is voor het verwerken van een oplossing en configuratie data. Bij de eerste kan gedacht worden aan bijvoorbeeld de locatie van de auto's en hun eindbestemmingen. Login en wachtwoord van de parkeergarages en bestuurders is een voorbeeld van configuratiedata. De reden voor deze verdeling is dat verschillende processen / componenten deze data gaan gebruiken. Door deze verdeling kunnen de verschillende typen data ook fysiek op een andere locatie (computer) staan.

Zowel de adapter processen als het calculator/preparator process schrijven naar de oplossingen-data. De communicator loadbalancer schrijft en leest van en naar de configuratie data.

10.1.2 Communicatie van en naar loadbalancer

De in de Process Architecture geïntroduceerde loadbalancer heeft als doel alle communicatie door te sluizen naar de betreffende adapter. De navigatiesystemen en parkeergarages communiceren regelmatig met de ParkeerHeer: Auto's zullen bijvoorbeeld hun locatie en aankomsttijd regelmatig updaten. Dit probleem wordt ook wel beschreven in *Distributed Systems, Principles and Paradigms* [11] waarbij de loadbalancer ook wel de *Logical Switch* wordt genoemd. Er wordt een oplossing geboden in de vorm van een zogenaamde *TCP Handoff*. Deze methode zorgt ervoor dat de Logical Switch alleen bij het eerste verzoek van de client een functie heeft: Het geeft de client het adres van de juiste adapter. Hierdoor wordt een hoop communicatie van en naar de Logical Switch

voorkomen. Deze methode zou, weliswaar in aangepaste vorm, ook gebruikt kunnen worden voor de ParkeerHeer. De volgende figuur illustreert dit met behulp van een UML sequence diagram:



Figuur 31: Gebaseerd op een TCP Handoff beschreven in Distributed Systems, Principles and Paradigms [11]: De loadbalancer wordt alleen bij het eerste contact gebruikt om de juiste adapter te vinden. Hierna verloopt alle communicatie via deze adapter, waardoor de loadbalancer ontlast wordt.

In de derde stap verstuurt de Communicator Load Balancer de benodigde configuratie gegevens naar de juiste adapter, *en* naar de client. Deze configuratiedata is bijvoorbeeld een sessie-id, waarmee de twee elkaar kunnen identificeren in verband met veiligheid, of bepaalde instellingen die de client heeft, zoals bijvoorbeeld het willen navigeren naar gehandicaptenparkeerplaatsen.

10.2 Uitrekenen oplossing

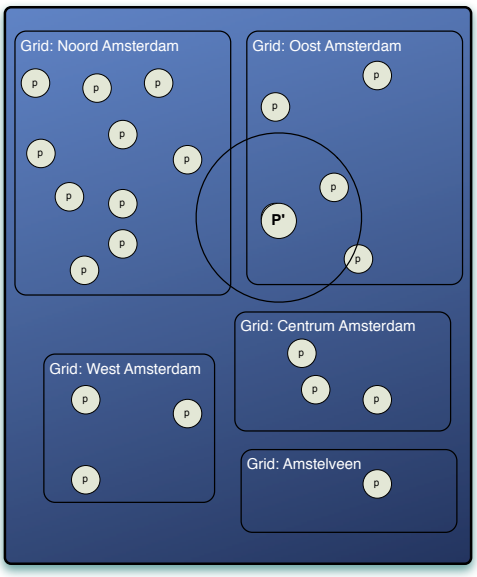
Het model dat is uitgewerkt in de Logical Architecture view wordt ingevoerd in een software pakket dat een oplossing uitrekent. Omdat het probleem een grote complexiteit heeft, is het niet mogelijk binnen afzienbare tijd een oplossing uit te rekenen. Daarom rekent het software pakket een sub-optimale oplossing uit. Hierbij geldt dat hoe minder auto's en parkeergarages in het systeem zitten, dichter de sub-optimale oplossing bij de optimale oplossing zit. Dit kunnen we bewerkstelligen door de kostenmatrix op te delen, en de delen één voor één te voeren aan het softwarepakket.

