Aggregaties en dynamische modellen

Bas Kruiswijk

Afstudeerscriptie

Rijksuniversiteit Leiden
Vakgroep Informatica
Mei 1997

Begeleider: Prof. Dr. G. Engels

1 Inleiding

Twee belangrijke onderdelen van een object-georiënteerd model zijn het object model en het dynamisch model. Het object model beschrijft de structuur van de objecten in een systeem en het geeft weer welke relaties er tussen objecten kunnen bestaan en welke attributen en operaties er zijn. Het object model beschrijft de statische structuur van het systeem. Het dynamisch model daarentegen modelleert de veranderingen op de objecten en hun relaties in de tijd. In het dynamisch model wordt middels een toestandsdiagram vastgelegd in welke volgorde de operaties van een object kunnen worden aangeroepen.

Het dynamisch model bevat een toestandsdiagram voor elke class in het object model. Dat betekent dat het dynamisch model een nauwe relatie heeft met het object model. Wanneer er tussen twee classes een inheritance- of whole-part-relatie in het object model is gedefinieerd, dan ligt er tussen de bijbehorende toestandsdiagrammen ook een bepaalde relatie.

In [EbeEng 95] is onderzocht welke beperkingen gelden voor de toestandsdiagrammen, wanneer er sprake is van een inheritance-relatie. Deze beperkingen zijn geformaliseerd door een homomorfisme te definiëren dat de toestanden in het toestandsdiagram van het subtype afbeeld op dat van het supertype. Als zo'n homomorfisme gedefinieerd kan worden, dan voldoen de toestandsdiagrammen aan de beperkingen.

Als vervolg op dit onderzoek is het interessant om te onderzoeken of er ook een dergelijke relatie bestaat tussen de toestandsdiagrammen van objecten in een aggregatie (whole-part relatie).

1.1 Aanpak

Aggregaties kunnen op verschillende manieren worden toegepast. Daarom is er voor gekozen om als eerste stap de verschillende vormen van aggregaties nader te onderzoeken en deze in te delen naar de verschillende manieren waarop aggregaties gebruikt kunnen worden.

Op basis van deze indeling wordt in hoofdstuk 2 een classificatie van aggregaties gemaakt. Bij elk type aggregatie wordt aan de hand van een voorbeeld afgeleid wat de relatie is tussen de classes in het object model en de bijbehorende toestandsdiagrammen in het dynamisch model.

De tweede stap in de aanpak is, om dit te vergelijken met de relatie tussen classes in het geval van inheritance. Aangezien zowel inheritance als aggregaties methoden zijn om hergebruik te realiseren, is het mogelijk dat er overeenkomsten zijn met de resultaten uit het onderzoek in [EbeEng 95]. Hoofdstuk 3 legt de resultaten van dat onderzoek naast de conclusies die in het tweede hoofdstuk zijn getrokken.

De laatste stap is om de in hoofdstuk 2 en 3 gevonden afhankelijkheden tussen toestandsdiagrammen van classes in een aggregatie formeel te beschrijven.

2 Classificatie

In dit hoofdstuk wordt een aantal verschillende typen aggregaties onderscheiden. Elk type kent een aantal beperkingen en specifieke eigenschappen. Op basis van deze specifieke eigenschappen wordt bekeken wat de afhankelijkheden en beperkingen zijn in het dynamisch model van de betrokken classes in de aggregatie.

2.1 Inleiding

2.1.1 Algemeen

Om te komen tot een indeling van aggregaties wordt gebruik gemaakt van de indelingen in [CoYo 91a/b] en [RuBlPr 91].

In [CoYo 91a/b] worden aggregaties ingedeeld in drie typen:

- Assembly (Bestaat uit)
- Container (Bevat)
- Collection (Behoort tot)

In [RuBlPr 91] wordt de volgende indeling genoemd:

- Fixed aggregate
- Variable aggregate
- Direct recursive aggregate
- Indirect recursive aggregate

Daarnaast worden in [GaHeJo 95] een aantal design patterns beschreven, die op verschillende manieren gebruik maken van aggregaties. Met name het Composite pattern is een interessant voorbeeld van het gebruik van aggregaties.

Door deze indelingen te combineren ontstaat een classificatie van aggregaties waarbij elk type een aantal specifieke eigenschappen heeft. Assembly wordt gecombineerd met Fixed aggregate, Collection wordt gecombineerd met Variable aggregate en Indirect recursive aggregate wordt gecombineerd met het Composite pattern. Op deze manier ontstaat de volgende indeling:

- Assembly (Fixed aggregate)
- Collection (Variable aggregate)
- Container
- Recursion (Direct recursive aggregate)
- Composition (Indirect recursive aggregate)

Elk van deze typen zal steeds op dezelfde manier beschreven worden. Eerst wordt globaal de structuur geschetst. Vervolgens wordt een beschrijving gegeven van de kenmerken van het betreffende type aggregatie. In de beschrijving van de kenmerken wordt met name aandacht besteed aan de wijze waarop de parts geïnstantieerd worden en de wijze waarop de operaties van de objecten in de aggregatie worden aangeroepen.

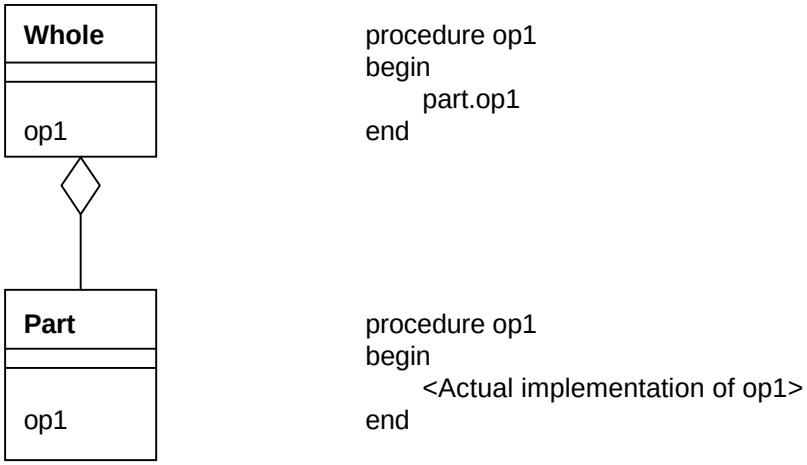
Vervolgens wordt nader ingegaan op het dynamisch model van de objecten in het betreffende type aggregatie. Hier zal worden ingegaan op de relatie tussen het dynamisch model van de parts en dat van de whole.

2.1.2 Schematechniek

Als schematechniek voor object modellen en dynamische modellen zal gebruik worden gemaakt van de OMT-notatie [RuBlPr 91]. Als aanvulling op deze techniek zal voor toestandsdiagrammen ook gebruik worden gemaakt van Harel's statechart notatie [Har 87].

2.1.3 Delegation

Een aantal typen aggregaties is gebaseerd op het gebruik van delegation. Delegation betekent dat een operatie niet rechtstreeks in de betreffende class geïmplementeerd is, maar wordt uitbesteed aan een andere class. De operatie bestaat dan uitsluitend uit een aanroep van een operatie van een ander object. In de verschillende typen aggregaties die hieronder beschreven worden zal het steeds gaan om delegation van de whole naar één van de parts. Dit wordt in het onderstaand voorbeeld verduidelijkt.



Het is niet noodzakelijk dat beide operaties dezelfde naam hebben. Voor de duidelijkheid worden echter meestal dezelfde namen aangehouden. Het is wel van belang dat de body van de delegerende operatie precies één aanroep van een andere operatie bevat.

Het is ook van belang op te merken dat hier wordt ingegaan op de inhoud van een operatie. De inhoud (functionaliteit) van de operaties maakt deel uit van het functioneel model. In het onderzoek naar de relatie tussen inheritance en dynamisch model [EbeEng 95] heeft men zich kunnen beperken tot het object model en het dynamisch model. Wat de operaties betreft wordt daarin vastgelegd welke operaties tot welke class behoren (object model) en in welke volgorde deze operaties aangeroepen mogen worden (dynamisch model). Zoals later zal blijken is delegation echter van wezenlijk belang voor de relatie tussen de classes in een aggregatie en hun dynamisch model.

2.1.4 Correctheid

In de beschrijving van het dynamisch model wordt afgeleid wanneer het toestandsdiagram van de whole in een aggregatie correct is in relatie tot de toestandsdiagrammen van de parts. Onder correctheid wordt hier verstaan dat het toestandsdiagram van de whole niet conflicteert met één van de toestandsdiagrammen van de parts. Wanneer delegatie wordt toegepast kan het zijn, dat de aanroep van een operatie in de whole, de aanroep van een operatie in een part tot gevolg heeft. Er is sprake van een conflict wanneer er een bepaalde reeks van achtereenvolgens aan te roepen operaties is toegestaan volgens het toestandsdiagram van de whole, maar niet in het toestandsdiagram van het part.

2.1.5 Operaties en events

In principe bestaan er twee soorten operaties: activiteiten en acties. Een activiteit is een operatie die wordt uitgevoerd gedurende de tijd dat een object in een bepaalde toestand is. Een actie is een operatie die wordt uitgevoerd bij een toestandsovergang. Zo'n toestandsovergang wordt veroorzaakt door het optreden van een event in een bepaalde toestand. Een event is een informatieoverdracht van een object naar een ander object. Gezien vanuit het zendende object wordt dit ook wel een message genoemd.

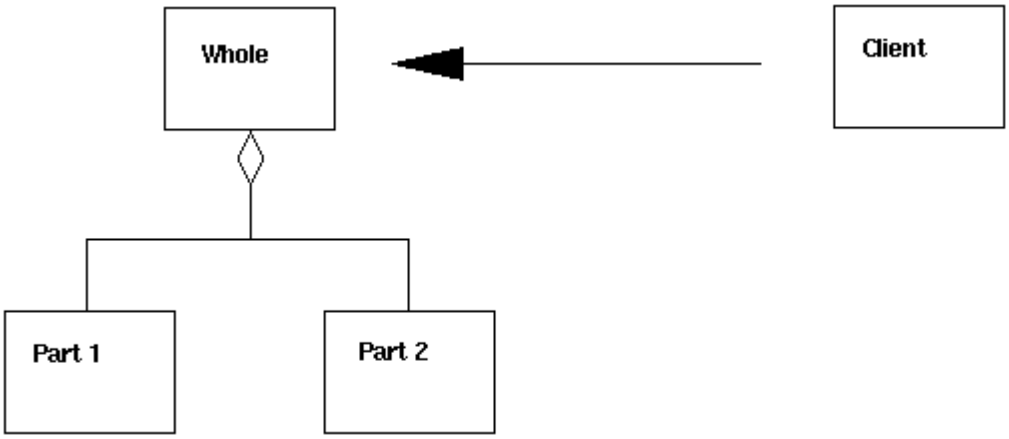
Het dynamisch model is vooral gericht op het modelleren van de volgorde van operaties (acties) in reactie op externe stimuli. Daarom wordt in toestandsdiagrammen vaak zowel het event als de operatie (actie) genoemd. In OMT wordt dit gedaan in de vorm "event / actie".

We zijn echter met name geïnteresseerd in de betrokken operaties en zullen daarom in de toestandsdiagrammen uitsluitend de operaties vermelden bij de toestandsovergangen. Het object dat het bijbehorend event veroorzaakt wordt client genoemd en in plaats van te zeggen dat de client een event veroorzaakt dat leidt tot de aanroep van een operatie, zullen we zeggen dat de client de operatie aanroept.

In de nu volgende paragrafen wordt elk type aggregatie afzonderlijk beschreven.

2.2 Assembly (Fixed Aggregate)

2.2.1 Structuur



2.2.2 Beschrijving

In dit type aggregatie zijn het aantal en de typen van de parts vast en statisch gedefinieerd. De instantiatie van de parts vindt gelijktijdig met de instantiatie van de whole plaats. Bovendien kunnen de instanties van de parts niet zonder de whole bestaan. De parts zijn dus vergelijkbaar met weak entities [Chen 76].

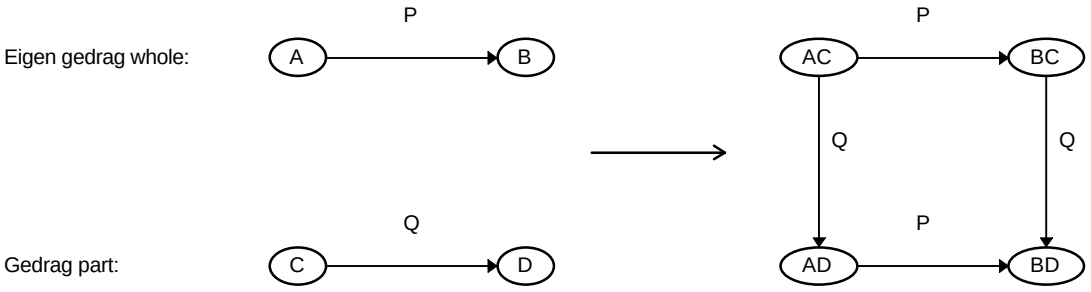
De whole voorziet in de interface naar de clients toe. Clients kunnen niet rechtstreeks operaties van één van de parts aanroepen. Op deze manier heeft de buitenwereld geen kennis nodig over de parts waaruit de whole is opgebouwd, en kan de aggregatie als een enkel object worden beschouwd. Operaties in de whole behoren tot het eigen gedrag van de whole of worden gedelegeerd naar één van de parts.

2.2.3 Dynamisch model

De whole biedt de interface voor de gehele aggregatie. Dit betekent dat alle operaties die een client kan aanroepen in de whole gedefinieerd zijn en dus onderdeel uitmaken van het dynamisch model (het toestandsdiagram) van de whole. Een deel van deze operaties wordt echter gedelegeerd naar één van de parts. Elk van de parts implementeert dus een deel van het gedrag van de whole en daarnaast wordt een deel van het gedrag rechtstreeks in de whole geïmplementeerd. Het toestandsdiagram van elk van de parts heeft dus een nauwe relatie met het toestandsdiagram van de whole.

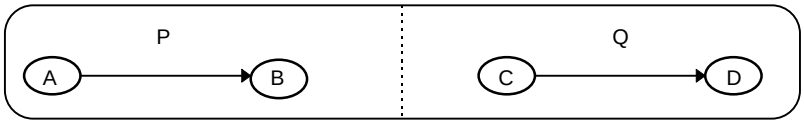
Wanneer we onderscheid maken tussen de operaties die rechtstreeks in de whole geïmplementeerd zijn en de operaties die gedelegeerd worden naar één van de parts, dan zijn er eigenlijk een aantal onafhankelijke toestandsdiagrammen. Een toestandsdiagram voor elk van de parts en het toestandsdiagram van het eigen gedrag van de whole. Het toestandsdiagram van de whole (de gehele aggregatie) is dan de logische AND van deze toestandsdiagrammen.

Dit toestandsdiagram van de whole kan worden geconstrueerd door het cartesisch produkt te nemen van de toestanden en toestandsovergangen. Schematisch komt dit op het volgende neer.



Er kunnen eventueel beperkingen worden toegevoegd door toestandsovergangen te verwijderen. De toestandsovergang Q tussen de toestanden BC en BD kan worden verwijderd. Hiermee wordt dan aangegeven dat een overgang van C naar D in het gedrag van het part alleen maar kan plaatsvinden als whole in toestand A is.

Door gebruik te maken van Harel's statechart notatie kan het toestandsdiagram van de whole ook worden weergegeven als een aantal parallelle toestandsdiagrammen. Schematisch kan dat als volgt worden weergegeven:



Beperkingen kunnen dan eenvoudig worden weergegeven door die bij de toestandsovergangen op te nemen. De beperking dat een toestandsovergang van C naar D alleen kan plaatsvinden als de whole in toestand A is, kan nu eenvoudig worden toegevoegd door bij toestandsovergang Q de beperking [whole in A] op te nemen.

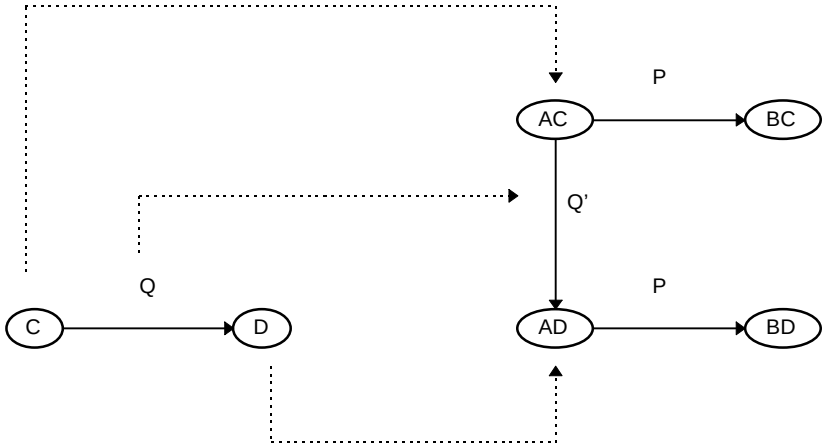
Dit is een constructieve definitie van het toestandsdiagram van de whole. Het moet echter ook mogelijk zijn om gegeven een compleet toestandsdiagram van de whole, te bepalen of dit correct is gegeven de toestandsdiagrammen van de parts. Dat is dan een descriptieve definitie.

Voor zo'n descriptieve definitie moet eerst worden bekeken hoe het toestandsdiagram van het part is ingebed in dat van de whole. Vervolgens moet worden bekeken of er conflicten kunnen ontstaan gegeven de relatie die er ligt tussen de toestandsdiagrammen.

De inbedding van het toestandsdiagram van het part in dat van de whole kan eenvoudig worden weergegeven door de gedelegeerd operaties te relateren aan de corresponderende operaties in de whole en vervolgens de bijbehorende begin- en eindtoestanden aan elkaar te relateren.

Wanneer Harel's statechart notatie wordt gebruikt en de whole op de hierboven beschreven manier wordt geconstrueerd, dan geldt de inbedding automatisch. De toestandsdiagrammen van de parts zijn immers rechtstreeks overgenomen in het toestandsdiagram van de whole. In het voorbeeld waarbij de whole is geconstrueerd als cartesisch produkt ligt de inbedding minder voor de hand, zeker wanneer beperkingen worden toegevoegd door toestandsovergangen te verwijderen.

Hieronder wordt dit schematisch weergegeven.

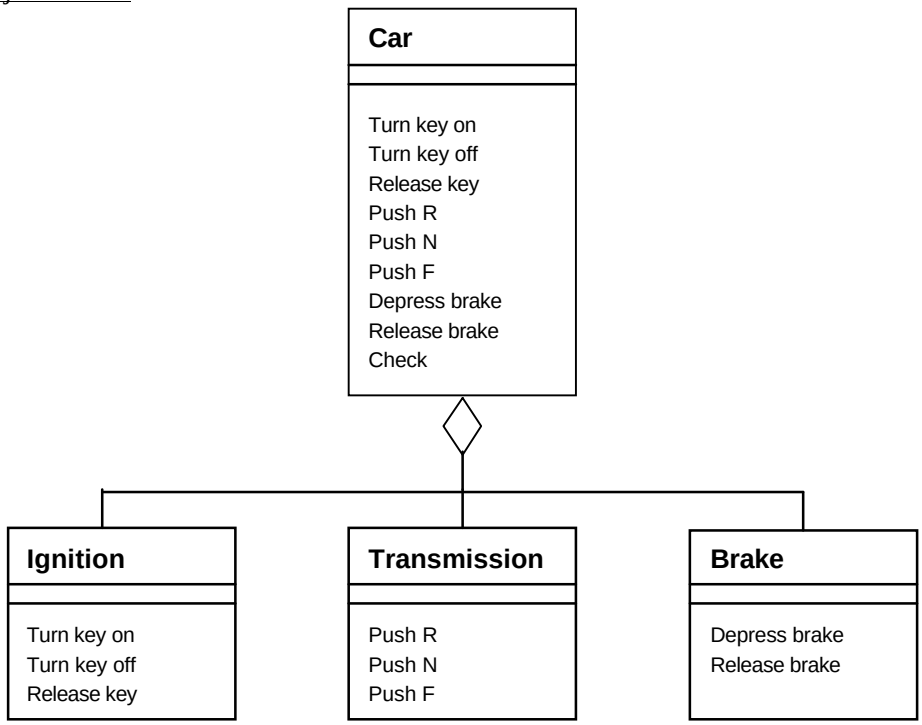


Als deze inbedding gedefinieerd is, kan worden bekeken of het toestandsdiagram van de whole correct is in relatie tot het toestandsdiagram van het part. Het moet nu zo zijn dat een toestandsovergang van AC naar AD (direct of indirect) in de whole ook altijd (middels delegatie) een toestandsovergang van C naar D in het part tot gevolg heeft.

