An Object Oriented Framework for Mapping Concurrent Applications to Parallel and Distributed Architectures

Rick van Rein[1]

July 24, 1994

[1]This is my Master's Thesis in Computer Science at the University of Twente. My graduation committee consists of the following persons:
Dr.Ir. Lodewijk Bergmans,
Dr.Ir. Mehmet Aksit,
Ir. Pierre Janssen
Dr. Satoshi Matsuoka.

Appendix A

In gewoon Nederlands

Het is natuurlijk niet mijn bedoeling om een ondoorgrondelijk afstudeerverslag te schrijven. Ik vind het erg belangrijk om ook aan niet-wetenschappers duidelijk te kunnen maken waar ik me met mijn opdracht mee bezig heb gehouden. Dat doe ik hier. Ik heb gekozen voor hele normale voorbeelden, omdat dat veel sprekender is dan wanneer ik op de gebruikelijke wijze termen uit de informatica uitleg.

A.1 Samenwerking

In mijn opdracht gaat het er om dat ik programma's bewerk voor hele grote computers, zoals supercomputers. Die computers werken volgens het principe van 'vele handen maken licht werk'. In een fabriek werken veel mensen samen, omdat dan het werk sneller geklaard is dan wanneer iemand het alleen zou moeten doen. Netzo bestaan de computers die ik voor ogen heb in mijn opdracht eigenlijk uit een kleiner aantal computertjes, die samen aan hun opdracht, hun programma, werken. Waar in de fabriek mensen staan, wordt het werk aan een programma door computers gedaan.

Het doel van veel mensen in de fabriek is, dat de fabriek sneller draait. Het is bijvoorbeeld uitstekend mogelijk om één mens al het werk te laten doen, maar als veel mensen samen werken, dat schiet het werk gewoon veel meer op. In de informatica is dat opschieten van groot belang. Sommige problemen zijn heel ingewikkeld, en duren daardoor te lang als ze door één computer worden uitgevoerd. Ook komt het voor dat heel vaak iets wordt gevraagd, en dat de computer harder moet werken omdat hij anders niet iedereen kan beantwoorden. Een jamfabriek met één werknemer kan nu eenmaal niet zo veel verzoeken om een potje jam afwerken als een fabriek met dertig werknemers.

Niet iedereen heeft in een fabriek het zelfde werk, maar toch is er samenhang: ze maken samen het product. Eén persoon staat de hele dag jam te koken, een ander vult er potjes mee, en weer een ander pakt die potjes in.

Nu zijn er wel valkuilen. We moeten voorkomen dat iemand potjes gaat inpakken voordat ze zijn gevuld. Een mens ziet dat natuurlijk meteen, maar een computer is zo slim niet. Die moeten we precies vertellen vanaf wanneer hij de potjes mag gaan inpakken, anders doet hij misschien dat te vroeg. We moeten dus een soort stoplichten maken waarop de computer moet wachten. Als de jam in de potjes zit, springt het licht op groen.

Een werkgever wil niet dat zijn mensen op stoplichten gaan zitten te wachten. Er moet doorgewerkt worden. Dus als er even gewacht moet worden op een volgende hoeveelheid potjes die ingepakt kan worden, kan de inpakker best even iets anders gaan doen, bijvoorbeeld helpen met het prakken van de aardbeien voor de jam. Zo ook willen we computers aan het werk houden, ze zijn er duur genoeg voor. Is er even niets te doen, dan kan hij mooi een ander klusje oppakken dat nog nodig is voor het programma dat loopt.

Door flink werk aan te pakken draait de fabriek dus zo goed mogelijk. Alleen moet wel voorkomen worden dat de inpakker zo druk met iets anders beziggaat, dat hij niet meer aan inpakken toekomt. Eventueel is het in zo'n geval mogelijk dat iemand anders het inpakken overneemt. Gelukkig kunnen we deze werkbereidheid ook in de computer inbouwen, zodat de samenwerking zo goed mogelijk is. We moeten alleen uitkijken dat we niet te vaak andere mensen klusjes laten opknappen, want die moeten wel steeds inwerken in dat klusje, en dat kost tijd en is dus kostbaar.

Dit is de manier waarop in de fabriek zelf wordt gewerkt, maar wat gebeurt er buiten de fabriek met de toelevering?

A.2 Opdrachten sturen

De toelevering van jam aan de winkels is uitbesteed aan een expediteur. Het afhalen van aardbeien en suiker enz. wordt door de fabriek zelf gedaan. Het is dus een af- en aanrijden van vrachtwagens rond de fabriek.