10.2.1 Opdelen kostenmatrix

In de kostenmatrix zitten een hoop waarden die we niet nodig hebben. Zo is het bijvoorbeeld logisch dat een auto met eindbestemming Utrecht centraal niet naar een parkeergarage in Groningen gedirigeerd wil worden. De kostenmatrix kan daarom worden opgedeeld in meerdere kleine matrices. De deling gebeurt door een grid over Nederland te leggen. Figuur 32 geeft een voorbeeld van zo'n grid.

Figuur 32: Voorbeeld van een verdeling van het grid

Bij het opdelen van de parkeergarages moeten we goed opletten dat we geen conflicten krijgen bij het toewijzen van auto's aan parkeergarages: We willen voorkomen dat een bestuurder niet aan een bepaalde parkeergarage wordt toegewezen omdat deze buiten het grid valt, maar wel binnen de maximale afstand. In figuur 33 wordt een dergelijk conflict uitgebeeld.



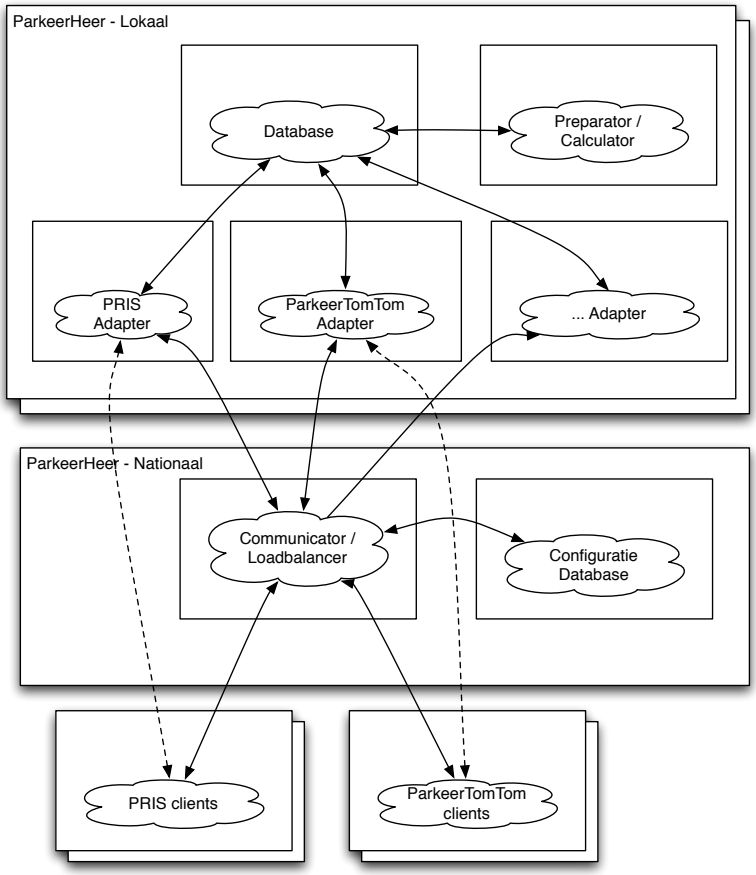
Figuur 33: Onwenselijke verdeling van het Grid: De cirkel om de grotere parkeergarage P' geeft de maximale afstand aan tussen eindbestemming en de parkeergarage. Een bestuurder kan een eindbestemming in het grid van Noord-Amsterdam hebben en toch worden toegewezen aan P' in grid Oost-Amsterdam.

10.2.2 Gevolgen voor de complexiteit

Zoals eerder aangegeven is de best-case complexiteit van orde $O(m! - n!)$, en de worst-case complexiteit van orde $O(m^n)$, waarbij m het aantal parkeerplaatsen is, en n het aantal auto's. Deze complexiteit zal niet afnemen wanneer de kostenmatrix wordt opgedeeld in meerdere grids. Wel zullen n en m kleiner worden per grid. Door de enorme stijging van het aantal mogelijkheden bij een lineaire stijging van n en m, kunnen we stellen dat een opdeling van het grid nuttig zal zijn. Hoewel het aantal mogelijkheden nog steeds erg groot is, zal de gebruikte integer programming methode een betere oplossing vinden.