2.2.4 Voorbeeld

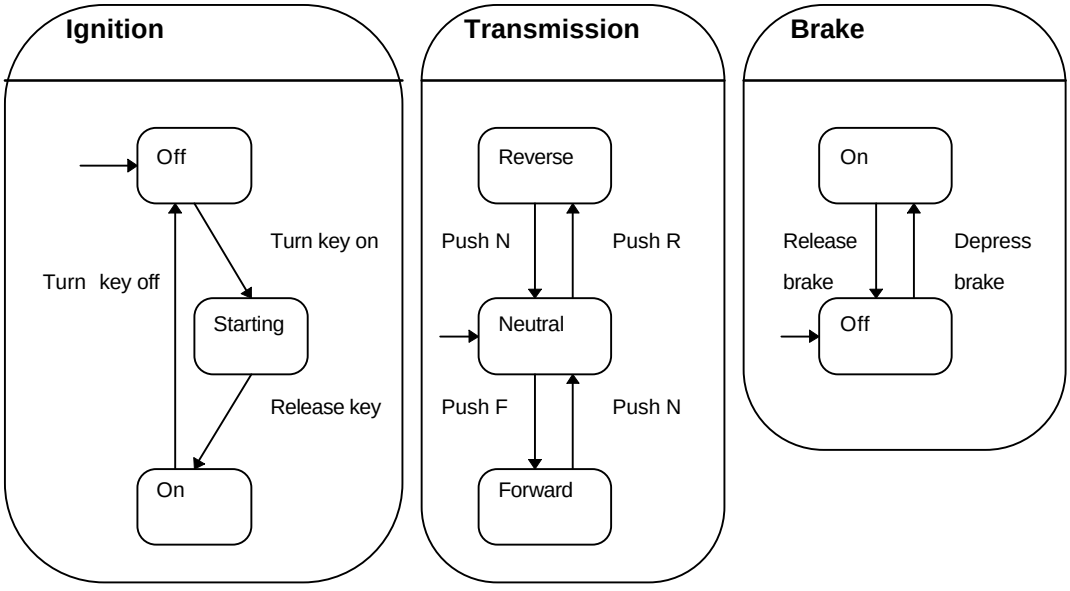
In dit voorbeeld wordt een auto beschreven, bestaande uit de onderdelen contact (ignition), versnelling (transmission) en rem (brake). Alle operaties van deze drie onderdelen komen ook voor in de whole. Deze operaties worden door de whole gedelegeerd naar het betreffende part. Naast deze gedelegeerde operaties bevat de whole ook een operatie die niet wordt gedelegeerd (check). Dat is dus het eigen gedrag van de whole.

Object model:

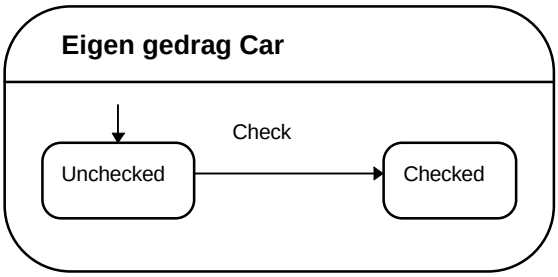


Dynamisch model:

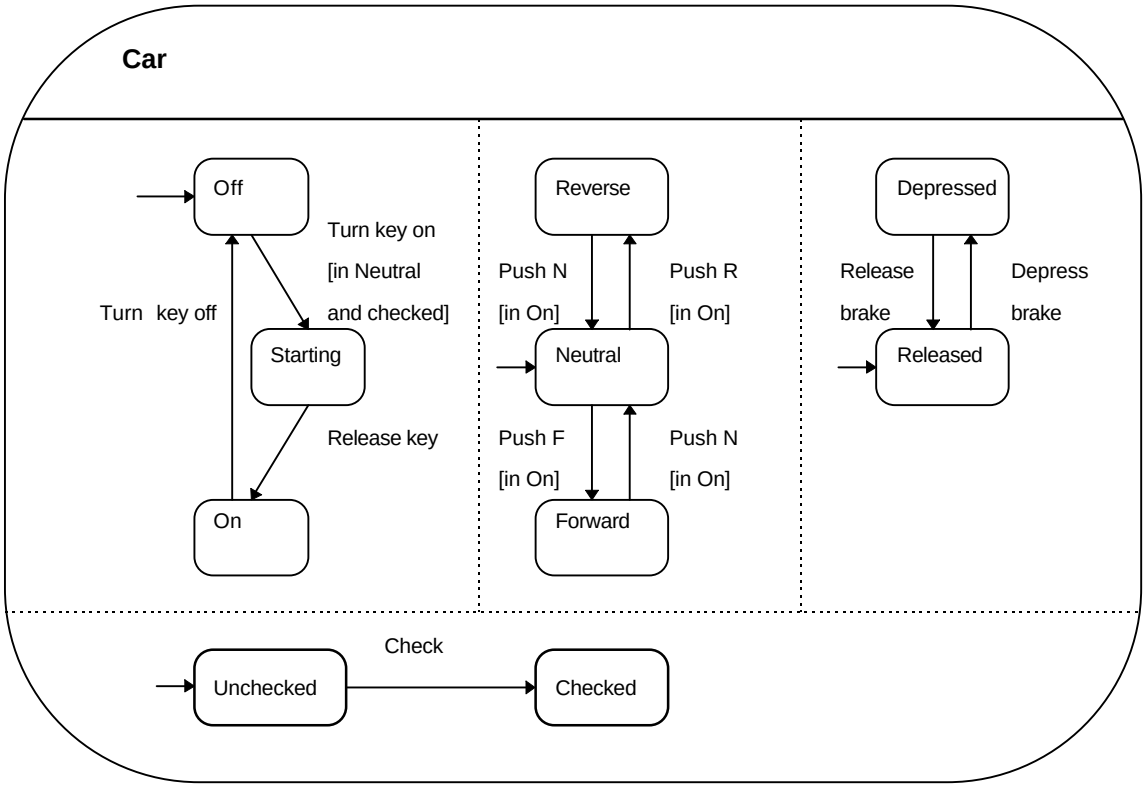
De toestandsdiagrammen van de parts zijn hieronder weergegeven.



Het toestandsdiagram van het eigen gedrag van de whole kan als volgt worden weergegeven:

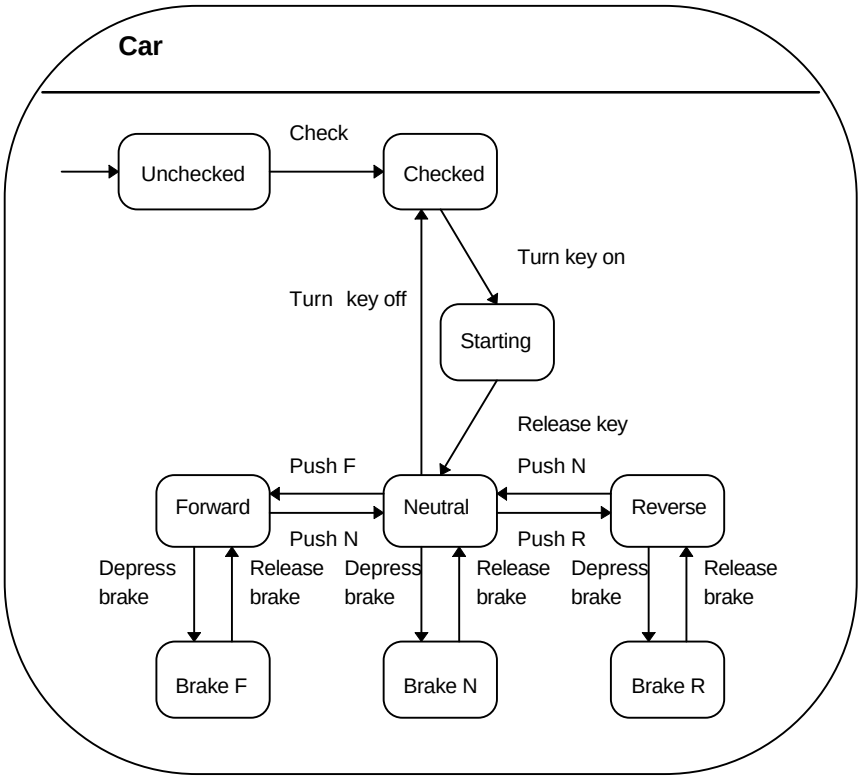


Het complete toestandsdiagram van Car kan nu worden geconstrueerd door de logische AND te nemen van de toestandsdiagrammen van de parts en het eigen gedrag van de whole. Vervolgens kunnen een aantal beperkingen worden toegevoegd, die de onderlinge afhankelijkheden van de toestandsdiagrammen specificeren. In dit voorbeeld is bijvoorbeeld de operatie Turn key on alleen toegestaan wanneer de andere toestandsdiagrammen (afkomstig van Transmission en het eigen gedrag van Car) in de toestanden Neutral en Checked zijn. Dit wordt gespecificeerd door bij de toestandsovergang Turn key on de beperking [in Neutral and Checked] op te nemen.



Het is in nu eenvoudig na te gaan hoe de toestandsdiagrammen van de parts zijn ingebed in dat van de whole. De toestanden en toestandsovergangen van de parts kunnen gewoon één op één worden afgebeeld op die van de whole.

Het is echter ook mogelijk om een nieuw toestandsdiagram van Car te maken. De beperkingen kunnen nu worden gespecificeerd door het aantal toestanden en toestandsovergangen te beperken.



Het is nu minder duidelijk, hoe toestandsdiagrammen van de parts zijn ingebed in het toestandsdiagram van de whole. Ook is minder duidelijk of het toestandsdiagram van de whole wel correct is in relatie tot dat van de parts.

Eerst worden alle toestanden in de toestandsdiagrammen van de parts (Ignition, Transmission en Brake) gerelateerd aan die van de whole.

{Off} → {Checked}

{Starting} → {Starting}

{On, Neutral} → {Neutral}

{Forward} → {Forward}

{Reverse} → {Reverse}

{Released} → {Forward, Neutral, Reverse}

{Depressed} → {Brake F, Brake N, Brake R}

Vervolgens worden de bijbehorende toestandsovergangen gerelateerd aan toestandsovergangen in de whole. In dit voorbeeld zijn de namen van de toestandsovergangen gelijk gehouden, zodat elke toestandsovergang in de parts wordt gerelateerd aan de toestandsovergang met dezelfde naam in de whole.

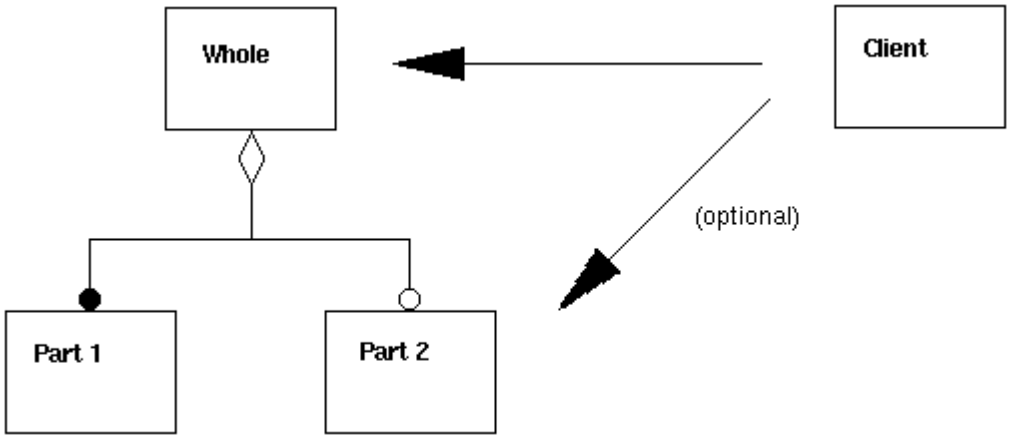
Om te bepalen of het toestandsdiagram correct is in relatie tot dat van de parts, moet worden nagegaan of er conflicten kunnen optreden. Er zou bijvoorbeeld een conflict kunnen optreden met het part Transmission als de toestandsovergang van Reverse naar Neutral in de whole op een andere manier zou kunnen plaatsvinden dan via de operatie Push N. Als dat zou kunnen, dan zou de whole zich in toestand Neutral bevinden, terwijl Transmission zich nog steeds in toestand Reverse bevindt. Als dan vervolgens de operatie Push F wordt aangeroepen is dat toegestaan in het toestandsdiagram van de whole. Maar omdat deze operatie wordt gedelegeerd naar Transmission ontstaat daar een conflict.

Datzelfde geldt bijvoorbeeld voor het part Brake. Een toestandsovergang van Forward, Neutral of Reverse naar Brake F, Brake N of Brake R moet alleen kunnen plaatsvinden door middel van de operatie Depress brake.

In dit voorbeeld treden dit soort conflicten echter niet op doordat het toestandsdiagram van de whole een deelverzameling van het cartesisch produkt van de toestandsdiagrammen van de parts en het eigen gedrag van de whole is. Er zijn geen toestanden of toestandsovergangen in de whole toegevoegd.

2.3 Collection (Variable Aggregate)

2.3.1 Structuur



2.3.2 Beschrijving

In dit type aggregatie is er een variabel aantal parts, met een eindig en vast aantal niveaus. De instantiatie van parts kan zowel gelijktijdig met de whole, als gedurende de gehele levensduur van de whole plaatsvinden. De instanties van de parts kunnen echter niet zonder de whole bestaan. In die zin zijn de parts ook hier vergelijkbaar met weak entities.

Voor wat de interface naar de clients betreft zijn er twee mogelijkheden:

- De whole biedt, net als bij assembly, een interface voor alle operaties. Clients beschouwen de aggregatie als één object. Operaties in de whole behoren tot het eigen gedrag van de whole of worden gedelegeerd naar één van de parts. Om te bepalen naar welk van de parts de operatie gedelegeerd moet worden, wordt door de client een qualifier meegegeven. Deze qualifier dient dus ter identificatie van de parts.
- De whole biedt alleen een interface voor zijn eigen gedrag. Hierin zijn in ieder geval ook operaties opgenomen die de structuur onderhouden (instantiatie van parts). De parts bieden een eigen interface. Hierdoor communiceren clients zowel met de whole als de parts, en dienen dus kennis te hebben over de objecten waaruit de aggregatie bestaat.

Het is uiteraard mogelijk om van deze twee mogelijkheden een mengvorm te hebben. Omdat dit verschil echter zo essentieel is, zal in het vervolg steeds expliciet vermeld worden of het gaat om Collection met of zonder eigen interface van de parts.

2.3.3 Dynamisch model

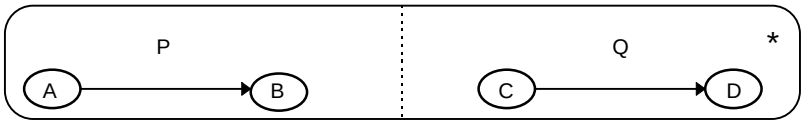
Er is een essentieel verschil tussen de twee mogelijkheden die in de vorige paragraaf genoemd zijn, voor wat betreft het dynamisch model. Daarom worden deze twee apart besproken.

<u>Collection zonder eigen interface van de parts</u>

Wanneer de whole de volledige interface biedt, dan is de situatie vergelijkbaar met assembly. Clients zien de aggregatie als geheel. Er is ook hier een sterke relatie tussen het toestandsdiagram van de whole en dat van de parts. Het feit dat het aantal parts pas tijdens executie van het programma bekend is, is echter een complicerende factor. Wanneer het toestandsdiagram van de whole weer zou worden geconstrueerd als logische AND van de

toestandsdiagrammen van de parts en het eigen gedrag van de whole, dan zou het niet statisch te bepalen zijn. Gedurende de levensduur van een object (tijdens executie van het programma) zou het toestandsdiagram groeien.

Het is mogelijk om de notatie voor toestandsdiagrammen uit te breiden, zodat het toestandsdiagram toch op voorhand te construeren is.



De rechterhelft van dit diagram correspondeert met het toestandsdiagram van het part. Wanneer dit part nu een multipliciteit heeft anders dan 1, dan wordt met het asterisk-symbool (*) aangegeven dat ook het toestandsdiagram meerdere keren kan voorkomen.

Als dit op deze manier gemodelleerd wordt, dan werkt de constructie van het toestandsdiagram van de whole nog op dezelfde manier als bij assembly. Ook de inbedding van de toestandsdiagrammen van de parts in die van de whole en de bepaling van de correctheid blijft hetzelfde.

Collection met eigen interface van de parts

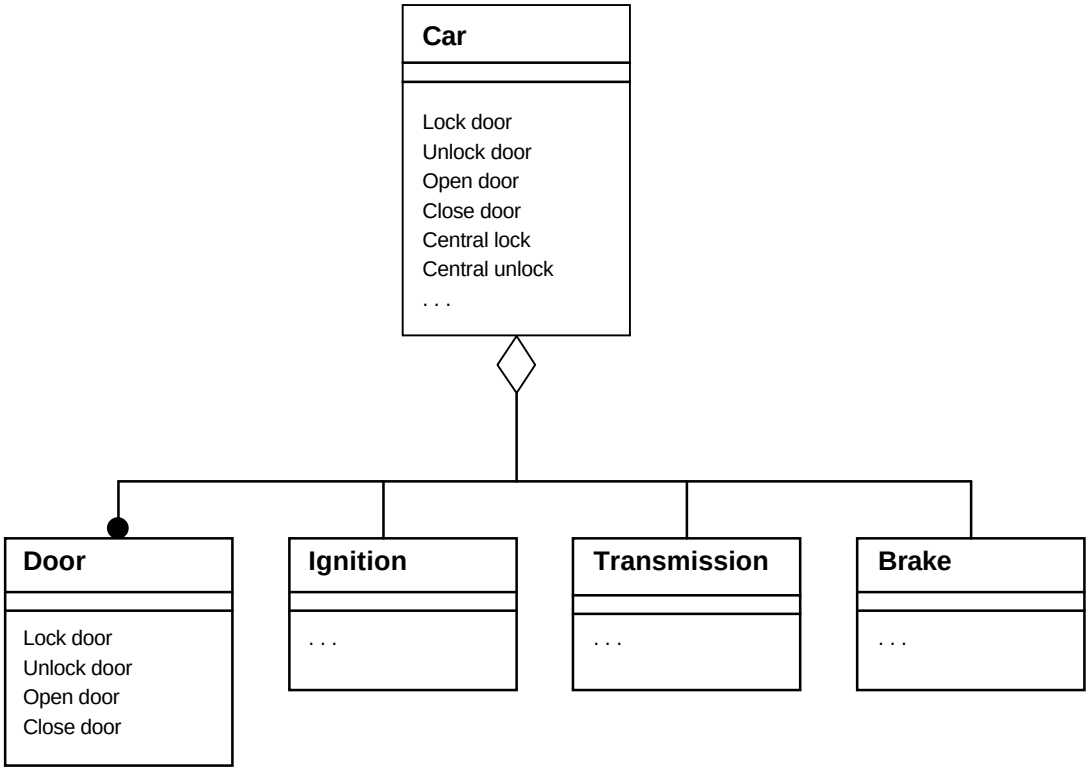
Wanneer de whole niet meer de interface biedt voor het geheel, maar ook de parts een interface bieden, dan wordt de situatie geheel anders. In de parts kan nu een toestandsovergang plaatsvinden, zonder dat er iets in de whole veranderd. In deze situatie zijn de toestandsdiagrammen geheel onafhankelijk.