Voor alles wat wordt gebracht en gehaald is er een laadplatform. Een vrachtwagen die aardbeienjam komt halen rijdt naar het aardbeienjam-laadplatform en haalt daar aardbeienjam af. Staan er al andere vrachtwagens te wachten, dan gaan die voor, dus sluit de vrachtwagen achter in de rij voor dat laadplatform aan.

Die rij is soms nogal lang, want onze fabriek loopt nog niet zo gesmeerd. Pas wanneer een vrachtwagen staat te toeteren, wordt er met de jamproductie begonnen. Wanneer er nog geen ingrediënten zijn, dan moeten er zelfs nog vrachtwagens weg om suiker en aardbeien op te gaan halen.

Het is natuurlijk jammer als de chauffeur van de vrachtwagen die jam kwam halen al die tijd staat te nietsen. Ook dat is immers verspilde werkkracht. Gelukkig is dat handig op te lossen: de chauffeur kan meehelpen met de jambereiding, of hij kan even iets anders gaan doen waarbij hij nuttig is. Zo blijkt opeens dat onze fabriek misschien toch best wel gesmeerd loopt, als niemand te beroerd is om de handen uit de mouwen te steken!

De expediteurs willen natuurlijk niet alleen aardbeienjam vervoeren. Ze zullen dus ook naar andere fabrieken rijden. Mijn opdracht werkt dan ook met programma's die zijn opgebouwd uit heel veel van zulke fabriekjes, allemaal netzo werkend als de hier beschreven aardbeienjamfabriek. Die fabriekjes heten dan opeens *objecten*, en de manier waarop de fabrieken werken heet *object-georiënteerd programmeren*. Dat is een manier van programmeren die tamelijk nieuw is, maar die snel aan populariteit wint doordat het heel veel lijkt op de manier waarop we in het dagelijkse leven dingen meemaken, dit in tegenstelling met de oudere soorten van programmeren.

A.3 Vertalen

Heel algemeen gesteld moet ik programma's vertalen. Dit lijkt wel iets op het vertalen van gewone spreektaal. We kunnen spreektalen in elkaar vertalen doordat ze uiteindelijk allemaal hetzelfde kunnen uitdrukken. Dus, ook al zeg je het anders, alle talen staan je toe de zelfde betekenis in een boodschap te leggen.

In mijn opdracht heb ik een manier ontwikkeld om te kijken naar de betekenis van een taal in plaats van naar de manier waarop je het zegt. Dat maakt het mogelijk om in principe elke willekeurige taal tegen mijn systeem te praten.

Wanneer je conciërge bent in een hotel, dan kun je door je gasten in allerlei talen worden aangesproken. Het is niet nodig dat je in elke taal wordt uitgelegd hoe een nieuwe gast kan inchecken in je hotel. Het is voldoende als je dat in één taal wordt uitgelegd. Als een gast het in een andere taal vraagt, kun je hem toch inchecken, doordat je de betekenis uit zijn verzoek haalt, en bij die beteknis weet je hoe te handelen. Datzelfde heb ik in mijn systeem gestopt: ik leer het van alles, maar in een bepaalde taal, en het systeem onthoudt de betekenis van wat ik hem heb geleerd, niet de manier waarop ik het uitgelegd heb.

Gelukkig zijn computertalen veel duidelijker dan spreektalen, vooral doordat je in computertaal veel minder subtiliteiten kunt uitdrukken, en omdat dubbelzinnigheid uit den boze is. Wat wel erg lastig is, is dat de talen enorm verschillen. Het is niet zo dat ik naar het Spaans, Italiaans en Portugees probeer te vertalen (dat zijn drie talen die heel erg op elkaar lijken qua zinsbouw en woordgebruik), maar ik betrek er ook Nederlands, Afrikaans en Japans bij, als het ware. Het zal duidelijk zijn dat ik het vertalen heel algemeen moet aanpakken. Vandaar dat ik naar de betekenis kijk in plaats van naar de manier van opschrijven. Dat heeft als extra voordeel dat ik straks ook met een geringe inspanning de taal van de Marsmannetjes aankan als die ons komen bezoeken...

A.4 Orthogonaliteit

Het woord orthogonaliteit komt heel vaak in mijn verslag voor. Het is een wat preciezer woord voor 'onafhankelijk'. Een voorbeeldje kan verduidelijken wat ik hiermee bedoel. *orthogonaliteit*

Orthogonaliteit kun je heel mooi zien aan de mengkranen in de douche. Met die kranen kun je de temperatuur en de straalkracht instellen, de twee zaken die het douchegenot bepalen. Als je er een tijdje onder staat wil je misschien de straal wat krachtiger hebben. Dat lukt alleen door aan *beide* kranen te draaien, want anders is de temperatuur niet goed meer.