10.3 Samenvattend

In dit hoofdstuk zijn de variabelen geïsoleerd die de performance van de ParkeerHeer beïnvloeden. Vervolgens is er een aantal alternatieven aangedragen om de performance te verbeteren, en dus ook om het systeem beter te schalen. Use case 12 is hiermee dan ook gerealiseerd. De processen kunnen nu als volgt gemapt worden op fysieke processoren of computer, afgebeeld in figuur 34:



Figuur 34: Schema van processen en computers van de ParkeerHeer: De wolken stellen de processen voor. Elk proces kan op een fysiek andere computer en/of processor (rechthoeken) draaien om zo de performance te vergroten. De pijlen tussen twee processen geven aan dat de processen met elkaar communiceren. De gestreepte lijnen geven de communicatie weer tussen clients en adapters, na het eerste contact met de loadbalancer.

Deel III
Discussie en Conclusie

Hoofdstuk 11: Conclusie

Hoofdstuk 12: Discussie

11 Conclusie

In dit hoofdstuk komen we terug op de doelstelling van dit onderzoek en kijken we of deze beantwoord kan worden. De doelstelling is:

Hoe kan de gemiddelde zoektijd van het zoekend verkeer worden verminderd?

En het antwoord (tussen haakjes staan verwijzingen naar paragrafen in dit hoofdstuk waar meer informatie te vinden is over het onderwerp):

Door navigatiesystemen die gebruikt worden door het zoekend verkeer te voorzien van extra functionaliteit, waarmee deze met een centrale server (de ParkeerHeer) in verbinding komt te staan (5.4). De ParkeerHeer verzamelt continu alle gegevens van aangesloten parkeergarages en navigatiesystemen en wijst periodiek de auto's toe aan vrije parkeerplaatsen (7.1). De toewijzing gebeurt door middel van een wiskundig model (7.3). De kern van dit model is de kostenmatrix, een tweedimensionale tabel waarbij elk veld een (auto,parkeerplaats) paar voorstelt. Door getallen toe te kennen aan de velden kan de functionaliteit van de ParkeerHeer worden bepaald: een auto met voorrang moet een lagere waarde krijgen in de bijbehorende kostenmatrix velden, waardoor deze eerder toegewezen zal worden aan een parkeerplaats. De functie die wordt gebruikt om de waarden in de kostenmatrix te berekenen, wordt ook wel de kostenfunctie genoemd (7.3.4). Door deze constructie is de ParkeerHeer flexibel en kan de functionaliteit gemakkelijk worden aangepast. De ParkeerHeer is dusdanig ingericht dat het tegelijk een oplossing uitrekent en openstaat voor communicatie (9). Daarnaast is de ParkeerHeer verdeeld in subcomponenten zodat verschillende ontwikkelteams er gelijktijdig aan kunnen werken (8). Ook kan de ParkeerHeer worden verdeeld over verschillende fysieke computers, waardoor het mogelijk is een groot aantal bestuurders en parkeergarages te bedienen (10).

12 Discussie

In dit hoofdstuk kijken we kritisch naar het ontwerp van de ParkeerHeer. Welke gevolgen hebben bepaalde ontwerpbeslissingen gehad op het ontwerp? En hoe is getracht om de negatieve gevolgen op te lossen, en is dit gelukt? Als laatste kijken we naar de toepasbaarheid van de ParkeerHeer: is het mogelijk om dit systeem daadwerkelijk te implementeren, of stuiten we hierbij op praktische onmogelijkheden?

12.1 Reserveren van parkeerplaatsen

De keuze om parkeerplaatsen niet te reserveren bij parkeergarages was een zeer bewuste, omdat het nog niet goed realiseerbaar is in de huidige infrastructuur: er zijn niet genoeg middelen om een bestuurder te authentiseren bij de ingang van een parkeergarage en niet alle parkeergarages hebben meerdere ingangen.