In zo’n geval is er geen sprake van een relatie tussen de toestandsdiagrammen van de parts en de whole. Dit type wordt daarom verder buiten beschouwing gelaten.

2.3.4 Voorbeeld

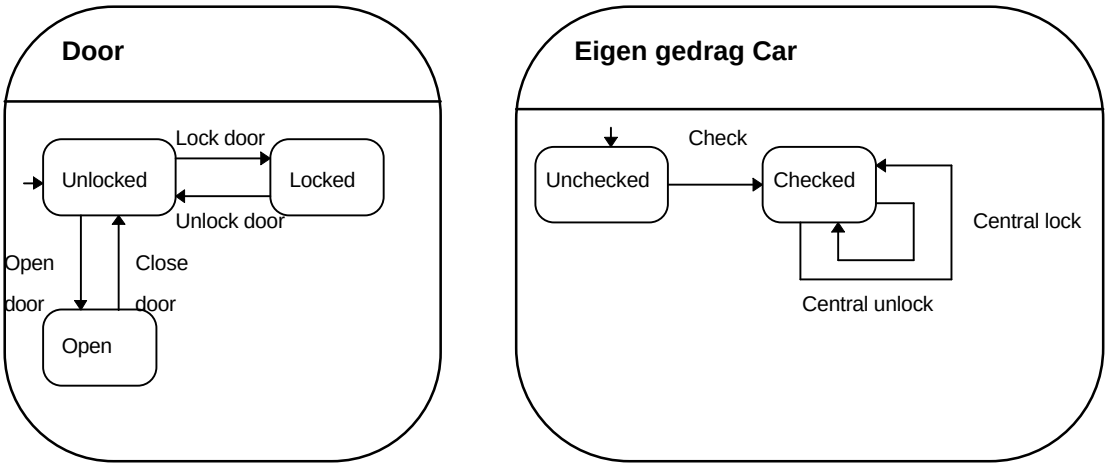
Het voorbeeld uit de vorige paragraaf wordt uitgebreid met het part Door. In tegenstelling tot de andere parts heeft het part Door een variabele multipliciteit. De operaties uit de vorige paragraaf worden hier niet meer genoemd. Dit voorbeeld gaat ervan uit dat de whole de volledige interface biedt.

Object model:

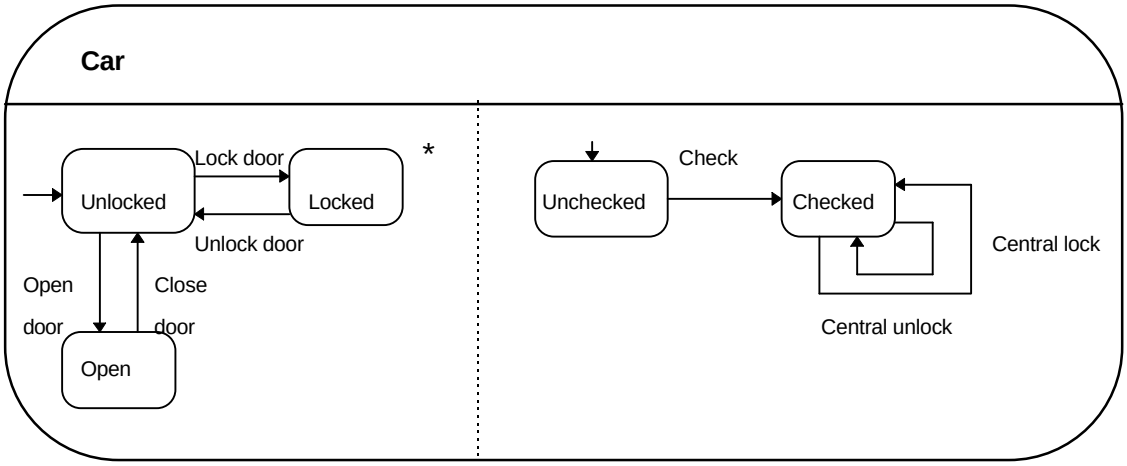


Dynamisch model:

Het toestandsdiagram van Door en het eigen gedrag van Car worden hieronder weergegeven:



Het toestandsdiagram van Car kan nu weer worden geconstrueerd door de logische AND te nemen van het toestandsdiagram van het part en het eigen gedrag van de whole. In dit geval heeft het part een multipliciteit N, zodat het toestandsdiagram van het part van een asterisk (*) moet worden voorzien.

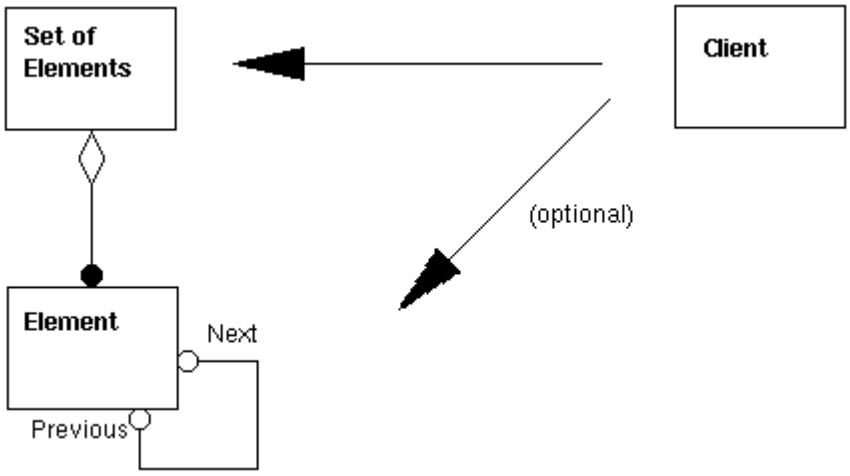


Het is in dit voorbeeld niet mogelijk om één toestandsdiagram te maken voor de whole als geheel. Dit toestandsdiagram zou hoe dan ook afhankelijk zijn van het aantal parts dat er is. Alleen wanneer er een vast aantal parts is, zou het mogelijk zijn.

De inbedding van de toestandsdiagrammen van de parts in die van de whole ligt hier weer voor de hand. De toestanden en toestandsovergangen van het part kunnen gewoon één op één worden afgebeeld op die van de whole.

2.4 Container

2.4.1 Structuur



2.4.2 Beschrijving

Bij dit type aggregatie is er net als bij Collection een variabel aantal parts. Het verschil is dat de parts slechts van één type zijn. Het betreft hier voornamelijk een semantisch verschil. De whole is hier een container-object en bevat een aantal objecten. De instantiatie van de parts kan ook hier zowel gelijktijdig met de whole, als gedurende de gehele levensduur van de whole plaatsvinden. De parts kunnen niet zonder de whole bestaan.

Net als bij Collection is ook hier de interface van de clients naar de parts optioneel. Het container-object kan de interface bieden voor alle elementen. Clients moeten in zo'n geval wel (net als bij Collection) een qualifier meegeven, om het gewenste element te identificeren.

Er is in dit type aggregatie wel een duidelijker onderscheid te maken tussen het eigen gedrag van de whole en dat van de parts. De whole zal operaties bevatten die op de hele verzameling objecten betrekking heeft, terwijl de operaties van de parts op één part betrekking hebben.

2.4.3 Dynamisch model

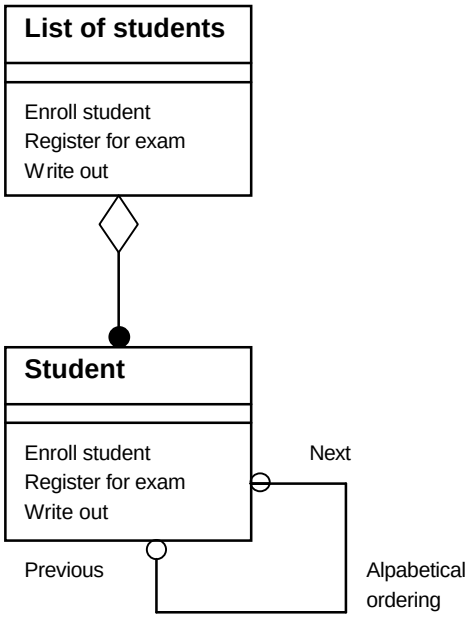
Voor wat het dynamisch model betreft is er weinig verschil met dat van Collection. Ook hier is het een groot verschil of de parts een eigen interface bieden of niet. Wanneer dat niet het geval is, dan kan het toestandsdiagram weer worden opgebouwd uit de logische AND van de toestandsdiagrammen van de parts en het eigen gedrag van de whole. In het geval van Container komt dat dus neer op een repeterend toestandsdiagram van Element en het toestandsdiagram van de whole (container-class).

Wanneer de parts wel een eigen interface bieden, dan zijn de toestandsdiagrammen weer geheel onafhankelijk.

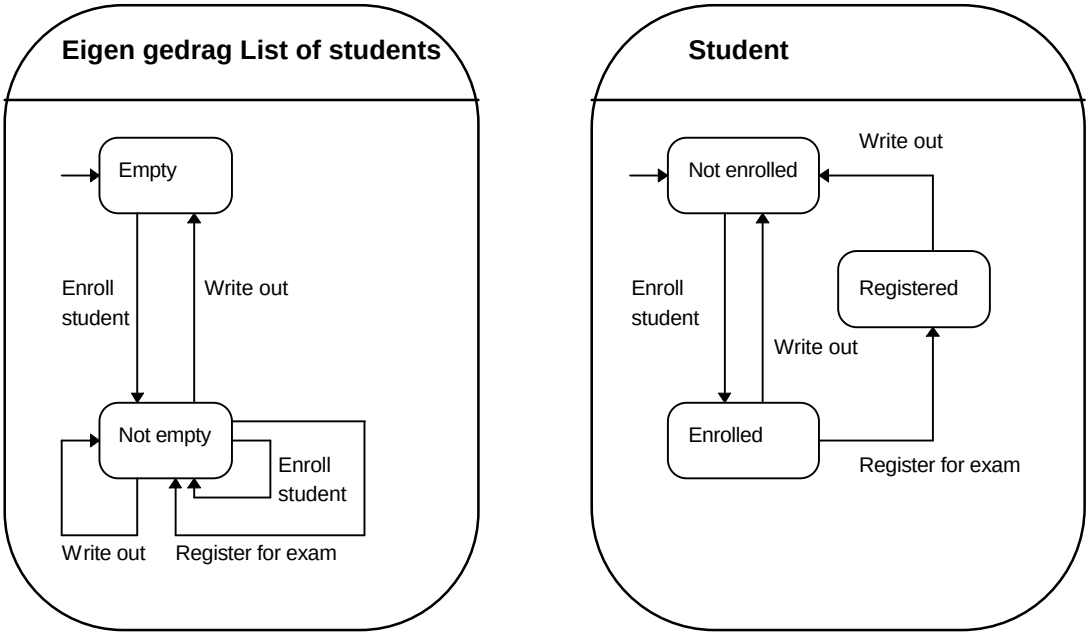
2.4.4 Voorbeeld

Als voorbeeld wordt hier een lijst van studenten gebruikt. Het container-object is hier de List of students die een variabel aantal students bevat. In dit voorbeeld wordt er vanuit gegaan dat het container-object de interface biedt voor de gehele aggregatie. Aan elk van de operaties in List of students wordt dus een qualifier meegegeven die aangeeft naar welk student-object de operatie gedelegeerd moet worden.

Object model:

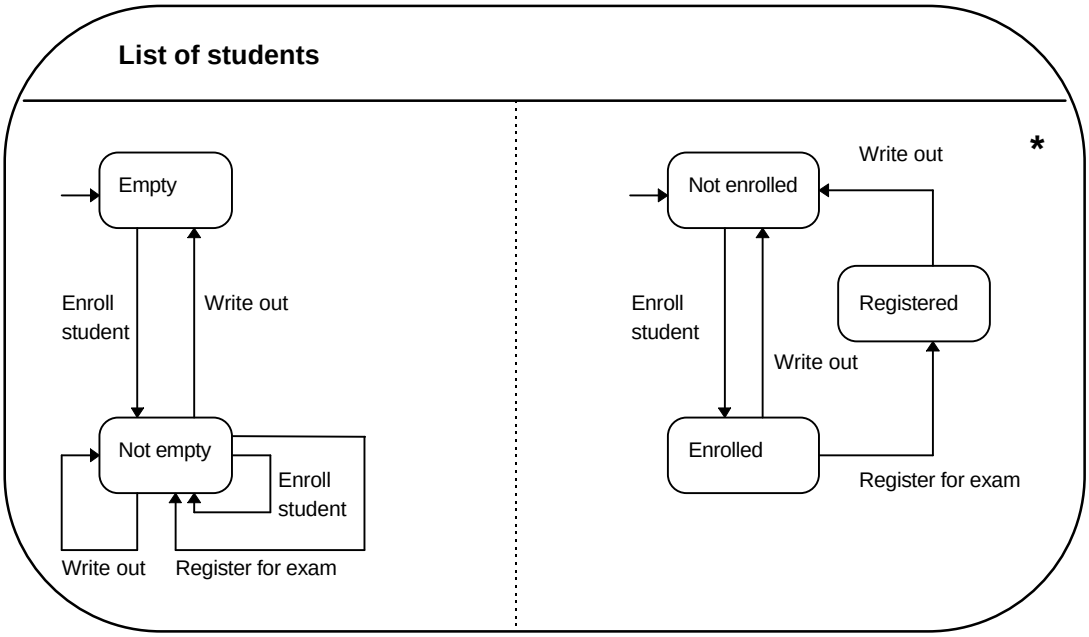


Dynamisch model:



In dit voorbeeld biedt de whole de interface voor de gehele aggregatie. Bovendien zorgt de whole voor het creëren en verwijderen van parts. Het creëren en verwijderen van student-objecten vindt in dit voorbeeld gelijktijdig met de operaties Enroll student en Write out plaats. Het is ook mogelijk dat de whole aparte operaties heeft ten behoeve van het creëren en verwijderen van parts.

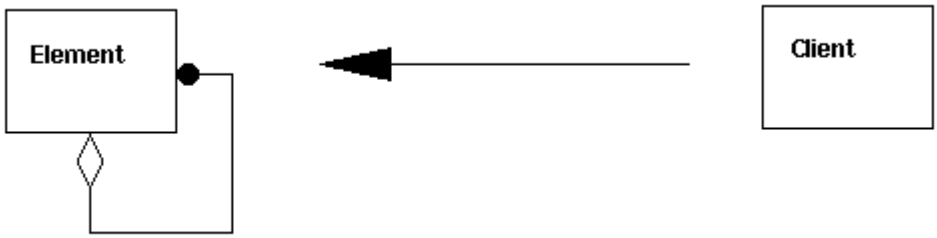
Het toestandsdiagram van de whole kan nu net als bij het type Collection worden geconstrueerd als logische AND van het toestandsdiagram van de whole en dat van het part.



Ook hier is de inbedding van het toestandsdiagram van het part in dat van de whole eenvoudig af te leiden.

2.5 Recursion (Direct Recursive Aggregate)

2.5.1 Structuur



2.5.2 Beschrijving

In dit type aggregatie zijn de whole en het part verschillende objecten van dezelfde class. Niet alleen het aantal parts, maar ook het aantal niveaus is variabel. Afhankelijk van de multipliciteit van de aggregatie ontstaat een lijst-, boom- of gerichte graaf-structuur. De elementen van deze structuur kunnen zowel whole als part zijn in een aggregatie.

In principe kunnen ook hier de parts niet zonder de whole bestaan. Zowel whole als part zijn echter objecten van dezelfde class en spelen in veel gevallen beide rollen.

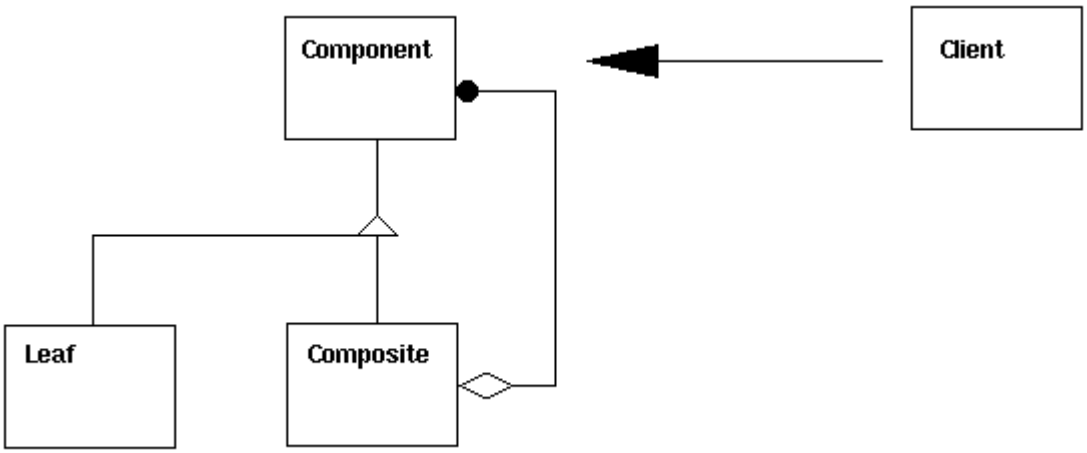
Doordat whole en part objecten van dezelfde class zijn kan ook niet worden gerealiseerd dat de whole de interface biedt voor de totale aggregatie.

2.5.3 Dynamisch model

Omdat whole en part beide een interface bieden zijn de afhankelijkheden in het dynamisch model tussen whole en part vergelijkbaar met het type “Collection met eigen interface van de parts” uit paragraaf 2.3.3. Hier geldt dus ook dat de toestandsovergangen van de whole en de parts geheel onafhankelijk van elkaar kunnen plaatsvinden.

2.6 Composition (Indirect Recursive Aggregate)

2.6.1 Structuur



2.6.2 Beschrijving

Dit type aggregatie geeft een hiërarchie van objecten weer waarbij er een uniforme interface is voor elk object in de hiërarchie, ongeacht of het een samengesteld object (Composite) of een primitief object (Leaf) is. De recursie is in dit type aggregatie indirect (via een inheritance-relatie).

In tegenstelling tot alle andere typen aggregaties lijkt hier het part de interface te bieden met de client. Door de inheritance relatie zal er echter een operatie in Leaf of in Composite worden aangeroepen (dynamic binding).

De whole (Composite) bevat operaties ten behoeve van samengestelde objecten (Composite) en van primitieve objecten (Leaf). De operaties ten behoeve van samengestelde objecten kunnen worden opgevat als het eigen gedrag van de whole. De operaties ten behoeve van de primitieve objecten worden gedelegeerd. Bij dit type aggregatie houdt dat in dat de operaties worden gedelegeerd naar alle parts.

Dit type is afgeleid van het Composite pattern in [GaHeJo 95].

