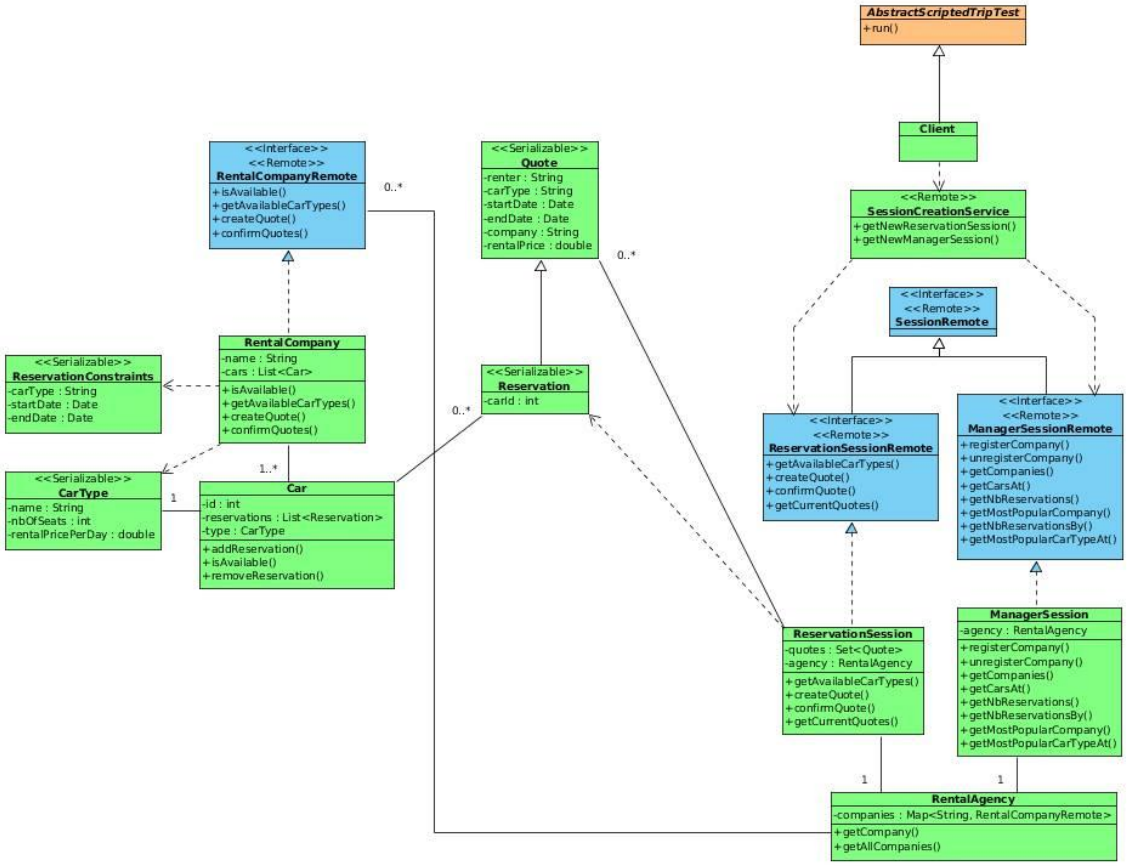
Ontwerp gedistribueerd Java RMI systeem

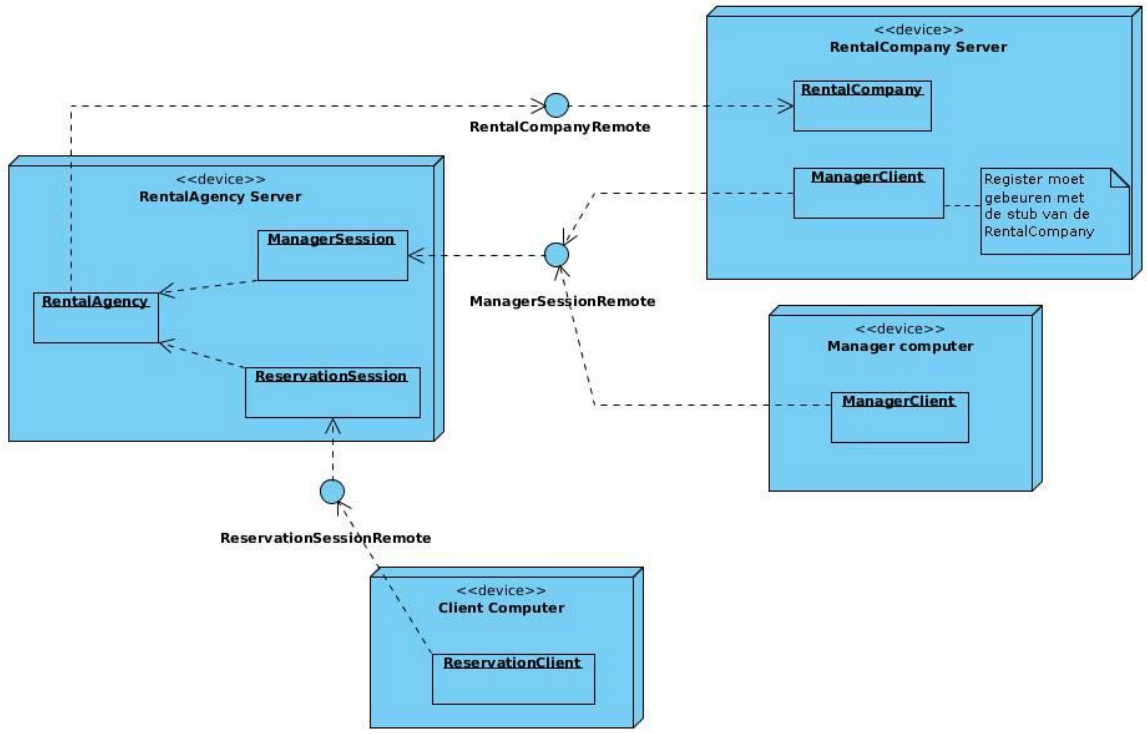
Toon Nolten & Stijn Hoskens

Klassediagram



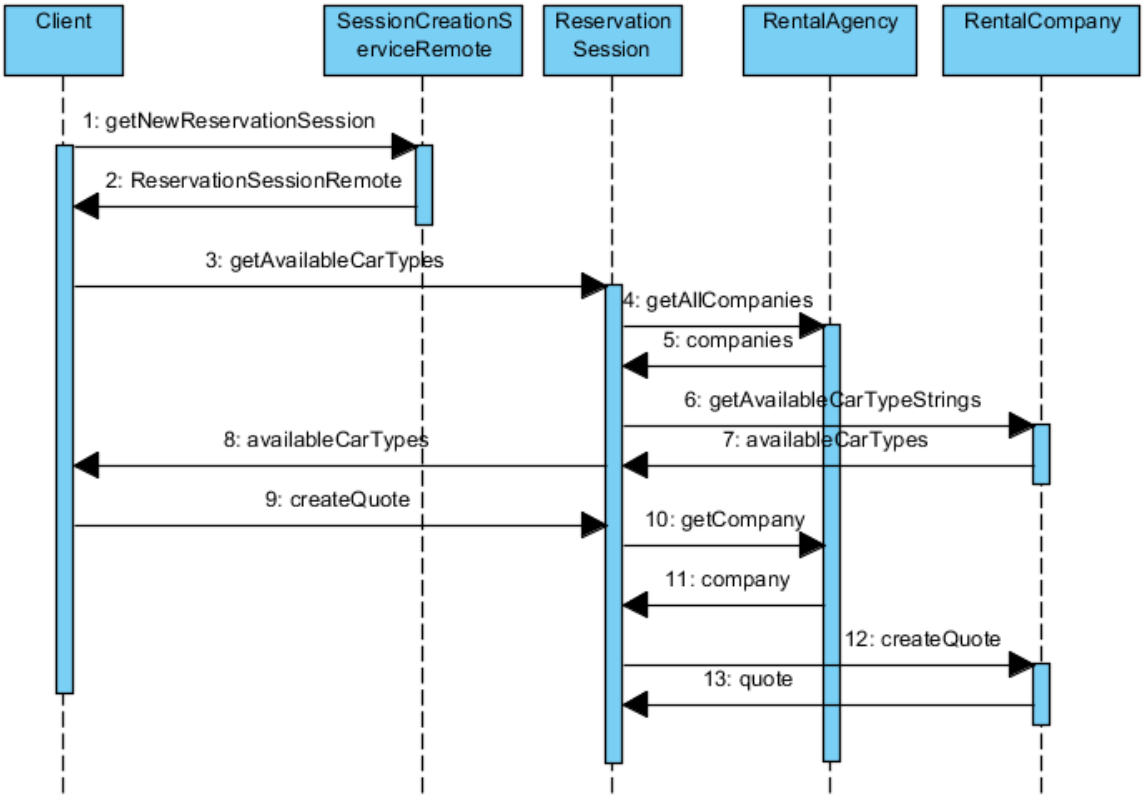
In bovenstaand klassediagram ziet u links de RentalCompany zelf, zoals ook gebruikt in de voorgaande sessie. Rechts ziet u de uitbereiding. Bovenaan is de client te zien die aan de hand van verschillende sessies oproepen doet aan het RentalAgency, die dan weer de vragen doorspeelt aan de desbetreffende RentalCompany.