Dat de auto's niet worden gereserveerd bij parkeergarages heeft bepaalde gevolgen voor de werking van de ParkeerHeer. De kans op een weigering is namelijk een stuk groter: de parkeerplaats kan bij aankomst bezet zijn door een auto die niet is meegenomen in het systeem (een bestuurder die geen gebruik maakt van de diensten van de ParkeerHeer). Er worden twee methoden gebruikt om het aantal weigeringen te verminderen: het instellen van een foutmarge en het voorspellen van het aantal vrije parkeerplaatsen in de toekomst. We behandelen deze twee nu één voor één:

Foutmarge

Met het instellen van de foutmarge wordt het aantal potentiële weigeringen verminderd. Het idee hierachter is, dat een bestuurder niet naar een parkeergarage wordt gedirigeerd wanneer de kans groot is dat deze geen vrije plekken heeft. Een nadeel van deze methode is echter, dat mogelijk hierdoor een te negatieve schatting wordt gemaakt van het aantal vrije plaatsen. Bestuurders worden naar een parkeergarage gestuurd die verder weg ligt, terwijl er dichterbij nog genoeg plaats is. Of nog erger: Alle parkeergarages zijn bijna vol. In dit geval wordt de bestuurder direct naar eindbestemming gestuurd, omdat de ParkeerHeer denkt dat er geen geschikte parkeergarages zijn. Wanneer de auto's gereserveerd kunnen worden bij parkeergarages is het instellen van een foutmarge niet meer nodig, omdat met grote zekerheid gezegd kan worden dat de parkeerplaats vrij is. Deze wordt immers vrijgehouden door de parkeergarage.

Voorspellend vermogen

Het is moeilijk de toekomst te voorspellen en helaas komen voorspellingen niet altijd uit. Dit geldt ook voor de toekomstvoorspellende algoritmen in de PRIS systemen. Er zal hierdoor altijd een foutmarge gehanteerd moeten worden.

In de toekomst kan het voorspellend vermogen in de ParkeerHeer zelf worden geïmplementeerd. Wanneer de in- en uitkomende data op een slimme manier worden opgeslagen kan hiermee een voorspellend model worden gemaakt. Hiermee kan het voorspellend vermogen worden verbeterd met behulp van zelflerende algoritmen. Hiermee zou use case 2 kunnen worden gerealiseerd.

Daarnaast kunnen de voorspellingen fluctueren, zoals in het volgende voorbeeld:

Auto a rijdt van Amsterdam Sloten naar Schiphol en wil daar parkeren. Bij het inschakelen van de ParkeerHeer wordt er meteen een parkeerplaats toegewezen. Echter na 5 minuten blijkt dat deze parkeergarage helemaal bezet zal zijn bij aankomst. De bestuurder wordt naar een andere parkeergarage gestuurd. Eenmaal aangekomen blijkt dat de schatting niet correct was en dat de bestuurder te ver heeft gereden.

Dit probleem doet zich voor bij wisselende parkeergarage voorspellingen, maar ook voor wisselende auto-voorspellingen, waarbij de aankomsttijd fluctueert tussen twee epochs in.

Wanneer de auto's gereserveerd worden in parkeergarages kan de ParkeerHeer de toekomst vele malen beter voorspellen: het is dan immers bekend hoeveel vrije plaatsen de parkeergarages hebben, en hoeveel daarvan gereserveerd zijn.

12.2 Statisch oplossen van een dynamisch probleem

De manier waarop de kostenmatrix wordt gemaakt maakt het noodzakelijk om periodiek een snapshot te maken van alle variabelen en deze als statisch te behandelen. Dit betekent dat de oplossing die wordt gevonden altijd uitgaat van verouderde data. Het kan voorkomen dat een bestuurder haar bestemming wijzigt en vervolgens alsnog naar een parkeergarage in de buurt van haar oude bestemming wordt gestuurd.