2.6.3 Dynamisch model

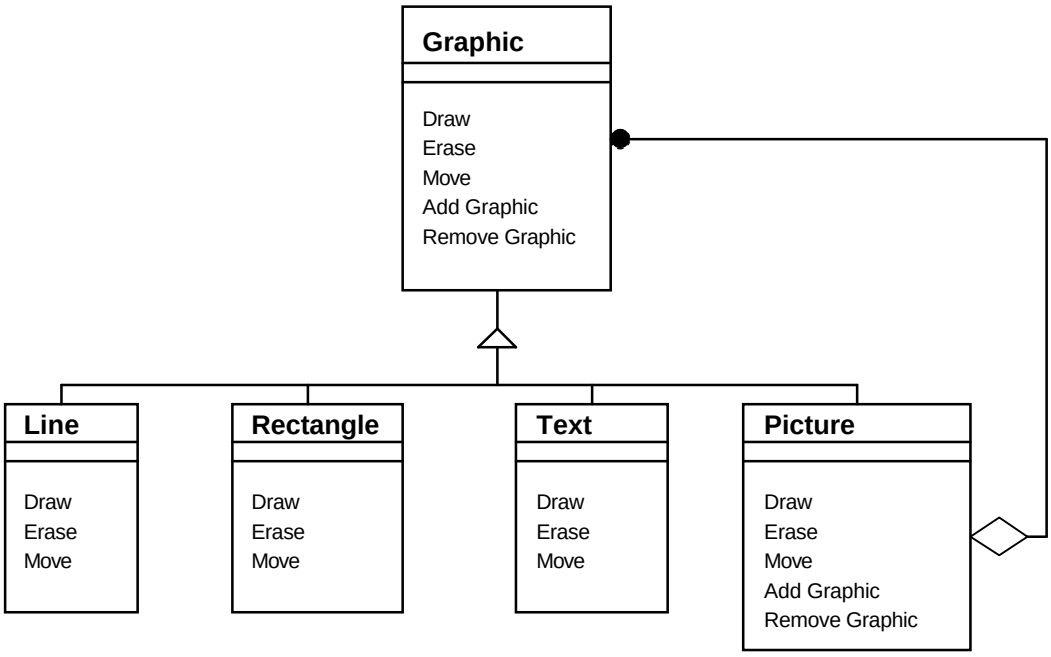
Ook hier is het van belang of de parts in de aggregatie een eigen interface bieden. Bij dit type aggregatie is dat het geval, waardoor de toestandsdiagrammen van whole en part in principe onafhankelijk van elkaar zijn. Er kunnen toestandsovergangen in een component-object (part) plaatsvinden, zonder dat er in het bijbehorende composite-object (whole) een toestandsovergang plaatsvindt.

2.6.4 Voorbeeld

In onderstaand voorbeeld wordt een grafische afbeelding gemodelleerd, die is opgebouwd uit de primitieve classes Line, Rectangle en Text en uit een container-class Picture. Op deze manier wordt een grafisch object opgebouwd uit andere grafische objecten die uiteindelijk zijn opgebouwd uit primitieve objecten.

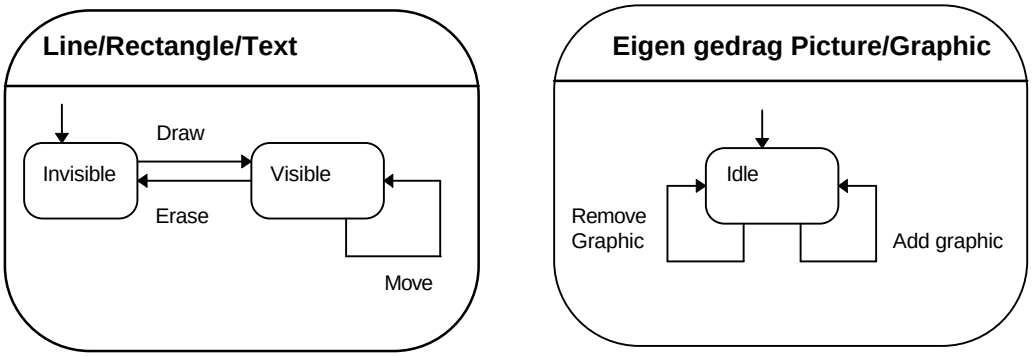
Deze constructie zorgt ervoor dat dezelfde interface wordt geboden, ongeacht of het samengestelde of primitieve objecten betreft.

Object model:



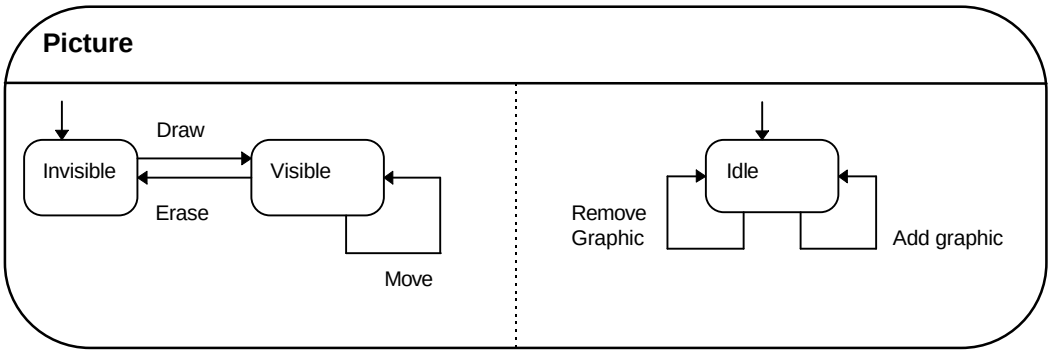
Dynamisch model:

Het toestandsdiagram van Line, Rectangle en Text bevat de operaties van de primitieve objecten. Het eigen gedrag van Picture en Graphic bevat de operaties van de samengestelde objecten.



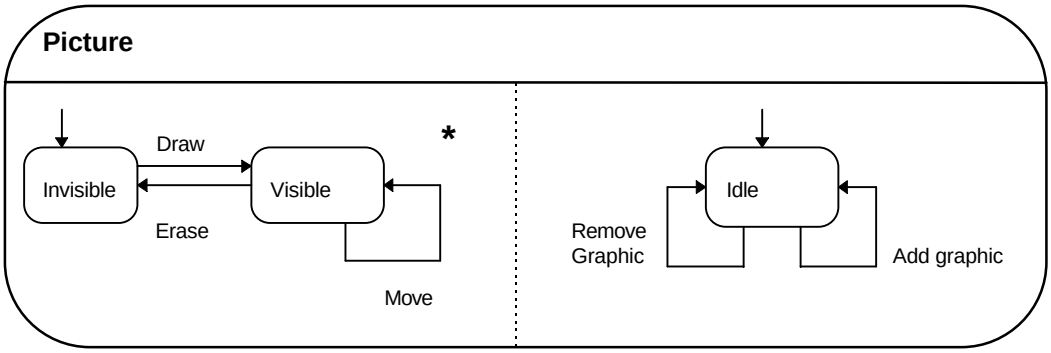
Wanneer we nu proberen een toestandsdiagram voor de totale aggregatie te maken, dan kan dit toestandsdiagram op verschillende manieren worden geconstrueerd.

De eerste manier is, om de bovengenoemde toestandsdiagrammen samen te voegen. De operaties en toestanden in dit toestandsdiagram hebben dan betrekking op de aggregatie als geheel.



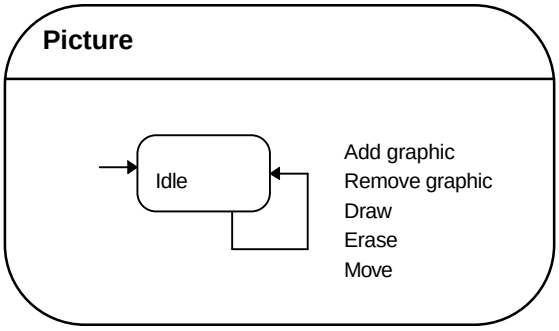
Het probleem hierbij is dat de objecten waar het samengestelde object uit is opgebouwd, ook afzonderlijk een interface bieden naar de clients. Er kan in één van die objecten een toestandsovergang plaatsvinden, zonder dat die in het samengestelde object plaatsvindt. Wanneer bijvoorbeeld in alle objecten waaruit picture is opgebouwd de operatie Erase zou worden aangeroepen, dan zou Picture in toestand Visible zijn, terwijl alle objecten waaruit hij is samengesteld in toestand Invisible zijn.

De tweede mogelijkheid is om een toestandsdiagram van de gehele aggregatie op dezelfde manier te construeren als bij het type Collection.



De betekenis van de toestanden Invisible en Visible is nu heel anders. Nu geeft dit per object (part) de toestand aan en niet meer voor het Picture als geheel. Het probleem blijft echter, dat in één van de parts een toestandsovergang plaatsvindt, zonder dat er in Picture een operatie is aangeroepen.

De laatste mogelijkheid is natuurlijk om in een samengesteld object de operaties Draw, Erase en Move altijd toe te staan. Voor de eenvoud van het diagram is slecht één pijl getekend voor vijf toestandsovergangen.



In dit voorbeeld blijkt het niet goed mogelijk te zijn een toestandsdiagram voor de totale aggregatie te bepalen. Omdat in een recursieve structuur de whole en de parts dezelfde classes zijn, is het onmogelijk om alleen de whole de interface te laten bieden voor de gehele aggregatie.

2.7 Conclusies

In dit hoofdstuk is een aantal typen aggregaties besproken en aan de hand van voorbeelden bekeken welke relatie er ligt tussen het dynamisch model van de whole en dat van de parts.

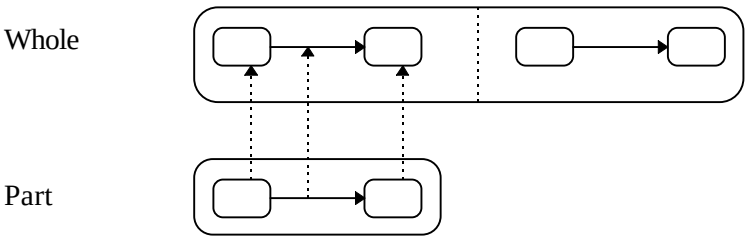
De belangrijkste conclusie is, dat er alleen sprake is van een relatie tussen het dynamisch model van de whole en dat van de parts als er gebruik gemaakt wordt van delegation. Bij gebruik van delegation heeft een toestandsovergang in de whole ook een toestandsovergang in één van de parts tot gevolg.

Daarnaast is het van belang of de whole de interface biedt voor de hele aggregatie. Als dat het geval is betekent dat, dat alle operaties van de parts wordt aangeroepen door middel van delegation. De parts bieden immers zelf geen interface. In dit geval kan er geen toestandsovergang in een part plaatsvinden zonder dat er een toestandsovergang in de whole plaatsvind. Het gedrag van het part is dan ingebed in dat van de whole.

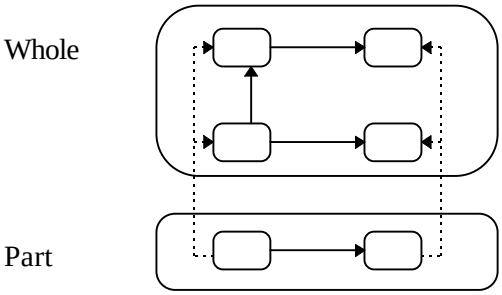
Alleen in een situatie dat de whole de interface biedt voor de gehele aggregatie en de operaties van de parts aanroept met behulp van delegation, kan worden vastgesteld of het toestandsdiagram van de whole correct is in relatie tot de toestandsdiagrammen van de parts.

2.7.1 Correctheid

Allereerst is gebleken dat een correct toestandsdiagram voor de whole kan worden geconstrueerd als logische AND van de toestandsdiagrammen van de parts. Het toestandsdiagram van elk part komt dan ongewijzigd terug in dat van de whole. Eventueel kunnen er beperkingen worden toegevoegd aan de toestandsovergangen in de whole. Dit kan als volgt schematisch worden weergegeven:



Daarnaast kan het toestandsdiagram van de whole worden geconstrueerd als cartesisch produkt van de toestandsdiagrammen van de parts. Eventueel kunnen toestandsovergangen worden verwijderd om bepaalde beperkingen te implementeren. Ook een op deze manier geconstrueerd toestandsdiagram is correct in relatie tot de toestandsdiagrammen van de parts. Dit kan als volgt schematisch worden weergegeven:



Wanneer het toestandsdiagram van de whole niet is opgebouwd op één van de hierboven genoemde manieren, is het niet meer eenvoudig na te gaan of het toestandsdiagram van de whole correct is in relatie tot de toestandsdiagrammen van de parts. Als bepaalde operaties in de whole worden gedelegeerd naar één van de parts kan het zo zijn, dat een toegestane reeks van operaties in de whole leidt tot een niet toegestane reeks van operaties in één van de parts. In zo'n geval is het toestandsdiagram van de whole niet correct in relatie tot dat van het part.

3 Relatie met inheritance

3.1 Algemeen

Een belangrijke toepassing van aggregaties is het realiseren van hergebruik. In die zin is een aggregatie een alternatief voor een generalisatie. In [GaHeJo 95] p. 19 wordt een aggregatie ook als alternatief voor inheritance beschreven.

Hergebruik met behulp van inheritance wordt ook wel white-box reuse genoemd, terwijl hergebruik met behulp van een aggregatie black-box reuse wordt genoemd. Bij inheritance erft het subtype alle attributen en operaties van het supertype, zodat alle details van het supertype bekend zijn bij het subtype. Bij een aggregatie kan de whole communiceren met een part via de interface die het part biedt. De whole kent de details van het part niet.

Een tweede opvallend verschil is het verschil in abstractieniveau tussen de objecten. Bij inheritance gebruikt het subtype functionaliteit van het supertype. De herbruikbare functionaliteit bevindt zich dus op een hoger abstractieniveau. Bij een aggregatie gebruikt de whole functionaliteit van het part. De herbruikbare functionaliteit bevindt zich dus op een lager abstractieniveau.

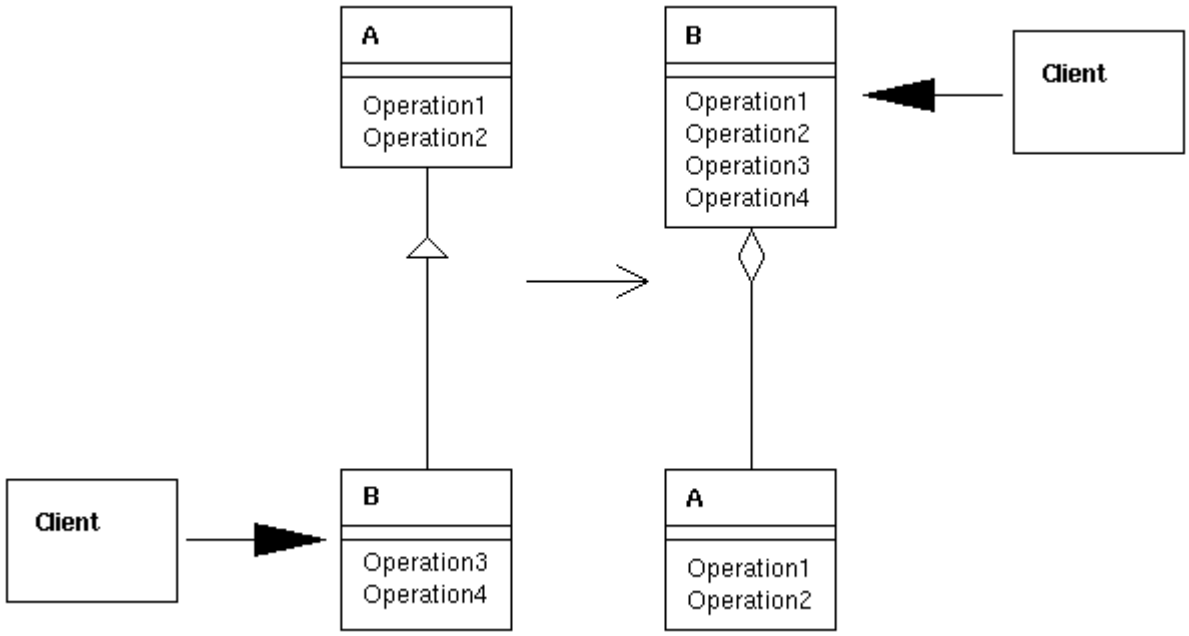
Een ander verschil is, dat bij inheritance de implementatie van een operatie wordt gevonden met behulp van dynamic binding. Tijdens executie wordt pas bepaald welke operatie feitelijk wordt uitgevoerd. Bij een aggregatie wordt gebruik gemaakt van delegation. De uit te voeren operatie wordt dan expliciet aangegeven en is bij executie op voorhand bekend.

Wanneer een object functionaliteit van meerdere andere objecten wil gebruiken, dan doet zich bij inheritance het probleem van multiple inheritance voor. Doordat een subtype functionaliteit erft van meerdere supertypes kunnen er conflicten ontstaan. Bij aggregaties bestaat dat probleem niet. Zie [RuBlPr 91] p. 65-69.

3.1.1 Van inheritance naar aggregatie

De typen Assembly en Collection (zonder eigen interface van de parts) uit het vorige hoofdstuk zijn typen aggregaties waarbij de parts herbruikbare functionaliteit bevatten en waarbij de whole (net als bij een subtype) een totale interface biedt.

Onderstaand figuur geeft schematisch weer, hoe een inheritance relatie kan worden omgezet naar een aggregatie. In dit voorbeeld worden de operaties 1 en 2 in class A hergebruikt in class B. Bij inheritance gebeurt dit, doordat B de operaties 1 en 2 erft van A. De operaties 3 en 4 zijn uitsluitend in het subtype geïmplementeerd. Bij de aggregatie gebeurt dit, doordat B de operaties 1 en 2 delegeert naar A. De operaties 3 en 4 zijn onderdeel van het eigen gedrag van B. In beide gevallen biedt B de interface naar de clients.



3.1.2 Dynamisch model

In [EbeEng 95] wordt de relatie beschreven tussen het dynamisch model van super- en subtype in een inheritance relatie. In dat onderzoek wordt correctheid op twee verschillende manieren gedefinieerd, afhankelijk van de exacte betekenis van het toestandsdiagram.

De eerste manier is om het toestandsdiagram op te vatten als een beschrijving van alle mogelijke reeksen van achtereenvolgens aan te roepen operaties (Observable behaviour). Correctheid betekent in dit geval, dat elke toegestane reeks van achtereenvolgens aan te roepen operaties in een subtype moet leiden tot een toegestane reeks van operaties in het supertype.

De tweede manier is om het toestandsdiagram op te vatten als een beschrijving van alle reeksen van operaties die kunnen worden uitgevoerd (Invocable behaviour). Correctheid betekent in dit geval, dat elke uitvoerbare reeks operaties in het supertype ook uitvoerbaar moet zijn in elk subtype.

Uit dat onderzoek blijkt, dat het toestandsdiagram van het subtype op drie verschillende manieren gerelateerd kan zijn aan het toestandsdiagram van het supertype.

- Parallel extension. Het toestandsdiagram van het subtype is uitgebreid met een extra toestandsdiagram dat onafhankelijk is van het toestandsdiagram van het supertype.
- State refinement. Het toestandsdiagram van het subtype is geconstrueerd uit dat van het supertype door een toestand te verfijnen (extra toestanden en overgangen binnen één toestand van het supertype).
- Adding transitions. Het toestandsdiagram van het subtype is geconstrueerd uit dat van het supertype door extra toestandsovergangen toe te voegen.

In het geval van Parallel extension en State refinement blijkt uit dit onderzoek, dat het toestandsdiagram van het supertype correct is in relatie tot dat van het subtype, wanneer de toestandsdiagrammen worden geïnterpreteerd als observable behaviour. Een toegestane reeks operaties in het subtype leidt tot een toegestane reeks operaties in het supertype. Wanneer de toestandsdiagrammen worden geïnterpreteerd als invocable behaviour, blijken Parallel

extension en State refinement niet correct te zijn. Niet elke uitvoerbare reeks operaties in het supertype blijkt uitvoerbaar in het subtype.

In het geval van Adding transitions blijkt precies het omgekeerde het geval te zijn. Deze constructie blijkt correct te zijn wanneer het toestandsdiagram wordt opgevat als invocable behaviour, terwijl dat niet het geval is bij observable behaviour.

Door middel van de omzetting zoals beschreven in de vorige paragraaf, kunnen deze drie gevallen ook bekeken worden voor aggregaties. In de volgende paragrafen worden deze drie situaties bekeken.