Deployment diagram

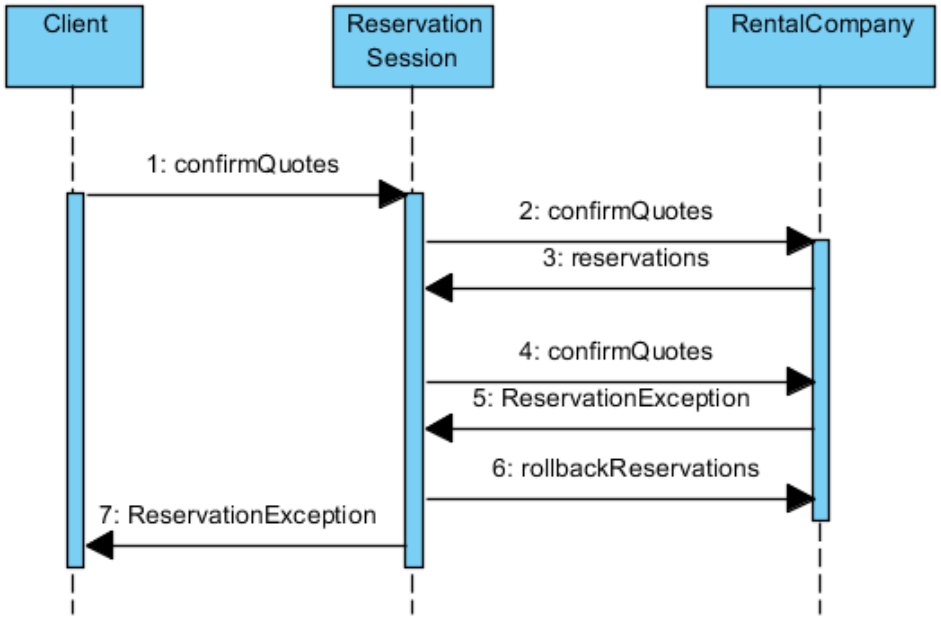


Hierboven is te zien hoe de verschillende componenten op verschillende apparaten draaien. Uiterst links is het RentalAgency te zien. Een ReservationClient is onderaan te zien, die zal via een ReservationSession verbinding maken met RentalAgency. Rechts bovenaan zien we een voorbeeld van een server van een rental company. Deze heeft als belangrijkste klasse de RentalCompany klasse, maar hij heeft ook een ManagerClient nodig om zichzelf te kunnen registreren en eventueel verwijderen uit het agentschap. Rechts in het midden is nog een afzonderlijke manager te zien. Deze kan bijvoorbeeld statistieken opvragen. In de implementatie is de ManagerClient en ReservationClient gebundeld in één client, om zo deze van de scripted test te laten overerven. Bij onze ontwerpfase vonden we dat deze beter gescheiden gehouden worden, aangezien ze verschillende verantwoordelijkheden hebben, en ook verschillende sessies bijhouden.