Een ander mogelijk onwenselijk scenario is er één waarin de bestuurder constant wisselt van parkeergarage, dus dat de auto van het spreekwoordelijke kastje naar de muur wordt gestuurd. De auto's updaten hun locatie periodiek, en er wordt periodiek een oplossing uitgerekend. Hierdoor kan het voorkomen dat in periode n de auto naar parkeerplaats b wordt gestuurd, en in de opvolgende epoch naar parkeerplaats c, een stuk verderop. Dit is een zeer onwenselijke situatie omdat een bestuurder hiermee veel langer onderweg is, terwijl het aantal weigeringen wel wordt verminderd. In deze situatie is het doel van de ParkeerHeer wel bereikt (het verminderen van het aantal weigeringen), maar schieten de bestuurders er helemaal niets mee op. Deze schommelingen kunnen deels worden voorkomen door de kostenfunctie goed in te richten. Zo zou het navigatiesysteem een geschiedenis van reeds toegewezen parkeergarages kunnen meegeven of, in het meest simpele geval, een bepaalde voorkeur. Parkeerplaatsen horende bij deze parkeergarage zullen dan in de kostenfunctie een lagere waarde krijgen, waardoor de bestuurder daar eerder naartoe wordt gestuurd.

12.3 Kostenmatrix

Met de kostenmatrix kan een groot gedeelte van de functionaliteit worden bepaald. Deze functionaliteit kan alles zijn: Bestuurders met een rode hoed kunnen voorrang krijgen, of Software Engineering studenten aan de Vrije Universiteit. Zo kan de situatie op de weg worden beïnvloed wanneer een groot aantal bestuurders gebruik maakt van de ParkeerHeer: bij een groot evenement waar

duizenden auto's op de weg zijn (zoals bijvoorbeeld een elfstedentocht) kan een deel van de bestuurders naar een andere parkeerplaats worden gedirigeerd om zo de belasting op de wegen gelijkmatiger te verdelen.

In dit onderzoek wordt nog geen voorstel gedaan voor de precieze invulling van de kostenfunctie. De voornaamste reden hiervoor is, dat het erg moeilijk is om aan te tonen dat de ontworpen functie correct functioneert. De wensen van de bestuurder en de mate van succes kunnen moeilijk vooraf worden bepaald.

De flexibiliteit van de kostenfunctie speelt hier handig op in, doordat deze in een zeer laat stadium van het ontwikkelproces nog kan worden verbeterd. Het is zelfs mogelijk om een feedback-functionaliteit in te voeren in de navigatiesystemen, waarbij de gebruiker gevraagd wordt of ze goed geholpen is. Deze feedback kan worden gebruikt om een zelflerende kostenfunctie te creeëren. Zover is het echter nog niet. Tot dan zullen variabelen als maximale afstand tot eindbestemming en parkeergarage, lengte van een epoch, en vele anderen met de hand worden aangepast en worden verbeterd tot een ideale waarde is gevonden.

12.4 Toepasbaarheid

Op papier hebben we nu een systeem ontworpen, maar kan de ParkeerHeer al worden gemaakt en in gebruik worden genomen? Welke obstakels zullen we hierbij tegenkomen?

Aansluiten navigatiesystemen

De navigatieystemen kunnen niet zomaar worden aangesloten op de ParkeerHeer: Hiervoor zal een *plugin* voor moeten worden geschreven. Dit is een kleine uitbreiding op de software die nieuwe functionaliteit mogelijk maakt. Het navigatiesysteem moet hiervoor wel de mogelijkheid bieden. In ieder geval ondersteunen TomTom en Garmin deze structuur, evenals Google Maps en Microsoft Earth. Er kan hier dan ook worden aangenomen dat dit geen probleem zal zijn.

De navigatiesystemen moeten worden aangesloten op het internet. Op dit moment is dit niet het geval voor het grootste deel van de navigatiesystemen in auto's. Wel zijn we er in ons vooronderzoek achter gekomen dat het percentage navigatiesystemen-met-internet in de toekomst zal groeien, mede doordat leveranciers als TomTom extra services willen aanbieden die afhankelijk zijn van het internet, zoals het weergeven van files.