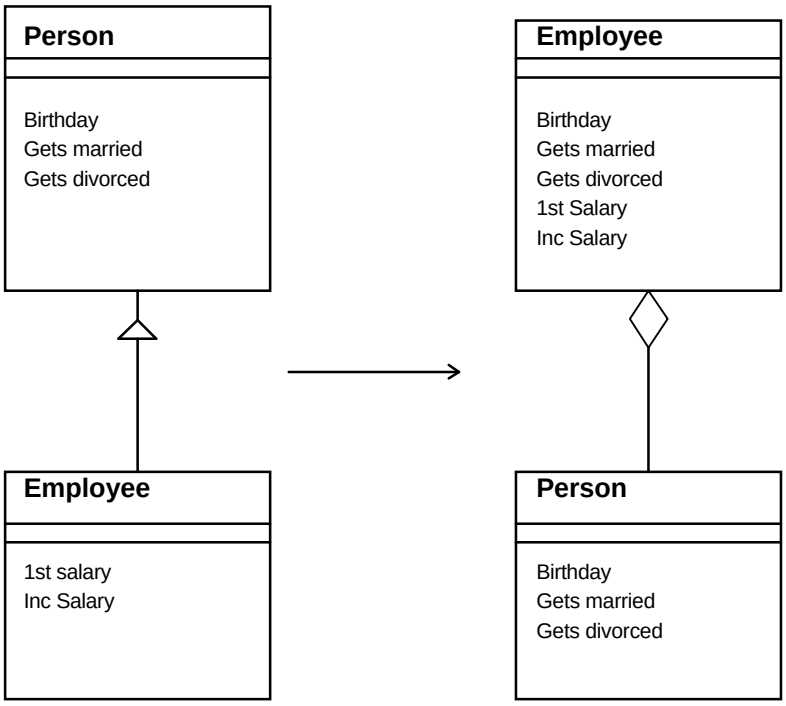
3.2 Parallel extension

In deze paragraaf wordt het voorbeeld van Parallel Extension uit [EbeEng 95] gebruikt. De inheritance-relatie wordt op de hiervoor beschreven manier omgezet naar een aggregatie. Tenslotte wordt bekeken wat de relatie is tussen het toestandsdiagram van de whole en dat van het part in deze aggregatie.

3.2.1 Voorbeeld

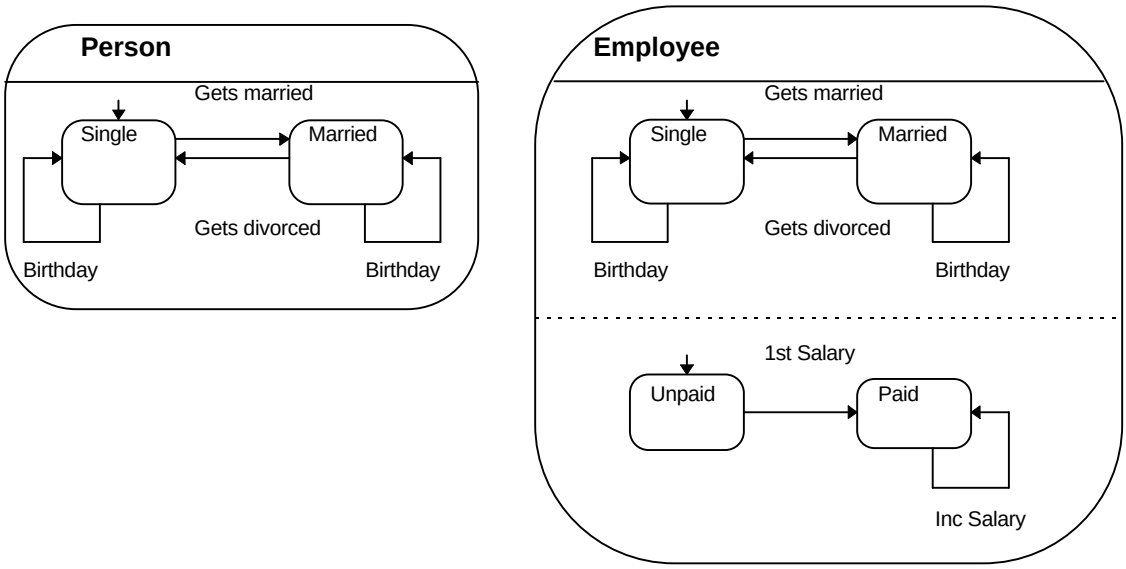
In dit voorbeeld wordt het gedrag van Person hergebruikt in Employee. In het geval van inheritance worden de operaties Birthday, Gets married en Gets divorced overgeërfd van Person. Wanneer deze inheritance-relatie wordt omgezet in een aggregatie, worden de operaties Birthday, Gets married en Gets divorced gedelegeerd van Employee naar Person.

Object model:



Dynamisch model:

Het gedrag van Person en Employee is in de aggregatie hetzelfde als in de inheritance-relatie. Daarom kan voor beide situaties het dynamisch model zoals in het voorbeeld in [EbeEng 95] worden gebruikt.



3.2.2 Dynamisch model

We kunnen in dit voorbeeld onderscheid maken tussen het eigen gedrag van de whole (1st Salary en Inc Salary in Employee) en het gedrag van het part (Birthday, Gets married en Gets divorced in Person). Zoals opgemerkt bij Assembly kan het toestandsdiagram van de whole nu worden geconstrueerd door de logische AND te nemen van deze toestandsdiagrammen. Als we dit doen krijgen we precies het getoonde toestandsdiagram van Employee.

Dit is dus een constructieve definitie van het toestandsdiagram. Het is echter min of meer toevallig dat het toestandsdiagram in het voorbeeld hier precies mee correspondeert. Het is mogelijk dat er in de whole extra beperkingen zijn toegevoegd. Ook zouden de twee toestandsdiagrammen kunnen zijn samengevoegd tot één diagram.

Zoals in de conclusies van het vorige hoofdstuk al naar voren is gekomen, blijkt in dit geval het toestandsdiagram van de whole altijd correct te zijn in relatie tot de toestandsdiagrammen van de parts. De toestanden en toestandsovergangen in de whole lopen gewoon parallel aan die in de parts.

3.3 State refinement

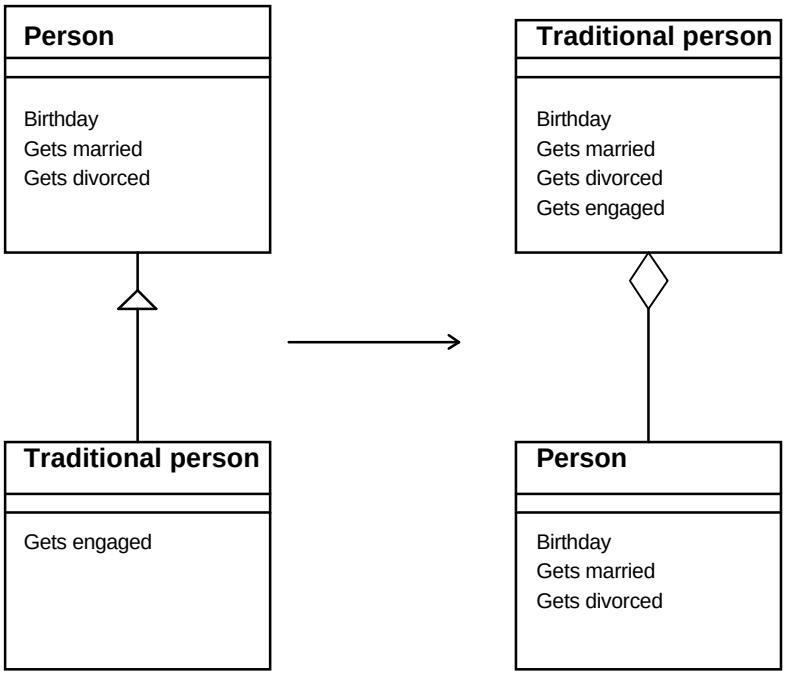
In deze paragraaf wordt het voorbeeld van State Refinement uit [EbeEng 95] gebruikt. De inheritance-relatie wordt op de hiervoor beschreven manier omgezet naar een aggregatie. Tenslotte wordt bekeken wat de relatie is tussen het toestandsdiagram van de whole en dat van het part in deze aggregatie.

3.3.1 Voorbeeld

In dit voorbeeld is Traditional person een Person waaraan een beperking is toegevoegd door de toestand Not married te verfijnen tot twee verschillende toestanden: Not engaged en Engaged. De beperking is, dat een toestandsovergang naar Married nu alleen kan plaatsvinden vanuit de toestand Engaged.

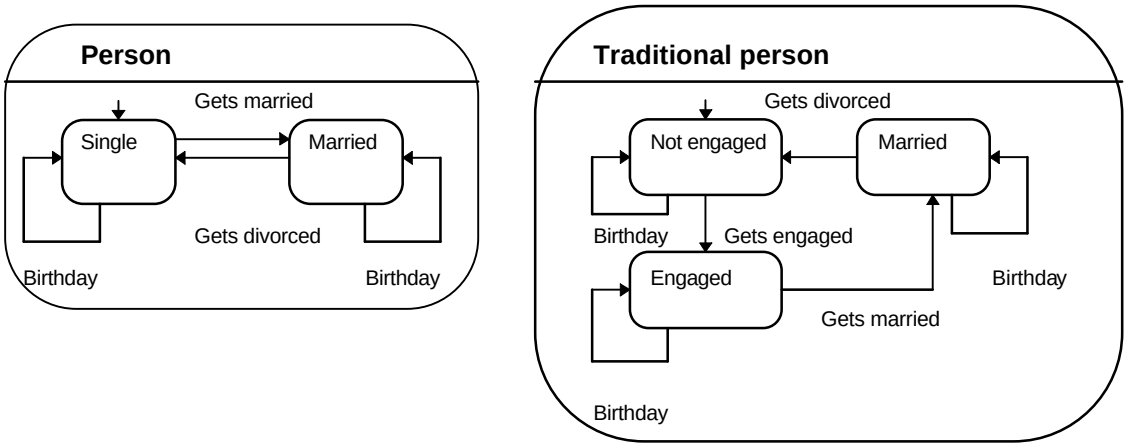
Net als in het vorige voorbeeld (Parallel extension) wordt ook hier het gedrag van Person hergebruikt. In de inheritance-relatie doordat Traditional person de operaties Birthday, Gets married en Gets divorced overerfd van Person, en in de aggregatie doordat Traditional person deze operaties delegeert naar Person.

Object model:



Dynamisch model:

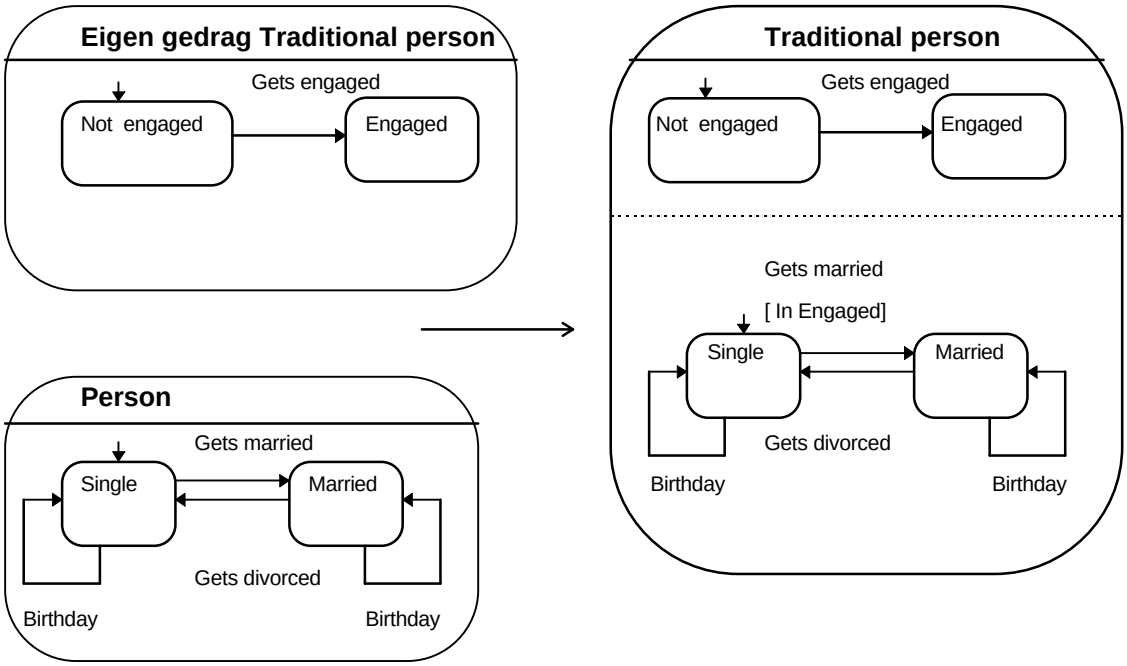
Het gedrag van Person en Traditional person is in de aggregatie hetzelfde als in de inheritance-relatie. Daarom kan voor beide situaties het dynamisch model zoals in het voorbeeld in [EbeEng 95] worden gebruikt.



3.3.2 Dynamisch model

We kunnen ook hier onderscheid maken tussen het eigen gedrag van de whole (Gets engaged in Traditional person) en het gedrag van het part (Birthday, Gets married en Gets divorced in Person).

Hieronder is links het toestandsdiagram van het eigen gedrag van Traditional person weergegeven en het toestandsdiagram van Person (hetzelfde als hierboven). Wanneer het toestandsdiagram van Traditional person wordt geconstrueerd als logische AND van deze twee toestandsdiagrammen, ontstaat het toestandsdiagram rechts.

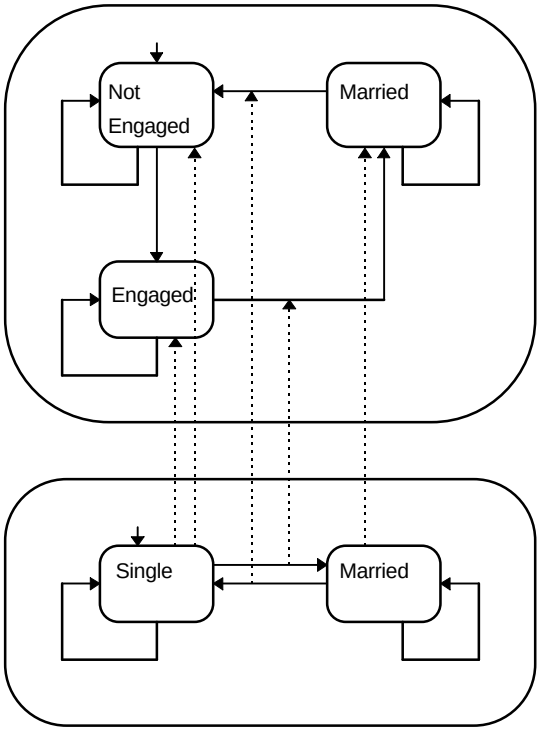


Het is in dit voorbeeld noodzakelijk om de logische AND van beide toestandsdiagrammen uit te breiden met een extra beperking. De toestandsovergang van Single naar Married (Gets married) mag alleen plaatsvinden wanneer het andere (parallelle) toestandsdiagram in toestand Engaged is.

In het oorspronkelijke voorbeeld is het toestandsdiagram van de whole echter niet op deze manier geconstrueerd. Het is nu niet direct duidelijk of dit toestandsdiagram ook correct is. Hiervoor moet worden bekeken hoe het toestandsdiagram van het part is ingebed in dat van de whole en of daar geen conflicten uit kunnen voortkomen.

In onderstaande schets wordt deze relatie weergegeven.

Toestandsdiagram van de whole



Toestandsdiagram van het part

Nu moet worden bekeken of er geen conflicten kunnen optreden. Wanneer het part in toestand Single is zou de whole altijd in één van de toestanden Not engaged of Engaged moeten zijn. Wanneer het part in toestand married is, zou de whole eveneens in toestand married moeten zijn. Dit is in dit voorbeeld gewaarborgd doordat een toestandsovergang van Not Engaged of Engaged naar Married alleen kan plaatsvinden middels een gedelegeerde operatie, waardoor het part ook in de correcte toestand blijft. Dit geldt ook voor de toestandsovergang van Married naar Not engaged of Engaged.

3.4 Adding transitions

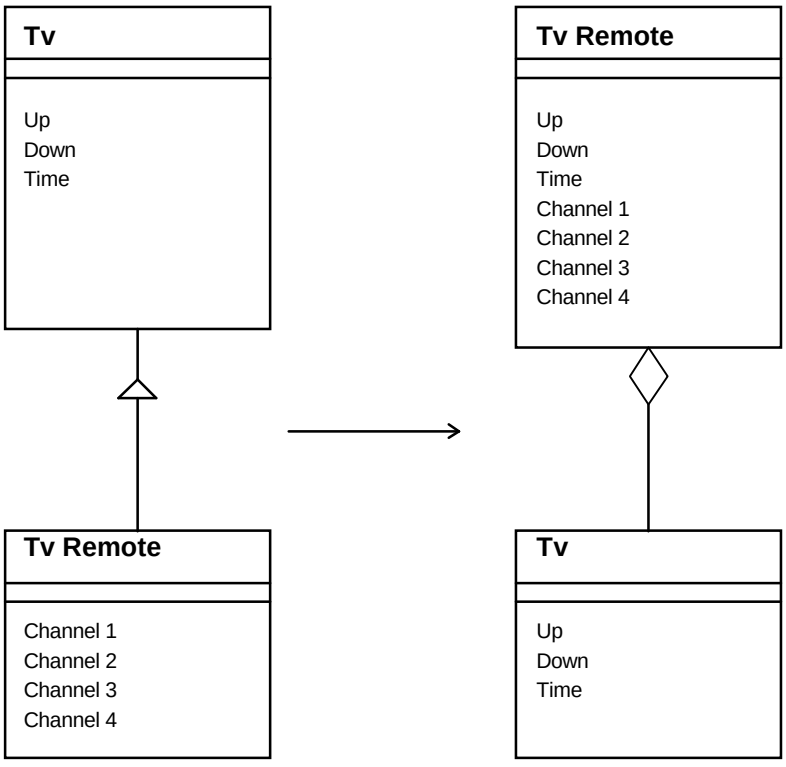
In deze paragraaf wordt het voorbeeld van Adding transitions uit [EbeEng 95] gebruikt. De inheritance-relatie wordt op de hiervoor beschreven manier omgezet naar een aggregatie. Tenslotte wordt bekeken wat de relatie is tussen het toestandsdiagram van de whole en dat van het part in deze aggregatie.

3.4.1 Voorbeeld

Dit voorbeeld beschrijft een TV waarbij tussen de channels kan worden gewisseld met behulp van een Up en Down operatie. De Tv Remote is een uitbreiding hierop, zodat met de toegevoegde operaties Channel 1 t/m Channel 4 ook rechtstreeks de channels gekozen kunnen worden.