Sequentiediagram

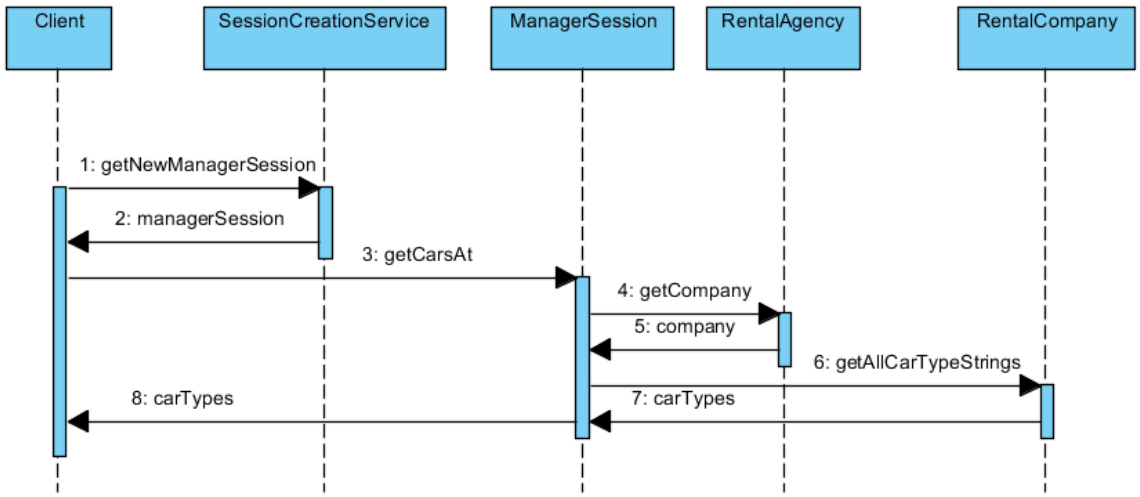


Op bovenstaand sequentiediagram is te zien hoe een client een quote kan aanmaken. Eerst bekijkt hij of hij een auto kan huren, vervolgens wordt de quote opgeslagen in de sessie.



Een verzameling quotes kan dan uiteindelijk bevestigd worden, hierboven is te zien wat er gebeurd wanneer een quote niet bevestigd kan worden. ConfirmQuotes wordt aangeroepen op de ReservationSession, deze doet dan confirmQuotes bij alle bedrijven waar een lopende bestelling bij is geplaatst. Wanneer het dan bij een bedrijf mislukt, wordt de sessie gewaarschuwd door een

exception, vervolgens doet de sessie een rollback bij alle companies en gooit hij de exception voort naar de client. Niets is dan gereserveerd.



Bovenstaande figuur stelt het gebruik van de manager-functies voor, met als voorbeeld het opvragen van de types autoin een welbepaald bedrijf. De ManagerSession wordt aangemaakt, vervolgens wordt aan de RentalAgency het desbetreffende bedrijf opgevraagd, en daaraan wordt dan de autotypes gevraagd.

Ontwerpbeslissingen

Het gedistribueerd systeem bestaat voornamelijk uit drie bestanddelen, RentalAgency, Clientsen de RentalCompanies. Via de Client kan men van bepaalde auto's opvragen of ze beschikbaar zijn en ze daarna reserveren,er kunnen bedrijven toegevoegd en verwijderd worden, verschillende statistieken kunnen opgevraagd worden, zoals gespecifieerd in de opgave. Eerst dachten we aan een opsplitsing tussen een ReservationClient en een ManagerClient, aangezien de verschillende verantwoordelijkheden,de bedoeling van de client is echter het testen van de rest dus hebben we geen aparte clients geïmplementeerd.RentalAgency zal op zijn beurt een map bijhouden van Strings met de bijhorende bedrijven, dit zal dan dienst doen als de zogenaamde Naming Service. Wanneer clients een beroep doen op de RentalAgency, zal deze de vraag doorspelen aan de verschillende RentalCompanies om zo een volledig beeld te scheppen van alle reservaties in alle verschillende verhuurbedrijven. Een RentalCompany is hetzelfde opgebouwd als in de vorige opgaves, zoals te zien is op het klassediagram. Deze bezit auto's en kan ze zodoende ook verhuren.