Aansluiten parkeergaragesysteem

De parkeergarages zullen hun informatie moeten delen met de ParkeerHeer. Op het moment van schrijven zijn dit nog gesloten systemen. Er wordt vanuit de politiek druk uitgeoefend op gemeenten en commerciële parkeergaragebedrijven om deze informatie beschikbaar te stellen. Dit wordt met name inggegeven door wetenschappelijke onderzoeken die hebben aangetoond dat met deze oplossing een hoop fileleed kan worden voorkomen. Het is dan ook niet onwaarschijnlijk dat dit binnenkort zal gebeuren.

Veiligheid

De ParkeerHeer is afhankelijk van de verkregen informatie van de parkeergarages, maar het voortbestaan van de parkeergarages is afhankelijk van haar inkomsten en dus het aantal auto's dat er parkeert. Dit creëert een zeker spanningsveld tussen de twee: De parkeergarages kunnen, om meer auto's naar zich toe te lokken, verkeerde data versturen naar de ParkeerHeer. Hierdoor zal de omzet van de parkeergarage toenemen, maar ook het aantal weigeringen. Deze situatie kan bijvoorbeeld worden voorkomen door contracten af te sluiten tussen Parkeerheer en Parkeergarages, en controle mechanismen in te voeren die een signaal geven wanneer deze contracten geschonden worden.

Deel IV
Appendices

Ontwerpvragen

Eisen

Woordenlijst

Referenties

Ontwerpvragen & Eisen

Verklaring afkortingen:

- 2de fase : Fase waarin de ParkeerHeer wordt doorontwikkeld. Onderwerpen die in deze fase vallen zullen niet in dit document worden besproken of uitgewerkt.
- VO: Vooronderzoek
- UPx: Uitgangspunt x
- Qx : Ontwerpvraag x
- Rx-y : Eis x-y

Ontwerpvragen

ID	Volgt uit	Wordt beantwoord in	Ontwerpvragen	Categorie
Q1	5.1.1	5.2	Welke factoren gaan worden beïnvloed om de zoektijd te verkleinen?	R1
Q2	5.1.1	Deel 3	Hoeveel moet de ParkeerHeer de zoektijd minimaal verkleinen?	R1
Q3	5.1.2	Deel 3	Hoe snel moeten de auto's aan de parkeergarages worden toegewezen?	R1
Q4	5.1.2	Deel 3	Hoe kan de ParkeerHeer geschaald worden zodat het meer auto's kan ondersteunen?	R2
Q5	5.1.2	Deel 3	Hoe kan de ParkeerHeer geschaald worden zodat het meer parkeerplaatsen kan ondersteunen?	R3
Q6	5.1.3	Deel 3	Welke delen van de ParkeerHeer kunnen worden aangepast, en welke niet?	R4
Q7	5.2	5.3	Op welke manier gaan het aantal weigeringen worden verminderd?	R1
Q8	5.2	5.4	Hoe gaan de bestuurders de benodigde informatie aanleveren en aangeleverd krijgen?	R5
Q9	5.2	5.4	Hoe gaan de parkeerplaatsen de benodigde informatie aanleveren en aangeleverd krijgen?	R5
Q10	5.3	Deel 3	Hoe kan de foutgevoeligheid worden verminderd?	R1
Q11	5.3	Deel 3	Welke eisen stellen we aan het toewijzen?	R1
Q12	5.3	5.4	Welke bestaande systemen kunnen gebruikt worden om de toewijzing tot stand te kunnen laten komen?	R2,R3,R4
Q13	5.3	5.4	Welk subsysteem gaat de oplossing berekenen?	R4
Q14	5.4	Deel 3	Welke gegevens moeten er verzonden worden van en naar de parkeergarages?	R5
Q15	5.4	Deel 3	Welke gegevens moeten er verzonden worden van en naar de bestuurders?	R5
Q16	5.4	Deel 3	Hoe kan de communicatie tussen ParkeerHeer en haar clients worden beveiligd?	R5
Q17	6.1	6.1	Op welk moment moet er een oplossing worden berekend?	R1
Q18	6.1	2de fase	Hoe kan het aantal schommelingen worden verminderd?	R1
Q19	6.1	2de fase	Hoe lang moet een periode duren?	R1
Q20	6.2	6.2	Hoe moet de data verzameld worden?	R5
Q21	8.2	8.2	Hoe gaat de ParkeerHeer met alle verschillende typen navigatiesystemen en parkeergarages communiceren?	R4