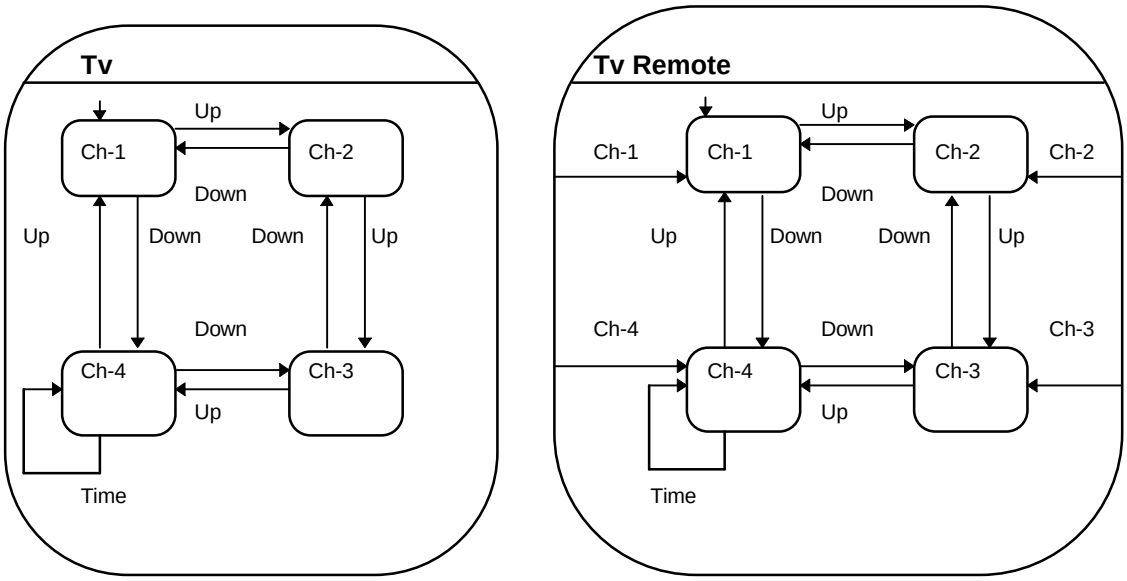
Het gedrag van Tv wordt hergebruikt in Tv Remote. In het geval van inheritance worden de operaties Up, Down en Time overgeërfd van Tv. Wanneer deze inheritance-relatie wordt omgezet in een aggregatie, worden de operaties Up, Down en Time gedelegeerd van Tv Remote naar Tv.

Object model:



Dynamisch model:

Het gedrag van Tv en Tv Remote is in de aggregatie hetzelfde als in de inheritance-relatie. Daarom kan voor beide situaties het dynamisch model zoals in het voorbeeld in [EbeEng 95] worden gebruikt.



3.4.2 Dynamisch model

Ook hier is het mogelijk om onderscheid te maken tussen het eigen gedrag van de whole (Channel-1 t/m Channel-4) en het gedrag van het part (Up, Down, Time). Het verschil met de vorige twee voorbeelden is echter, dat de toestandsdiagrammen beide dezelfde toestanden bevatten, met andere toestandsovergangen. Het toestandsdiagram van de whole kan dus geconstrueerd worden, door deze beide toestandsdiagrammen samen te voegen. Dit leidt tot het toestandsdiagram van Tv Remote zoals in het voorbeeld.

In dit voorbeeld zijn er in de whole meer toestandsovergangen dan in het part. Wanneer op een bepaald moment de whole en het part zich in toestand Ch-1 bevinden en de operatie Channel 4 wordt aangeroepen, dan zal de whole zich in toestand Ch-4 bevinden terwijl het part zich nog in toestand Ch-1 bevindt. Wanneer vervolgens de operatie Time wordt aangeroepen, dan is dat toegestaan volgens het toestandsdiagram van de whole, want deze bevindt zich in toestand Ch-4. In het part ontstaat echter een conflict, omdat deze zich nog in toestand Ch-1 bevindt waar de operatie Time niet toegestaan is.

In tegenstelling tot de twee vorige voorbeelden (Parallel extension en State refinement) blijkt in dit geval het toestandsdiagram van de whole niet correct te zijn in relatie tot dat van het part. Dit is opvallend omdat dit vergelijkbaar is met de resultaten uit [EbeEng 95]. Wanneer de toestandsdiagrammen worden opgevat als een beschrijving van het Observable behaviour, blijkt Adding transitions niet correct te zijn en Parallel extension en State refinement wel.

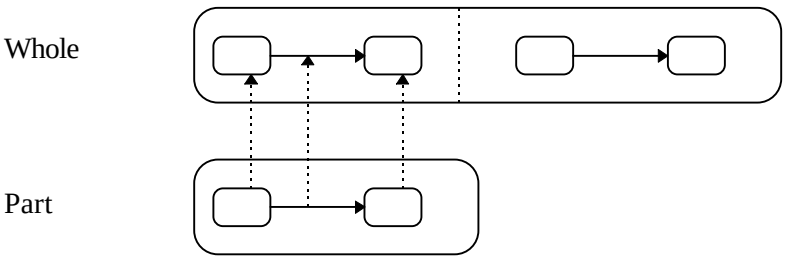
In het geval van Observable behaviour is het toestandsdiagram van het subtype correct in relatie tot het toestandsdiagram van supertype wanneer een toegestane reeks operaties in het subtype leidt tot een toegestane reeks operaties in het supertype. Omdat in de omzetting van een inheritance-relatie naar een aggregatie de rol van het subtype door de whole wordt vervuld en de rol van het supertype door het part, is de definitie van correctheid vergelijkbaar met die voor een aggregatie. Bij een aggregatie (van het type Assembly) is het toestandsdiagram van de whole correct in relatie tot dat van het part als een toegestane reeks van operaties in de whole leidt tot een toegestane reeks van operaties in het part.

3.5 Conclusies

Doordat het mogelijk is om een inheritance relatie om te zetten naar een aggregatie kunnen de resultaten van het onderzoek in [EbeEng 95] ook hier worden gebruikt. Hieronder worden de relaties tussen het dynamisch model van de whole en de parts zoals die in dit hoofdstuk zijn beschreven vergeleken met de bevindingen in het vorige hoofdstuk.

Parallel extension:

In deze situatie is het toestandsdiagram van de whole uitgebreid met het toestandsdiagram van het part. Dit kan als volgt worden weergegeven:

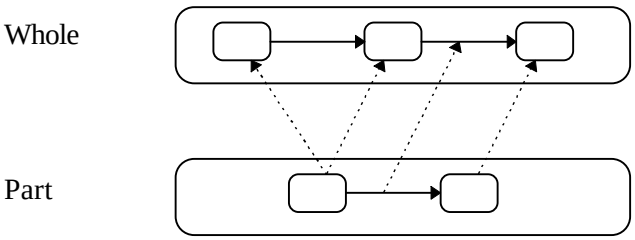


Deze situatie komt overeen met de constructie van het toestandsdiagram van de whole als logische AND van de toestandsdiagrammen van de parts en het eigen gedrag van de whole. Zoals ook in de conclusies bij hoofdstuk 2 is vermeld is het toestandsdiagram van de whole in zo'n geval altijd correct in relatie tot die van de parts.

Het toestandsdiagram zou in zo'n geval ook kunnen worden geconstrueerd als cartesisch produkt.

State refinement:

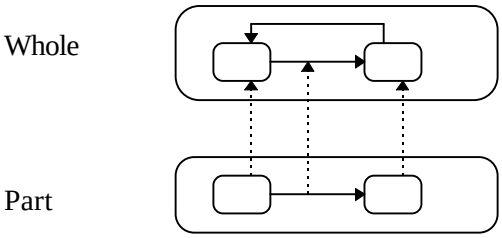
In deze situatie is het toestandsdiagram van de whole ontstaan door een toestand in het part te verfijnen. Dit gebeurt door deze toestand te splitsen in één of meerdere toestanden en daartussen extra toestandsovergangen toe te voegen. Deze toestandsovergang heeft een operatie die niet naar dit part wordt gedelegeerd. Dit kan als volgt worden weergegeven.



Ook in dit geval blijft het toestandsdiagram van de whole correct in relatie tot dat van de parts, doordat er geen toestandsovergangen worden toegevoegd anders dan binnen de opgesplitste toestand. Het is niet mogelijk dat in de whole een operatie is toegestaan, terwijl de gedelegeerde operatie in het part niet is toegestaan.

Adding transitions:

In deze situatie is het toestandsdiagram ontstaan door toestandsovergangen toe te voegen aan het toestandsdiagram van het part. Dit kan als volgt worden weergegeven:



Hierdoor ontstaat de situatie dat een toestandsovergang in de whole die correspondeert met een toestandsovergang in het part (doordat de operatie gedelegeerd wordt) ook kan ontstaan via een omweg, doordat extra toestandsovergangen in de whole zijn toegevoegd waarbij de operaties niet worden gedelegeerd naar het part.

Het toestandsdiagram van de whole is in deze situatie dus over het algemeen niet correct in relatie tot de toestandsdiagrammen van de parts.

Dit is in overeenstemming met de conclusies in [EbeEng 95] met betrekking tot Observable behaviour in het geval van een inheritance-relatie. Ook daar blijkt Adding transitions niet correct te zijn en Parallel extension en State refinement wel. Wanneer de toestandsdiagrammen worden opgevat als een beschrijving van het Observable behaviour, dan betekent correctheid in een inheritance-relatie, dat een toegestane reeks operaties in het subtype moet leiden tot een toegestane reeks operaties in het supertype. In een aggregatie (van het type Assembly) betekent correctheid dat een toegestane reeks operaties in de whole moet leiden tot een toegestane reeks operaties in het part.

In de omzetting van een inheritance-relatie naar een aggregatie vervult de whole de rol van het subtype en het part de rol van het supertype. De conclusies in [EbeEng 95] met betrekking tot Observable behaviour in een inheritance-relatie zijn dus vergelijkbaar met de conclusies met betrekking tot aggregaties van het type Assembly.

4 Formele beschrijving

4.1 Inleiding

In hoofdstuk 2 en 3 zijn verschillende typen aggregaties met elkaar vergeleken. Met name is de aandacht gericht geweest op de afhankelijkheden die er zijn tussen het dynamisch model van de whole en dat van de parts.

Eén van de belangrijkste aspecten daarbij was of de whole de interface biedt voor de gehele aggregatie. Als dat het geval is, is de whole de enige die operaties van de parts aanroept. De meest eenvoudige manier waarop dat kan plaatsvinden is, doordat de whole bepaalde operaties delegeert naar één van zijn parts. Op deze manier ontstaat er een relatie tussen het toestandsdiagram van de whole en dat van de parts.

Het doel van de formele beschrijving is om aan te geven wanneer het toestandsdiagram van de whole correct is in relatie tot de toestandsdiagrammen van de parts. Correctheid betekent hier dat elke toegestane reeks operaties in de whole moet leiden tot een toegestane reeks operaties in elk van de parts. Dit kan worden gedaan door te bepalen welke relatie er moet liggen tussen het toestandsdiagram van de whole en dat van de parts zodat het toestandsdiagram van de whole correct is.

In dit hoofdstuk wordt eerst aangegeven welke aanpak wordt gevolgd om tot zo'n afbeelding te komen en wat in dit verband onder correctheid wordt verstaan. Vervolgens wordt op een informele manier aangegeven hoe de relatie tussen de toestandsdiagrammen tot stand komt en hoe op basis van deze afbeelding kan worden afgeleid dat het toestandsdiagram van de whole correct is.

Vervolgens wordt een formele definitie opgesteld met behulp van de formele specificatietaal Z [Spiv 92], [Dil 90].

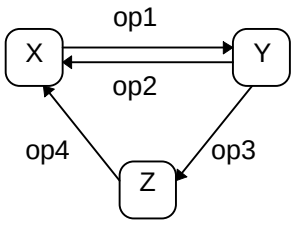
4.2 Aanpak

De aanpak die in dit hoofdstuk wordt weergegeven heeft tot doel te kunnen bepalen of het toestandsdiagram van de whole correct is in relatie tot het toestandsdiagram van de parts.

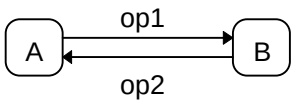
- Wat is de relatie tussen het toestandsdiagram van de whole en dat van de parts.
- Aan welke voorwaarden moet de relatie voldoen zodat het toestandsdiagram van de whole correct is in relatie tot de toestandsdiagrammen van de parts.

Deze aanpak kan met een eenvoudig voorbeeld worden geïllustreerd:

Toestandsdiagram Whole:



Toestandsdiagram Part:



Wanneer de operaties op1 en op2 in de whole worden gedelegeerd naar op1 en op2 in het part, dan zal een toestandsovergang in de whole eveneens leiden tot een toestandsovergang in het part. Hierdoor ontstaat dus een relatie tussen het toestandsdiagram van de whole en dat van het part. De toestandsovergang (X, Y) in de whole correspondeert met toestandsovergang (A, B) in het part.

Onder correctheid wordt hier verstaan dat het toestandsdiagram van de whole niet conflicteert met één van de toestandsdiagrammen van de parts. Er is sprake van een conflict wanneer er een bepaalde reeks van achtereenvolgens aan te roepen operaties is toegestaan volgens het toestandsdiagram van de whole, maar niet in het toestandsdiagram van het part.

Wanneer nu in de whole achtereenvolgens de operaties op1, op2, op1 worden aangeroepen, dan zou dit, op grond van de relatie, dezelfde volgorde van operaties in het part tot gevolg hebben. Er zou dan geen conflict optreden. Echter wanneer achtereenvolgens de operaties op1, op3, op4, op1 worden aangeroepen, dan zou dit in het part de volgorde op1, op1 tot gevolg hebben. Volgens het toestandsdiagram van het part is dat echter niet toegestaan. Hier is dus sprake van een conflict.

4.3 Informele beschrijving

4.3.1 Uitgangspunten

We gaan uit van een aggregatie met de volgende eigenschappen:

- De whole biedt de interface voor alle operaties in de aggregatie.
- De operaties van de parts worden middels delegation aangeroepen door de whole.

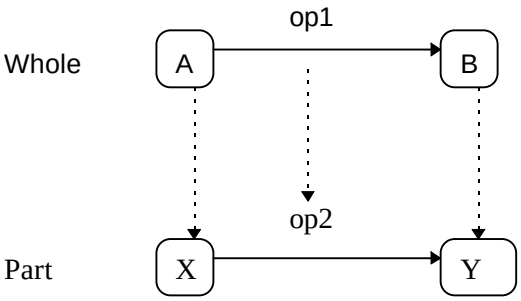
Verder wordt deze beschrijving beperkt tot het type Assembly.

4.3.2 Relatie tussen toestandsdiagrammen

Uitgangspunt in deze beschrijving is de relatie die er ligt vanwege de delegatie van operaties van de whole naar één van de parts. Alleen vanwege het feit dat operaties in de whole worden gedelegeerd naar de parts, is er sprake van een relatie tussen de betrokken toestandsdiagrammen. Op basis hiervan zijn er twee mogelijke afbeeldingen van het gedrag van de whole op dat van een part.

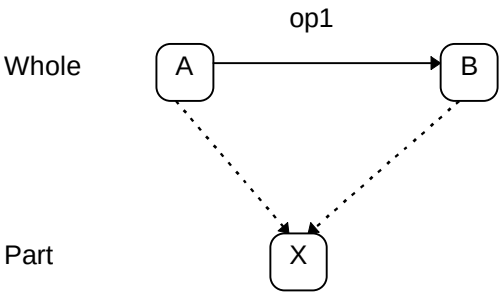
Gedelegeerd operatie

Hier wordt de operatie op1 in whole gedelegeerd naar op2 in het part. Wanneer een operatie wordt gedelegeerd naar het part, dan worden de begin- en eindtoestanden afgebeeld op de bijbehorende begin- en eindtoestanden in het part.



Niet gedelegeerd operatie

Hier wordt operatie op1 in de whole niet gedelegeerd naar een operatie in het part. Deze operatie is dus onderdeel van het eigen gedrag van de whole, of deze operatie wordt gedelegeerd naar een operatie in een andere part in de aggregatie. Wanneer een operatie niet wordt gedelegeerd, zullen de begin- en eindtoestanden moeten worden afgebeeld op dezelfde toestand in het part.



Deze twee mogelijkheden vormen de basis van de relatie die wordt gedefinieerd tussen het toestandsdiagram van de whole en de toestandsdiagrammen van de parts.

De beschrijving van de relatie tussen de toestandsdiagrammen wordt in een aantal stappen gedaan:

- Er wordt onderscheid gemaakt tussen de operaties van de whole die behoren tot het eigen gedrag van de whole en operaties die worden gedelegeerd naar één van de parts.
- Voor de operaties die worden gedelegeerd wordt een functie gedefinieerd die aangeeft naar welke operatie van een part deze operatie wordt gedelegeerd.
- De operaties worden gerelateerd aan de toestandsovergangen waarmee ze corresponderen.
- De vorige twee stappen worden gecombineerd tot een relatie tussen de toestandsovergangen in de whole en die in de parts.
- Er wordt een relatie gelegd tussen de toestanden in het toestandsdiagram van de whole en die in de parts.

Onderscheid in typen operaties

De eerste stap is dat er onderscheid moet worden gemaakt tussen de verschillende typen operaties in de whole:

- Lokale operaties. Deze operaties behoren tot het eigen gedrag van de whole en zijn ook geïmplementeerd in de whole.
- Gedelegeerde operaties. Deze operaties worden door de whole gedelegeerd naar één van de parts. Deze operaties zijn dus geïmplementeerd in één van de parts.

De lokale operaties worden in het vervolg de l-operaties genoemd en de gedelegeerde operaties worden de d_p-operaties genoemd. De parameter p geeft aan naar welk part de operaties gedelegeerd worden, zodat er voor elk part een verzameling d-operaties is in de whole.

Delegatie-functie

De d_p-operaties bestaan dus uitsluitend uit een aanroep van een operatie van part p. De d_p-operaties en de corresponderende operaties in het part hoeven niet dezelfde naam te hebben. Daarom wordt er een functie gedefinieerd die de d_p-operaties afbeeldt op de operaties die ze aanroepen in part p. Op deze manier ontstaat een afbeelding van een deel van de operaties van de whole (de d_p-operaties) op de operaties van de parts.

Relatie tussen operaties en toestandsovergangen

Elke operatie correspondeert met één of meer toestandsovergangen. De aanroep van een operatie leidt daardoor tot een toestandsovergang. Daarom kan er een relatie worden gelegd tussen operaties en de bijbehorende toestandsovergangen.

Relatie tussen toestandsovergangen

Door de delegatie-functie zijn operaties in de whole gerelateerd aan operaties in de parts. Deze operaties zijn vervolgens weer gerelateerd aan toestandsovergangen. Hieruit volgt dat er een relatie bestaat tussen toestandsovergangen in het toestandsdiagram van de whole en toestandsovergangen in het toestandsdiagram van de parts.

Relatie tussen toestanden (homomorfisme)

Om de relatie tussen de toestanden te bepalen moeten alle operaties in de whole worden bekeken. Er zijn dan twee situaties te onderscheiden:

- De operatie in de whole heeft een relatie met een operatie in het part. De operatie wordt dus gedelegeerd naar het part.
- De operatie heeft geen relatie met een operatie in het part. De operatie is dus onderdeel van het eigen gedrag van de whole, of wordt gedelegeerd naar een ander part.

In de eerste situatie wordt de operatie wordt gedelegeerd naar een operatie in het part. De begin- en eindtoestanden worden in dat geval gerelateerd aan de begin- en eindtoestanden van de corresponderende operatie in het part.

In de tweede situatie wordt de operatie niet gedelegeerd. De begin- en eindtoestanden moeten dan worden gerelateerd aan één toestand in het part.

Het kan nu zo zijn dat dit laatste niet mogelijk is. Wanneer twee of meer toestanden met elkaar verbonden zijn door toestandsovergangen die niet worden gedelegeerd naar het betreffende part, dan moeten deze toestanden op dezelfde toestand in het part worden afgebeeld. Het kan echter zo zijn dat twee of meer van deze toestanden ook begin -of eindtoestand zijn van operatie die wel wordt gedelegeerd naar het betreffende part. Op grond daarvan moeten deze twee toestanden op twee verschillende operaties worden afgebeeld.

Tot slot moet nog gelden dat de starttoestand van het toestandsdiagram van de whole moet worden afgebeeld op de starttoestand van het toestandsdiagram van het part.

4.3.3 Bepaling correctheid

Er moet nu worden nagegaan of er geen conflict kan ontstaan, wanneer de in de vorige paragraaf beschreven afbeelding (homomorfisme) gemaakt kan worden. Er is sprake van een conflict wanneer een bepaalde reeks van achtereenvolgens aan te roepen operaties is toegestaan volgens het toestandsdiagram van de whole, maar niet in het toestandsdiagram van het part.