De communicatie tussen de verschillende systemen, gebeurt aan de hand van drie interfaces, ReservationSessionRemote, ManagerSessionRemote en RentalCompanyRemote. In de eerste interface worden de methodes aangeroepen door de Client, de implementatie van de methodes en dus de uitvoer gebeurt aan de kant van RentalAgency door een klasse ReservationSession. De client communiceert ook via een soortgelijke interface ManagerSessionRemote met RentalAgency, die een implementatie biedt door middel van ManagerSession. De rental agency roept verschillende methodes op van de RentalCompany door middel van RentalCompanyRemote, de interface die RentalAgency tot zijner beschikking heeft.
Er zijn vier geserializeerde klassen (waarbij één van een ander overerft), Quote,Reservation, CarType en ReservationConstraints. De klasse Quote is als het ware een poging tot reservatie, de clientzal deze doorgestuurd krijgen van RentalAgency, die deze op zijn beurt krijgt van RentalCompany. Wanneer deze poging succesvol is, is de Quote ingevuld en wordt deze een Reservation, met een carId. Deze wordt dan ook doorgestuurd over de verschillende systemen. Ook CarTypes kunnen zo

worden doorgestuurd, ze worden immers opgevraagd voor statistieken. ReservationConstraints worden daarentegen de andere kant opgezonden. Ze vertrekken van bij de rental agency en zijn de beperkingen waaraan de te maken reservatie moet voldoen.

De klassen ReservationSession en ManagerSession, die een implementatie bieden voor de respectievelijke Remote interfaces, zijn beide actief aan de zijde van de RentalAgency. Deze bieden namelijk methodes aan die via het verhuur-agentschap lopen.

Het enige object dat in de rmiregistry geregistreerd moet worden is de SessionCreationService. Op die manier kan iedereen aan zijn ReservationSession-, respectievelijk ManagerSession-(stubs) komen. SessionCreationService is dus onze vervanger van een 'Naming Service' maar is eigenlijk alleen een aanspreekpunt om nieuwe sessions te krijgen, een bestaand object kan zo niet teruggevonden worden. Het derde remote-object meer precies de RentalCompany-stubs worden door een 'manager' met een ManagerSession(Remote) geregistreerd en onthouden door de RentalAgency.

De belangrijkste Session die life cycle management behoeft, is de ReservationSession. Deze bezit immers een soort van winkelmandje met daarin verschillende Quotes, die op het einde van de Session afgerekend, of eventueel afgebroken moet worden. De ReservationSession bevindt zich op elk moment van de uitvoering in een bepaalde staat, die bijgehouden moet worden. Dit gebeurt eenvoudigweg met een set van Quotes. Bij aanmaak is deze set leeg, en zal ze door de ReservationClient langzaamaan gevuld worden. Wanneer de rekening wordt gemaakt, ook al is ze niet succesvol, verdwijnt de referentie naar de Session, en zal deze worden verwijderd. Indien men terug wilt reserveren, wordt een nieuwe ReservationSession aangemaakt. ManagerClient heeft een soortgelijke Session, alleen bevindt deze zich niet in een verschillende staat op verschillende plaatsen van uitvoering. En dus is het van minder belang om deze telkens opnieuw aan te maken. Een ManagerSession zal blijven bestaan zolang de ManagerClient bestaat. Bij de RentalCompany ligt de zaak dan weer anders. Deze bevindt zich ook steeds in een andere staat. De referenties naar die bedrijven worden aangemaakt doordat de ManagerClient de benodigde informatie doorspeelt naar RentalAgency. Zolang het bedrijf bestaat, zal de RentalAgency een referentie hebben naar die remote interface, en dingen kunnen opvragen. Bij het uitschrijven zal de referentie verdwijnen.

Synchronizatie is nodig bij het reserveren van een auto. Stel, iemand vraagt een bepaalde auto op, deze is beschikbaar en dus plaatst hij deze Quote in zijn winkelmandje. Persoon twee doet hetzelfde en kan dit aangezien de auto nog niet concreet verhuurt is. Persoon twee bevestigd zijn Quote samen met de eerste. Gevolg: zonder synchronizatie kan dit ertoe leiden dat ze beide in de waan zijn dat de reservatie gelukt is. Hetzelfde kan gebeuren wanneer iemand vraagt of een auto beschikbaar is, samen met iemand die de reservatie van diezelfde auto bevestigt. Door deze methodes gesynchronizeerd te maken, zal steeds de ene na de andere uitgevoerd worden, en zal dit probleem dus vermeden worden. Dit wordt gedaan in de RentalCompany, wanneer de Quotes worden bevestigd. Het ongedaan maken van reservaties is eveneens gesynchronizeerd gemaakt.