Figuur 35: Ontwerpvragen overzicht. De laatste kolom geeft de hoofdcategorie aan waar deze vraag in valt. Dit betekent niet dat deze vraag *hoofdzakelijk* over deze categorie gaat.

Eisen

ID	Uitleg	Volgt uit	Eisen: Functionaliteit
R1	5.1.1	UP1	De ParkeerHeer zal de zoektijd verkleinen
R1-a	5.2.4	Q1	De ParkeerHeer zal het aantal keer dat auto's geweigerd worden bij een parkeergarage omdat deze geen vrije plaatsen heeft, verminderen
R1-b	5.3.4	Q7	De ParkeerHeer zal auto's naar vrije parkeerplaatsen dirigeren
R1-c	5.3.4	Q7	De ParkeerHeer zal geen vrije plaatsen reserveren bij parkeergarages
R1-d	5.3.4	Q7	De ParkeerHeer zal rekening houden met alle aangesloten auto's en parkeergarages
R1-e	5.3.4	Q7	De ParkeerHeer zal een schatting maken van het aantal vrije parkeerplaatsen in de toekomst
R1-f	5.6	Q10	De ParkeerHeer zal een foutmarge hanteren omtrent het geschatte aantal vrije parkeerplaatsen in de toekomst
R1-g	5.6	Q11	Wanneer er voor een auto geen parkeergarage gevonden kan worden zal de ParkeerHeer deze naar haar eindbestemming dirigeren
R1-h	5.6	Q11	Aangesloten bestuurders zullen worden toegewezen aan parkeergarages die zo dicht mogelijk bij hun eindbestemming liggen
R1-i	5.6	Q11	De afstand tussen eindbestemming en toegewezen parkeergarage mag niet groter zijn dan een voorafbepaalde minimum
R1-j	5.6	Q11	Er zullen niet meer auto's worden toegewezen aan een parkeergarage dan er vrije plaatsen zijn
R1-k	5.6	Q3	De ParkeerHeer zal een auto aan een parkeergarage hebben toegewezen voordat de afstand van de auto tot haar eindbestemming kleiner is dan een voorafbepaald minimum
R1-l	7.3	R1-g	Een bestuurder zal naar precies 1 parkeerplaats worden gestuurd, of direct naar eindbestemming
R1-m	7.3	R1-g	Een parkeerplaats zal door 0 of 1 auto's worden bezet
R1-n	7.3	R1-h	De kostenfunctie bepaalt in hoeverre een auto wil parkeren bij een parkeerplaats
R1-o	7.3	7.3	De auto's worden dusdanig toegewezen aan parkeerplaatsen dat de totale kosten (berekent door middel van de kostenfunctie) minimaal zijn

ID	Uitleg	Volgt uit	Eisen: Uitbreidbaarheid
R2	5.1.2	UP2	Het aantal bestuurders dat is aangesloten op de ParkeerHeer zal zo groot mogelijk zijn
R2-a	5.1.2	UP2, VO	De ParkeerHeer zal tot 20.000 auto's kunnen bedienen
R2-b	5.1.2	UP2, VO	De ParkeerHeer zal dusdanig worden ingericht dat het door hardware-uitbreiding tot 2.000.000 auto's kan bedienen
R3	5.1.2	UP3	Het aantal parkeerplaatsen dat is aangesloten op de ParkeerHeer zal zo groot mogelijk zijn
R3-a	5.1.2	UP3, VO	De ParkeerHeer zal tot 180.000 parkeerplaatsen kunnen bedienen
R3-b	5.1.2	UP3, VO	De ParkeerHeer zal dusdanig worden ingericht dat het door hardware-uitbreiding tot 9.000.000 parkeerplaatsen kan bedienen