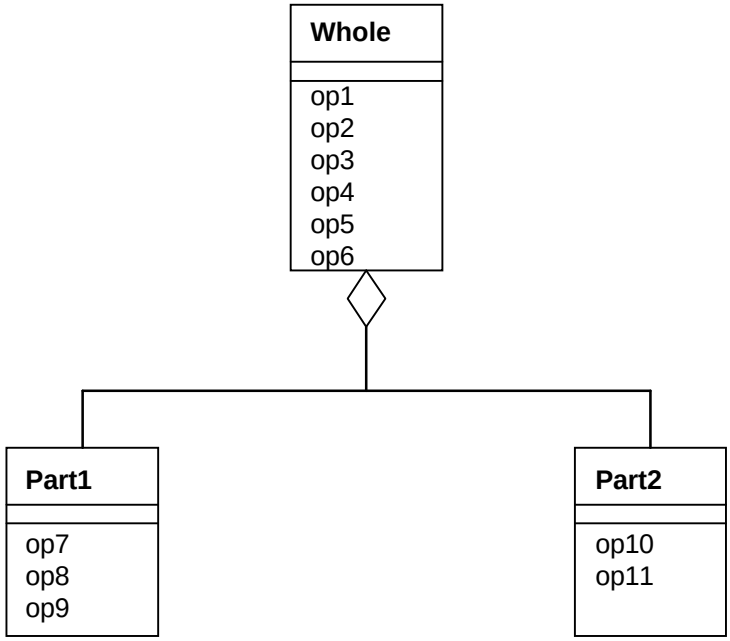
Doordat er een relatie ligt tussen de operaties en toestandsovergangen van de whole en die van de parts kan voor een concrete reeks operaties in de whole worden afgeleid welke reeks voor elk van de parts ontstaat. Dit kan op de volgende manier:

- Verwijder alle operaties uit de reeks die niet worden gedelegeerd naar het betreffende part p (dus niet behoren tot de d_p-operaties). Hiermee worden dus ook de lokale operaties (d_l-operaties) verwijderd.
- Vervang alle d_p-operaties door de operaties waarop ze zijn afgebeeld voor het betreffende part.

De reeks van achtereenvolgens aan te roepen operaties moet nu toegestaan zijn volgens het toestandsdiagram van het betreffende part. Deze correctheid kan worden aangetoond, door vast te stellen dat een toestandsovergang in het part alleen kan plaatsvinden als de bijbehorende operatie in de whole wordt aangeroepen. Wanneer er in het eigen gedrag (of het gedrag dat gedelegeerd wordt naar een ander part) een mogelijkheid is om van de begin- naar de eindtoestand dan ontstaat er een conflict en kan de afbeelding niet worden gemaakt.

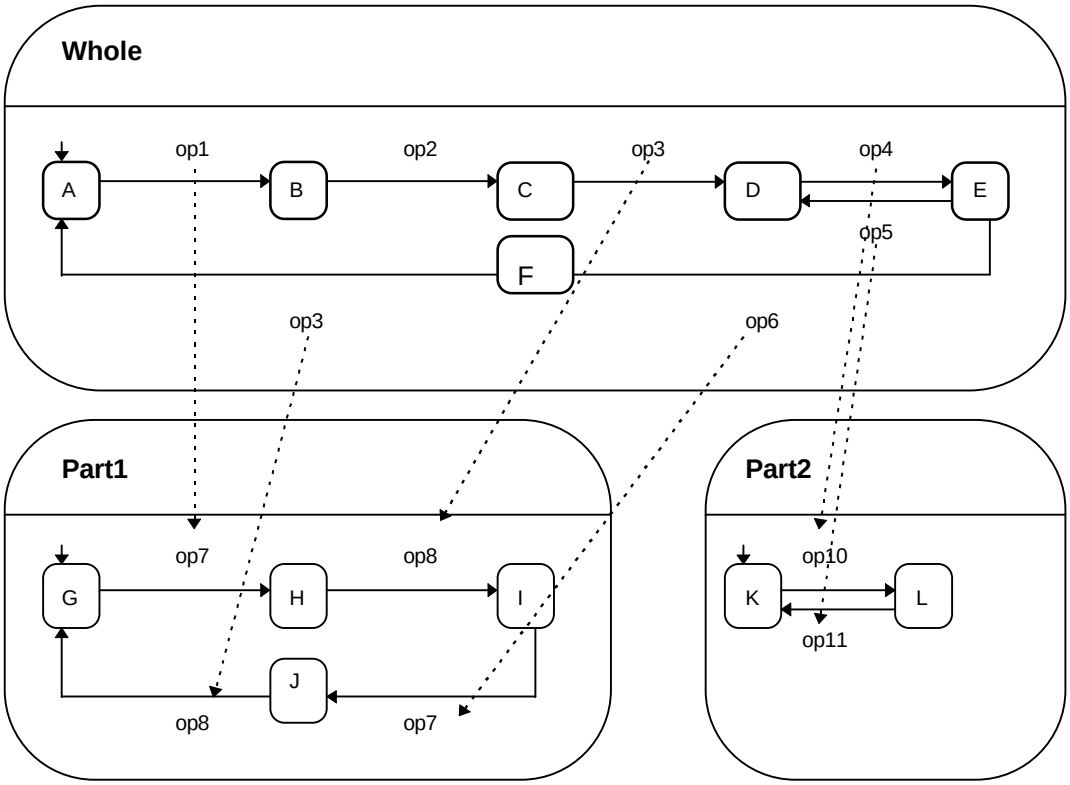
4.3.4 Voorbeeld

Object model:



Dynamisch model:

De delegatie van operaties is in dit diagram als gestippelde pijlen toegevoegd.



Onderscheid in typen operaties.

Voor wat betreft de operaties in de whole wordt onderscheid gemaakt in de lokale operaties (l-operaties, de operaties die worden gedelegeerd naar part1 (d_{part1}-operaties) en de operaties die worden gedelegeerd naar part2 (d_{part2}-operaties).

l-operaties: {op2}

d_{part1}-operaties: {op1, op3, op6}

d_{part2}-operaties: {op4, op5}

Delegatie-functie.

De d_{part1}-operaties en d_{part2}-operaties worden afgebeeld op de operaties in de parts die aangeroepen worden.

Voor de d_{part1}-operaties geldt:

op1 → op7

op3 → op8

op6 → op7

En voor de d_{part2}-operaties geldt:

op4 → op10

op5 → op11

Relatie tussen operaties en toestandsovergangen.

De relatie tussen operaties en toestandsovergangen is triviaal omdat de toestandsovergangen in het toestandsdiagram de naam krijgen van de betrokken operatie. Voor de volledigheid wordt hier de relatie toch weergegeven.

Voor de whole geldt:

op1 → (A, B)

op2 → (B, C)

op3 → {(C, D), (F, A)}

op4 → (D, E)

op5 → (E, D)

op6 → (E, F)

Voor part1 geldt:

op7 → {(G, H), (I, J)}

op8 → {(H, I), (J, G)}

En voor part2 geldt:

op10 → (K, L)

op11 → (L, K)

Relatie tussen toestandsovergangen

De toestandsovergangen in het toestandsdiagram van de whole worden gerelateerd aan de toestandsovergangen van de parts.

Voor part1 geldt:

(A, B) → {(G, H), (I, J)}

(C, D) → {(H, I), (J, G)}

(E, F) → {(G, H), (I, J)}

(F, A) → {(H, I), (J, G)}

En voor part2 geldt:

(D, E) → {(K, L)}

(E, D) → {(L, K)}

Relatie tussen toestanden

Eerst worden de starttoestanden van de toestandsdiagrammen aan elkaar gerelateerd:

Voor part1 geldt:

A → G

Voor part 2 geldt:

A → K

Vervolgens worden de toestandsovergangen bekeken. Hierbij worden de volgende regels toegepast:

- Wanneer de toestandsovergang in de whole een relatie heeft met één of meer toestandsovergangen van het part dan worden begin- en eindtoestanden gerelateerd aan de corresponderende begin- en eindtoestanden in het part. Dit zijn de toestandsovergangen waarvan de operatie wordt gedelegeerd naar het part (d_p-operaties).
- Wanneer de toestandsovergang in de whole geen relatie heeft met een toestandsovergang in het part, dan worden begin- en eindtoestand op dezelfde toestand in het part afgebeeld. Dit zijn de toestandsovergangen waarvan de operaties niet worden gedelegeerd naar het part (l-operaties of d_p-operaties van een ander part).

Voor part1 geldt:

Er geldt (A, B) → {(G, H), (I, J)} en A → G, dus A → G en B → H

Er geldt (B, C) heeft geen relatie met part1 en B → H, dus C → H

Er geldt (C, D) → {(H, I), (J, G)} en C → H, dus C → H en D → I

Er geldt (D, E) heeft geen relatie met part1 en D → I, dus E → I

Er geldt (E, D) heeft geen relatie met part 1 en E → I, dus D → I

Er geldt (E, F) → {(G, H), (I, J)} en E → I, dus E → I en F → J

Er geldt (F, A) → {(H, I), (J, G)} en F → J, dus F → J en A → G

Hieruit blijkt dat elke toestand in de whole kan worden afgebeeld op precies één toestand in het part. Samengevat ziet de afbeelding er als volgt uit:

Whole	A	B	C	D	E	F
Part 1	G	H	H	I	I	J

Voor part2 geldt:

Er geldt (A, B), (B, C) en (C, D) hebben geen relatie met part2 en A → K, dus A, B, C en D → K

Er geldt (D, E) → (K, L) en D → K, dus D → K en E → L

Er geldt (E, D) → (L, K) en E → L, dus E → L en D → L

Dit laatste is in tegenspraak met de reeds gevonden afbeelding D → K. Voor part2 is het dus niet mogelijk om een afbeelding van de toestanden van de whole op die van het part te maken.

Bepaling correctheid.

Uit bovenstaande afbeelding volgt dat het toestandsdiagram van de whole wel correct is in relatie tot het toestandsdiagram van part1 maar niet in relatie tot het toestandsdiagram van parts. Dit betekent dat er voor elke toegestane reeks van achtereenvolgens aan te roepen operaties in de whole een correcte reeks voor part1 ontstaat, maar dat dat niet geldt voor part2.

Neem nu als voorbeeld de reeks (op1, op2, op3, op4, op5, op4, op6, op3). Als alle operaties die niet in d_{part1} zitten worden verwijderd ontstaat de reeks (op1, op3, op6, op3). Als vervolgens de delegatie-functie wordt toegepast ontstaat de reeks (op7, op8, op7, op8). Dit is een correctie reeks voor part1. Op dezelfde manier ontstaat voor part2 de reeks (op9, op10, op9). Dit is een ook een correctie reeks voor part2. Een ander voorbeeld is echter de reeks (op1, op2, op3, op4, op6, op3, op1, op2, op3, op4). Dit levert voor part1 de reeks (op7, op8, op7, op8) op en voor part2 de reeks (op9, op9). De reeks is wel correct voor part1 maar niet voor part2.

4.4 Formele beschrijving

Deze formele beschrijving maakt gebruik van de volgende definities uit [EbeEng 95]:

- STDiagram
- Class
- _ ⊢ _ _ (De verzameling mogelijke toestandsovergangen voor een bepaalde class)
- R (Verzameling toegestane reeksen van toestanden voor een bepaalde class)
- _correspondsTo_ (De reeks operaties die overeenkomt met een reeks toestanden)
- S (Verzameling toegestane reeksen van operaties voor een bepaalde class)

Deze basisdefinities zullen eerst beschreven worden.

Daarna zullen een aantal relaties uit de informele beschrijving formeel worden beschreven. De essentie is om te komen tot een functie die de toestanden van de whole afbeeldt op die van het part. Dit homomorfisme wordt uiteindelijk gebruikt om correctheid aan te kunnen tonen.

4.4.1 Basisdefinities

Gegeven verzamelingen

In de nu volgende Z-specificaties wordt gebruikt gemaakt van de verzamelingen ID en STATE. De verzameling ID is de verzameling identifiers van operaties en de verzameling STATE is de verzameling toestanden.

$$[ID, STATE]$$

Toestandsdiagram

Het toestandsdiagram wordt beschreven als een eindige automaat. De verzameling Q stelt de verzameling toestanden voor. De toestand q_0 is de starttoestand. Σ is de verzameling operaties en δ is de functie voor de toestandsovergangen.

STDiagram

$$Q : \mathbf{F}\ STATE$$

$$\Sigma : \mathbf{F}\ ID$$

$$\delta : \mathbf{F}\ ((Q \times \Sigma) \times Q)$$

$$q_0 : Q$$

Class

Een class wordt alleen beschreven voor wat betreft het dynamisch model. Attributen worden buiten beschouwing gelaten.

Class

$M : \mathbf{F}\ ID$

$STD : STDiagram$

$STD.\Sigma = M$

Gedrag

De functie $\vdash\!\!\!-$ beschrijft voor een bepaalde class de mogelijke toestandsovergangen. Wanneer er dus in een class C ten minste één operatie is die een toestandsovergang van q1 naar q2 kan veroorzaken, dan geldt de relatie $q_1 \vdash\!\!\!-_C q_2$.

$(\_ \vdash\!\!\!-_{\_} \_) : Class \rightarrow (STATE \leftrightarrow STATE)$

$\forall\ C : Class;\ q, \hat{q} : STATE \bullet$

$q \vdash\!\!\!-_C \hat{q}$

$\Leftrightarrow$

$\{q, \hat{q}\} \subseteq C.STD.Q \wedge$

$(\exists\ m : C.STD.\Sigma \bullet ((q, m), \hat{q}) \in C.STD.\delta \vee q = \hat{q})$

Toegestane reeksen van toestanden

Uit de definitie van het gedrag volgt dat er een verzameling kan worden gedefinieerd die alle toegestane reeksen van toestanden bevat. Als bijvoorbeeld voor een bepaalde class C met begintoestand q0 geldt, dat $q_0 \vdash\!\!\!-_C q_1$ en $q_1 \vdash\!\!\!-_C q_2$, dan is de reeks $(q_0, q_1, q_2) \in R(C)$.

$R : Class \rightarrow \mathbf{P}\ seq\ STATE$

$\forall\ C : Class;\ r : seq\ STATE \bullet$

$r \in R(C)$

$\Leftrightarrow$

$r \neq \langle\rangle \wedge$

$r(1) = C.STD.q_0 \wedge$

$\forall\ i : 1..(\#r - 1) \bullet r(i) \vdash\!\!\!-_C r(i + 1)$

Corresponderende operaties

Met de hierboven genoemde reeksen van toestanden corresponderen reeksen van achtereenvolgens aan te roepen operaties. Met elke reeks toestanden correspondeert een reeks operaties.

$$(\_\text{correspondsTo}_{\_}\ \_) : \text{Class} \rightarrow (\text{seq ID} \leftrightarrow \text{seq STATE})$$

$$\forall\ C : \text{Class};\ s : \text{seq ID};\ r : \text{seq STATE} \bullet$$

$$s\ \text{correspondsTo}_C\ r$$

$$\Leftrightarrow$$

$$s = \langle\rangle \wedge \#r = 1 \quad \vee$$

$$((r(1), s(1)), r(2)) \in C.\text{STD}.\delta \wedge$$

$$s(2..)\ \text{correspondsTo}_C\ r(2..)$$

Toegestane reeksen van operaties

Wanneer de voorgaande twee definities worden gecombineerd kan de verzameling van toegestane reeksen van achtereenvolgens aan te roepen operaties worden gedefinieerd.

$$S : \text{Class} \rightarrow \mathbf{P}\ \text{seq ID}$$

$$\forall\ C : \text{Class};\ s : \text{seq ID} \bullet$$

$$s \in S(C)$$

$$\Leftrightarrow$$

$$\exists\ r : R(C) \bullet s\ \text{correspondsTo}_C\ r$$

4.4.2 Definities voor aggregaties

Aggregatie-relatie

De isPartOf relatie definieert een aggregatie-relatie tussen twee classes.

$$(\_isPartOf\_) : Class \leftrightarrow Class$$

Delegatie-functie

De delegatesTo relatie definieert de situatie dat een operatie in de whole wordt gedelegeerd naar een operatie in een part. In deze definitie is C de whole en C' het part (C' isPartOf C) en wordt operatie m in class C gedelegeerd naar operatie m' in C'.

$$(\_delegatesTo_{\_,\_}\ \_) : (Class \leftrightarrow Class) \rightarrow ID \rightarrow ID$$

$$\forall\, C, C' : Class;\ m, m' : ID \mid m\ delegatesTo_{C,C'}\ m' \bullet C'\ isPartOf\ C \wedge m \in C.M \wedge m' \in C'.M$$

Bepaling d_p-operaties

Deze functie geeft aan welke operaties in de whole (C) gedelegeerd worden naar een bepaald part (C'). Hiervoor moet dus gelden dat C de whole is en C' het part (C' isPartOf C). De verzameling $d_{C,C'}$ is gelijk aan het domein van $delegatesTo_{C,C'}$. Dit zijn alle operaties m in C die worden gedelegeerd naar een operatie m' in C'.

$$d_{\_,\_} : (Class \leftrightarrow Class) \rightarrow \mathbf{F}\ ID$$

$$\forall\, C, C' : Class \bullet d_{C,C'} = dom\ delegatesTo_{C,C'}$$

Homomorfisme

Onderstaand homomorfisme geeft een totale afbeelding van de toestanden van de whole op de toestanden van het part. Voor elk part wordt zo'n homomorfisme gedefinieerd.

Elke toestandsovergang in de whole wordt bekeken. Wanneer de operatie behorend bij de toestandsovergang wordt gedelegeerd naar het part ($m \in d_{C,C'}$) dan worden de bijbehorend begin- en eindtoestanden afgebeeld op de begin- eindtoestanden van de corresponderende toestandsovergang in het part [a].

Wanneer de operatie behorend bij de toestandsovergang niet wordt gedelegeerd, dan worden de begin- en eindtoestanden op dezelfde toestand in het part afgebeeld [b].

$$\text{isHomomorphism} : (\text{STATE} \to \text{STATE}) \leftrightarrow \text{Class} \leftrightarrow \text{Class}$$

$$\begin{aligned}
&\forall\, h : \text{STATE} \to \text{STATE};\ C, C' : \text{Class} \bullet \\
&\quad \text{isHomomorphism}(h, C, C') \\
&\quad \Leftrightarrow \\
&\qquad h \in C.\text{STD.Q} \to C'.\text{STD.Q} \quad \wedge \\
&\qquad C' \text{ isPartOf } C \quad \wedge \\
&\qquad h(C.\text{STD.q}_0) = C'.\text{STD.q}_0 \quad \wedge \\
&\qquad \forall\, q, \hat{q} : C.\text{STD.Q};\ m : C.\text{STD.}\Sigma \bullet \\
&\qquad\quad ((q, m), \hat{q}) \in C.\text{STD.}\delta \\
&\qquad\quad \Rightarrow \\
&\qquad\qquad m \in d_{C,C'} \quad \wedge \\
&\qquad\qquad\quad m \text{ delegatesTo } m' \quad \wedge \\
&\qquad\qquad\quad ((h(q), m'), h(\hat{q})) \in C'.\text{STD.}\delta \quad \vee \qquad [a] \\
&\qquad\qquad m \notin d_{C,C'} \quad \wedge \\
&\qquad\qquad\quad h(q) = h(\hat{q}) \qquad [b]
\end{aligned}$$

Correctheid

Op basis van dit homomorfisme moet kunnen worden aangetoond dat het toestandsdiagram van de whole (C) correct is in relatie tot het toestandsdiagram van het part (C').

Voor elke toegestane reeks van operaties in de whole moet gelden dat er een bijbehorende toegestane reeks in het part bestaat. Omdat het homomorfisme een afbeelding van toestanden (en niet van operaties) is, zal dit in twee stappen worden bewezen.