ID	Uitleg	Volgt uit	Eisen: Aanpasbaarheid
R4	5.1.2	UP4	De ParkeerHeer zal gemakkelijk aangepast kunnen worden om extra functionaliteit te ondersteunen
R4-a	5.5.3	Q14, Q15, Q16	De clients van de ParkeerHeer zullen zo weinig mogelijk business logic bevatten

ID	Uitleg	Volgt uit	Eisen : Communicatie
R5	5.4	R2,R3,R4 ,VO	De ParkeerHeer zal een centrale server die haar diensten aan verschillende type clients beschikbaar stelt
R5-a	5.4	R2,R3,R4 ,VO	De ParkeerHeer zal met de bestuurders communiceren via een navigatiesysteem
R5-b	5.4	R2,R3,R4 ,VO	De ParkeerHeer zal met parkeergaragesystemen kunnen communiceren
R5-c	5.4	Q12,Q13	De communicatie tussen de ParkeerHeer en haar clients zal via het internet verlopen
R5-d	5.4	Q12,Q13	De ParkeerHeer zal op een veilige manier met haar clients communiceren
R5-e	5.5	Q14,Q15, Q16	De ParkeerHeer zal communiceren met haar clients via asynchrone communicatie

Gebruikte termen

Tussen haakjes de paragraaf of het hoofdstuk waar de term wordt uitgelegd

- Dynamisch Parkeerverwijssysteem (1)
- Zoekend verkeer (2)
- Zoektijd (2)
- Weigering (2)
- Software Architectuur (3)
- Design (3)
- Ontwerp (3)
- Business Modelling discipline (3.2)
- Requirements / eisen discipline (3.2)
- Analysis & Design / Ontwerp discipline (3.2)
- QOC - Questions, Options, Criteria (3.3)
- Ontwerpvragen (3.3)
- Opties (3.3)
- Criteria (3.3)
- Ontwerpbeslissing (3.3)
- Ontwerpruimte (3.3)
- Perspectief / Viewpoint (3.4.2)
- Schaalbaarheid (5.1)
- Performance (5.1)
- Authenticeren (5.3)
- Client-Server architectuur (5.4)
- Protocol (5.4)
- Scope (5.4)
- (a)synchrone communicatie (5.5)
- Business Logic (5.5)
- Use Case / Scenario (6)
- Epoch (7.1)
- Bottleneck (8)

- Adapter (8)
- API - Application Programming Interface (8)
- Loadbalancer (9)
- Plugin (12.4)

Referenties

[1] Barbara Paech Allen H. Dutoit. Rationale management in software engineering. 2003.

[2] Martijn van Noort Bart van Arem, Ben Jansen. Slimmer en beter- de voordelen van intelligent verkeer. 2008.

[3] CPB. Mobiliteit auto's in Nederland. 2008.

[4] Stedenbouw en Architectuur. Mobiliteit auto's in nederland. 2008.

[5] Ir. Oei Hway-liem. Mogelijke veiligheidseffecten van navigatiesystemen. 2003.

[6] Philippe Kruchten. Architectural blueprints - the "4+1" view model of software architecture. *IEEE Software 12 (6)*, 12 (6):42–50, 1995.

[7] Philippe Kruchten. *The Rational Unified Process: An Introduction.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[8] Philippe Kruchten. What is the rational unified process? 2003.

[9] Rick Kazman Len Bass, Paul Clements. *Software Architecture in Practice (2nd edition).* 2003.

[10] Michael Tong Sang. Technisch ontwerp parkeertomtom. 2008.

[11] Andrew S. Tanenbaum and Maarten Van Steen. *Distributed Systems: Principles and Paradigms.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2001.