- Als een homomorfisme bestaat dan kan elk toegestaan gedrag in de whole worden afgebeeld op toegestaan gedrag in het part. Onder toegestaan gedrag wordt hier een toegestane reeks toestanden verstaan. Dit is stelling 1.
- Toegestaan gedrag (een reeks toestanden) leidt tot een toegestane reeks van achtereenvolgens aan te roepen operaties. Dit is stelling 2.

Stelling 1

$$\forall\, C, C' : Class;\ h : STATE \rightarrow STATE \bullet$$
$$isHomomorphism(h, C, C')$$
$$\Rightarrow$$
$$\forall\, q, \hat{q} : C.STD.Q \bullet$$
$$q \vdash_C \hat{q} \Rightarrow h(q) \vdash_{C'} h(\hat{q})$$

Bewijs

Stel

$$q \vdash_C \hat{q}$$

Hetgeen betekent

$$\{q, \hat{q}\} \subseteq C.STD.Q \wedge$$
$$(\exists\, m : C.STD.\Sigma \bullet ((q, m), \hat{q}) \in C.STD.\delta)$$

In het toestandsdiagram van de whole is er dus een toestandsovergang van q naar $\hat{q}$ veroorzaakt door de operatie m. Op grond van het homomorfisme h betekent dit het volgende voor het toestandsdiagram van het part.

$$\{h(q), h(\hat{q})\} \subseteq C'.STD.Q \wedge$$
$$(\exists\, m : C.STD.\Sigma \bullet$$
$$m \in d_{C,C'} \wedge$$
$$m\ delegatesTo\ m' \wedge$$
$$((h(q), m'), h(\hat{q})) \in C'.STD.\delta \vee$$
$$m \notin d_{C,C'} \wedge$$
$$h(q) = h(\hat{q}))$$

Dit betekent dat er een corresponderende toestandsovergang in het part is, of dat q en $\hat{q}$ worden afgebeeld op dezelfde toestand in het part.

Hieruit volgt

$$h(q) \vdash_{C'} h(\hat{q})$$

Het gevolg hiervan is, dat dit ook geldt voor de verzameling van toegestane reeksen van toestanden.

Gevolg 1

$$r \in R(C) \Rightarrow h \circ r \in R(C')$$

Nu is vastgesteld dat elke toegestane reeks van toestanden in de whole correspondeert met een toegestane reeks van toestanden in het part als er een homomorfisme bestaat, moet vervolgens worden vastgesteld, dat dit tevens leidt tot toegestane reeksen van achtereenvolgens aan te roepen operaties.

Stelling 2

$$\forall\, C, C' : \text{Class};\ h : \text{STATE} \to \text{STATE} \bullet$$
$$\text{isHomomorphism}(h, C, C')$$
$$\Rightarrow$$
$$(s \in S(C)) \Rightarrow$$
$$\exists\, s' \in S(C') \bullet t = s \upharpoonright d_{C,C'} \Rightarrow \forall\, i : 1..\#t \bullet t(i)\ \text{delegatesTo}\ s'(i)$$

Bewijs

Stel, er zijn (in de whole) reeksen toegestane operaties $s \in S(C)$ en reeksen toegestane toestanden $r \in R(C)$, zodanig dat $s\ \text{correspondsTo}_C\ r$.

Als gevolg van Gevolg 1 geldt dan ook (in het part) dat er reeksen toegestane toestanden $r' \in R(C')$ zijn met $r' = h \circ r$, ofwel

$$\forall\, i : 1..\#r \bullet r'(i) = h(r(i))$$

Laat $t == s \upharpoonright d_{C,C'}$

en $s' == \forall\, i : 1..\#t : t(i)\ \text{delegatesTo}\ s'(i)$

Dan moeten als gevolg van de definitie van de correspondsTo-relatie twee situaties worden onderscheiden:

Situatie 1

$\#r' = 1$, dan is ook $\#r = 1$ en dus $s = \langle\rangle = s'$

Situatie 2

$\#r' > 1$

Als er een toestandsovergang in de whole is van toestand r(1) naar r(2) via operatie s(1):

$((r(1), s(1)), r(2)) \in C.STD.\delta$

dan moeten de twee situaties (a en b) uit het homomorfisme worden onderscheiden:

Situatie 2a

Als de operatie s(1) gedelegeerd wordt naar het betreffende part dan geldt:

$s(1) \in d_{C,C'} \quad \wedge$

$s(1)\ delegatesTo\ s'(1) \quad \wedge$

$s'(1) \in C'.M$

Volgens het homomorfisme volgt daaruit dat:

$((r'(1), s'(1)), r'(2)) \in C'.STD.\delta$

Situatie 2b

Als de operatie s(1) niet gedelegeerd wordt naar het betreffende part dan geldt:

$s(1) \notin d_{C,C'}$

Volgens het homomorfisme volgt daaruit dat:

$h(r(1)) = h(r(2))$ en dus dat $r'(1) = r'(2)$

Bij elkaar levert dit op:

$s' = \langle\rangle \wedge \#r' = 1 \quad \vee$ [1]

$s(1) \in d_{C,C'} \quad \wedge$

$((r'(1), s'(1)), r'(2)) \in C'.STD.\delta \quad \vee$ [2a]

$s(1) \notin d_{C,C'} \quad \wedge$

$r'(1) = r'(2)$ [2b]

Als nu als inductiehypothese wordt gebruikt:

$s'(2..)\ correspondsTo_{C'}\ r'(2..)$

dan leidt dit via inductie tot:

$s'\ correspondsTo_{C'}\ r'$

4.5 Vergelijking met inheritance

Het hierboven gedefinieerde homomorfisme is afgeleid van het homomorfisme dat is gedefinieerd voor een inheritance-relatie in [EbeEng 95]. De belangrijkste verschillen tussen deze twee homomorfismen worden hier kort beschreven.

Het homomorfisme voor een inheritance-relatie beeldt de toestanden in het toestandsdiagram van elk subtype af op de toestanden in het toestandsdiagram van het supertype. Het homomorfisme voor een aggregatie-relatie beeldt de toestanden in het toestandsdiagram van de whole af op de toestanden in het toestandsdiagram van elk van de parts. Dit is in overeenstemming met de automatische omzetting van een inheritance-relatie in een aggregatie-relatie zoals dat in hoofdstuk 3 is beschreven. De whole vervult de rol van het subtype en het part vervult de rol van het supertype.

Een ander aspect dat bij inheritance een rol speelt zijn ε-overgangen. Deze spontane toestandsovergangen ontstaan in het supertype, wanneer er een operatie in één van de subtypen wordt aangeroepen die niet is geïmplementeerd in het supertype. Vanuit het supertype gezien ontstaat er dan een spontane toestandsovergang. Deze situatie doet zich niet voor bij aggregatie-relaties. In een aggregatie kan een toestandsovergang alleen plaatsvinden door een expliciete aanroep van een operatie.

Een andere verschil is, dat de relatie tussen de operaties van super- en subtypen eenvoudiger is dan bij een aggregatie. Een operatie in het subtype heeft een relatie met een operatie in het supertype als de operatie overgeërfd is van het supertype. Het gaat in zo'n geval dus gewoon om precies dezelfde operatie. Bij een aggregatie ligt er een relatie tussen een operatie in de whole en een operatie in een part wanneer de operatie in de whole wordt gedelegeerd naar deze operatie in het part. Er wordt dus gebruik gemaakt van informatie over de inhoud van de operaties in de whole.

4.6 Toepassing formele beschrijving

Tot slot van de formele beschrijving wordt deze beschrijving toegepast op een aantal voorbeelden uit het tweede en derde hoofdstuk.

4.6.1 Assembly

Wanneer het voorbeeld van het type Assembly in paragraaf 2.2.4 wordt bekeken blijkt dat de functie h (homomorfisme) er als volgt uit komt te zien.

Car	Uncheck	Checked	Starting	Forward	Neutral	Reverse	Brake F	Brake N	Brake R
Ignition	Off	Off	Starting	On	On	On	On	On	On
Transmission	Neutral	Neutral	Neutral	Forward	Neutral	Reverse	Forward	Neutral	Reverse
Brake	Off	Off	Off	Off	Off	Off	On	On	On

Volgens de definitie van het homomorfisme moeten er nu een aantal zaken gelden:

Totale functie

Het homomorfisme is een totale functie, dus elke toestand in de whole wordt op precies één toestand in elk van de parts afgebeeld.

$$h \in C.STD.Q \rightarrow C'.STD.Q$$

Voor bovengenoemde afbeelding geldt dat.

Starttoestanden

De starttoestand van het toestandsdiagram van de whole wordt afgebeeld op de starttoestanden in elk van de parts.

$$h(C.STD.q_0) = C'.STD.q_0$$

In de bovengenoemde afbeelding is de starttoestand van de whole Unchecked, en deze toestand wordt op de starttoestanden van Ignition (Off), Transmission (Neutral) en Brake (Off) afgebeeld.

Gedelegeerde operaties

Voor de toestandsovergangen waarbij de operatie (m) wordt gedelegeerd (naar m'), moet gelden dat de afbeelding de begin- en eindtoestand (q en $\hat{q}$) op een corresponderende begin- en eindtoestand ($h(q)$ en $h(\hat{q})$) afbeeldt.

$$((q, m), \hat{q}) \in C.STD.\delta \Rightarrow ((h(q), m'), h(\hat{q})) \in C'.STD.\delta$$

Dit kan voor elke gedelegeerde toestandsovergang worden gecontroleerd. We bekijken hier als voorbeeld de toestandsovergang ((Neutral, Turn key off), Checked) en het homomorfisme voor part Ignition.

h(Neutral) = On

h(Checked) = Off

Turn key off delegatesTo Turn key off

Nu moet dus ((On, Turn key off), Off) een toestandsovergang zijn in het part Ignition.

Niet gedelegeerde operaties

Voor de toestandsovergangen waarbij de operatie niet wordt gedelegeerd, moet gelden dat de begin- en eindtoestand (q en $\hat{q}$) op dezelfde toestanden in het part worden afgebeeld.

$$((q, m), \hat{q}) \in C.STD.\delta \Rightarrow h(q) = h(\hat{q})$$

Bekijken we de toestandsovergang ((Unchecked), Check), Checked) in het eigen gedrag van Car, dan moet voor elk van de parts gelden dat h(Unchecked) = h(Checked).

4.6.2 State refinement

Wanneer het voorbeeld van State Refinement in paragraaf 3.3.1 wordt bekeken blijkt dat de functie h (homomorfisme) er als volgt uit komt te zien.

Traditional Person	Not Engaged	Engaged	Married
Person	Single	Single	Married

Ook hier is eenvoudig na te gaan dat de functie h een totale functie is.

De starttoestand in Traditional Person (Not Engaged) wordt inderdaad afgebeeld op de starttoestand in Person (Single).

Onder de afbeelding h corresponderen de gedelegeerd operaties als volgt:

((Engaged, Gets married), Married) → ((Single, Gets married), Married)

((Married, Gets divorced), Not engaged) → ((Married, Gets divorced), Single)

((Married, Birtday), Married) → ((Married, Birthday), Married)

((Not engaged, Birtday), Not engaged) → ((Single, Birthday), Single)

((Engaged Birtday), Engaged) → ((Single, Birthday), Single)

Voor de niet gedelegeerde operatie Gets engaged zou moeten gelden dat de begin- en eindtoestanden op dezelfde toestand worden afgebeeld. De toestanden Not engaged en Engaged worden inderdaad beide op Single afgebeeld.

4.6.3 Adding transitions

Wanneer het voorbeeld van Adding transitions in paragraaf 3.4.1 wordt bekeken lijkt het zeer voor de hand te liggen om de volgende afbeelding (homomorfisme) te kiezen.

Tv Remote	Ch-1	Ch-2	Ch-3	Ch-4
Tv	Ch-1	Ch-2	Ch-3	Ch-4

Dit is een totale functie, waarbij de starttoestanden (Ch-1 in beide gevallen) op elkaar zijn afgebeeld. Ook alle gedelegeerd operaties corresponderen onder de afbeelding. Dit is in dit voorbeeld heel eenvoudig te zien omdat alle namen van toestanden en toestandsovergangen gelijk gehouden zijn.

Neem als voorbeeld:

((Ch-1, Up), Ch-2) $\rightarrow$ ((Ch-1, Up), Ch-2))

Het probleem ontstaat echter met de regel die geldt voor de niet gedelegeerde operaties.

$$((q, m), \hat{q}) \in C.STD.\delta \Rightarrow h(q) = h(\hat{q})$$

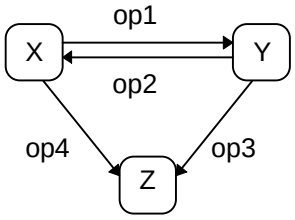
De toestandsovergang in de whole ((Ch-1, Channel 2), Ch-2) zou betekenen dat Ch-1 en Ch-2 op dezelfde toestanden moeten zijn afgebeeld. Dat is helaas niet het geval.

4.7 Kanttekeningen

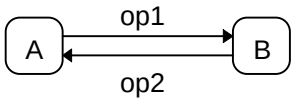
In deze formele beschrijving is nu aangetoond dat, wanneer een homomorfisme gedefinieerd kan worden, het toestandsdiagram van de whole correct is in relatie tot dat van de parts. Ook is gebleken dat in de gebruikte voorbeelden inderdaad blijkt dat het homomorfisme voldoet. Het blijft echter de vraag of het homomorfisme niet een te sterke eis is voor correctheid.

Het homomorfisme zorgt ervoor dat er in het toestandsdiagram van de whole geen (reeks van) toestandsovergangen die niet gedelegeerd worden parallel lopen aan toestandsovergangen die wel gedelegeerd worden. Dit kan met onderstaand voorbeeld worden geïllustreerd.

Toestandsdiagram Whole:



Toestandsdiagram Part:



In dit voorbeeld zijn er weliswaar twee niet gedelegeerde toestandsovergangen op3 en op4 die parallel lopen aan de gedelegeerde toestandsovergangen op1 en op2. Er is echter geen pad van X naar Y of andersom via niet gedelegeerd operaties. In dit voorbeeld kan er geen homomorfisme worden gedefinieerd, terwijl toch kan worden vastgesteld dat er geen enkele reeks van toegestane operaties in de whole zal leiden tot een niet toegestane reeks operaties in het part.

Uit dit voorbeeld blijkt dat er situaties denkbaar zijn waarin het homomorfisme een te sterke eis is voor correctheid. Als in dit voorbeeld de whole en het part worden vervangen door subtype respectievelijk supertype, dan ontstaat een voorbeeld voor een inheritance-relatie waarvoor het homomorfisme in [EbeEng 95] een te sterke eis is voor correctheid.

5 Samenvatting en conclusies

Het doel van deze scriptie was om te onderzoeken wat de relatie is tussen aggregaties in het object model en het bijbehorende model.

Dit onderzoek is in een aantal stappen uitgevoerd. Allereerst is op basis van bestaande classificaties van aggregaties in [CoYo 91a/b] en [RuBlPr 91] een classificatie gemaakt die aansluit bij het doel van het onderzoek. Van elk type aggregatie is mede aan de hand van een voorbeeld bekeken wat de afhankelijkheden en beperkingen zijn in het dynamisch model van de whole en de parts in de aggregatie.

Een belangrijke conclusie uit deze eerste stap was, dat er alleen iets gezegd kan worden over afhankelijkheden tussen het dynamisch model van de whole en dat van de parts als de whole de interface biedt voor de gehele aggregatie. Clients kunnen in zo'n geval dus niet rechtstreeks operaties van parts aanroepen. In plaats daarvan worden bepaalde operaties van de whole gedelegeerd naar één van de parts.

Vervolgens kon worden afgeleid dat wanneer het toestandsdiagram van de whole is geconstrueerd als logische AND of als cartesisch produkt van de toestandsdiagrammen van de parts en het eigen gedrag van de whole, het toestandsdiagram van de whole correct is in relatie tot dat van de parts. Dit blijft ook zo als er beperkingen bij de toestandsovergangen van de whole worden toegevoegd of als er toestandsovergangen worden verwijderd.

De tweede stap in het onderzoek was om de relatie met inheritance te onderzoeken. Het is mogelijk gebleken om een inheritance-relatie automatisch om te zetten naar een aggregatie-relatie. Op deze manier is het mogelijk om de resultaten uit het onderzoek in [EbeEng 95] te bekijken. Het type Parallel Extension blijkt de conclusies met betrekking tot de correctheid van de logische AND te bevestigen. Het type State Refinement introduceert een nieuw type waaruit blijkt dat het toestandsdiagram van de whole ook correct is wanneer daarin een toestand van een part is gesplitst in meerdere toestanden, zolang de niet gedelegeerd operaties zich maar beperken tot overgangen tussen deze toestanden. Het type Adding Transitions blijkt duidelijk niet tot een correcte aggregatie te leiden, doordat er in de whole toestandsovergangen worden toegevoegd die niet worden gedelegeerd. Deze toestandsovergangen lopen parallel aan toestandsovergangen die wel gedelegeerd worden waardoor er conflicten kunnen ontstaan.

De derde stap heeft tot doel om de gevonden beperkingen formeel te beschrijven. In deze formele beschrijving wordt voor elk part een afbeelding gemaakt van de toestanden van de whole op de toestanden het part. Dit gebeurt grofweg op dezelfde manier als in [EbeEng 95], waar een afbeelding wordt gedefinieerd van de toestanden van elk subtype op de toestanden van het supertype. Deze afbeelding wordt een homomorfisme genoemd. Wanneer dit homomorfisme bestaat kan worden bewezen dat elke toegestane reeks operaties in de whole leidt tot een toegestane reeks operaties in elk van de parts.

5.1 Vervolgonderzoek

Het onderzoek heeft zich beperkt tot het type Assembly. In dit type is de multipliciteit van de aggregatie altijd 1. In de classificatie zijn ook de typen Collection en Container onderkend waarbij de multipliciteit niet gelijk aan 1 hoeft te zijn. Als vervolg op dit onderzoek zou het nog interessant kunnen zijn om een homomorfisme te definiëren waarbij de correctheid voor de typen Collection en Container kan worden getoetst.

Delegatie is in dit onderzoek zeer beperkt gedefinieerd als de delegatie van een operatie in de whole naar precies één part. Het zou interessant kunnen zijn om delegatie naar meerdere (of alle) parts te bekijken.

Daarnaast is er in dit onderzoek geen aandacht besteed aan lokale operaties in parts. Lokale operaties in een part zouden in tegenstelling tot de gedelegeerde operaties wel door clients kunnen worden aangeroepen. Ook dit zou in een vervolgonderzoek aan de orde kunnen komen.

Tenslotte is nog onvoldoende duidelijk of het homomorfisme niet een te sterke eis voor correctheid is.

Referenties

[Chen 76] P.P. Chen, The Entity Relationship Model - Towards a Unified View of Data, ACM Transactions on Database Systems, Vol 1, No 1, pp 9-38, 1976.

[CoYo 91a] P. Coad, E. Yourdon, Object Oriented Analysis (2nd edition), Yourdon Press, Englewood Cliffs, New Jersey, 1991.

[CoYo 91b] P. Coad, E. Yourdon, Object Oriented Design, Yourdon Press, Englewood Cliffs, New Jersey, 1991.

[Dil 90] Antoni Diller, Z: an introduction to formal methods, Wiley & Sons, 1990.

[EbeEng 95] Jürgen Ebert, Gregor Engels, Specialization of Object Life Cycle Definitions. Rijksuniversiteit Leiden, juni 1995.

[GaHeJo 95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, Design Patterns, Elements of reusable object-oriented software, Addison-Wesley, 1995.

[Har 87] David Harel, Statecharts: a Visual Formalism for Complex Systems, Science of Computer Programming 8 (1987,3), 231-274.

[RuBlPr 91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, Object Oriented Modeling and Design, Prentice Hall, Englewood Cliffs NJ, 1991.

[Spiv 92] J.M. Spivey, The Z Notation (2nd edition), Prentice Hall, New York, 1992.