Dit voorbeeld laat zien dat de kranen in zo'n mengkraan niet onafhankelijk invloed hebben op temperatuur en straalkracht, m.a.w. de kranen zijn niet orthogonaal. Alles wat je wenst is mogelijk, maar je moet daarvoor wel een *combinatie* van kranen verdraaien.

In de douche is dit keurig opgelost door een nieuw type mengkraan, namelijk die waarbij een handle wordt gebruikt voor het instellen van beide zaken. Draai je de handle vlak in het rond, dan stel je de temperatuur in. Til je de handle omhoog, dan regel je de straalkracht. Hiermee is het dus mogelijk om met één beweging de straalkracht te veranderen, waarbij de temperatuur van het water gelijk blijft. Zo'n kraan is dus wèl orthogonaal, want de twee instellingen zijn onafhankelijk van elkaar mogelijk.

Nu is dit alles in de douche nog wel te overzien, maar in een ingewikkelde fabriek met veel instelknoppen in de regelkamer wordt het al heel anders. Daar is orthogonaliteit van de instellingen een noodzaak, want als het bijregelen van één knop betekent dat dertig andere knoppen ook weer moeten worden bijgeregeld, enzovoort, dan is het einde zoek. Zo is het ook met mijn afstudeeropdracht. Ik probeer als het ware een programma bij te regelen (in het vertaalproces) totdat het geschikt is voor een bepaald type computer. Om te voorkomen dat alle instelknoppen elkaar beïnvloeden is ook voor mijn opdracht orthogonaliteit een heel handige manier om me veel instelwerk te besparen.

Een leuk aspect aan orthogonaliteit is nog, dat alles mogelijk is. Als de mengkraan 100 temperaturen kan instellen, en 50 straalkrachten, dan zijn alle $100 \times 50 = 5000$ standen ook mogelijk, en ze zijn ook allemaal anders. Dat is bij niet-orthogonale dingen altijd maar de vraag.

A.5 Raamwerken

In mijn opdracht werk ik mijn oplossingen niet helemaal uit, maar ik bied een raamwerk. Dat is een beschrijving van de elementen die ik nodig heb om een oplossing samen te kunnen stellen. In het Engels heet dat een framework, wat zich vrij laat vertalen naar raamwerk: het geeft, net als een kozijn, de structuur aan waarbinnen de oplossing, het glas, doorgaans past, maar dat moet er nog wel binnen worden aangebracht. Naar wens kan dat matglas zijn, of groen glas, of gewoon glas. *raamwerk*

Ik maak zo'n structurele oplossing als volgt: ik splits het probleemgebied op in die delen die volgens mij van belang zijn, en die beschrijf ik zo nauwkeurig, dat ik bij een specifiek probleem hopelijk kan volstaan met het samennemen van standaardblokjes, eventueel nog hier en daar kleinigheden uitwerkend.

Dit lijkt wel wat op een administratie van een groot kantoor waarvandaan veel brieven moeten worden verstuurd. Dit probeert men zo veel mogelijk te doen met standaardbrieven, zodat voor een gegeven brief alleen een aantal velletjes, folders en kaartjes hoeft te worden bijeengeraapt, afhankelijk van wat we de ontvanger willen mededelen. Voor elk geval zijn brieven gemaakt, en in heel bijzondere gevallen schrijven we er met de hand even wat bij. Zo wordt het dus mogelijk, dankzij grondig denkwerk vooraf, snel een antwoord te zenden op elke brief, zelfs wanneer we met een afwijkend geval te maken hebben.

A.6 Mijn oplossing

De oplossing die ik voor het vertalen van elke computertaal naar iedere willekeurige computer aandraag, bestaat uit drie stappen. Die stappen worden achter elkaar uitgevoerd door de vertaler. Ik zal nu de drie stappen beschrijven, gevolgd door een beschrijving van een uitbreiding buiten de vertaler om.

A.6.1 Zoek de verschillen

De eerste stap in het vertaalproces is eigenlijk een zoekplaatje: 'zoek de tien verschillen'. Ik kijk naar de verwachtingen die de computertaal heeft van een computer, dus wat de computer allemaal moet kunnen volgens die taal, en dan kijk ik wat de verschillen zijn met de computer waar ik het computerprogramma op wil draaien. Als iets ontbreekt, dan moet ik het zelf maken met wat ik wel heb.

Dit lijkt op het bereiden van een maaltijd waarvoor niet alle ingrediënten in de keuken aanwezig zijn. Is er geen pannekoekmeel aanwezig, dan moet je het zelf maken met behulp van bloem, boekweitmeel en een snufje zout. Als dat eenmaal gemengd is, kun je verder gaan alsof je met pannekoekmeel aan het werk was: eitje erbij, melk, glad roeren en bakken.

Dit moet ook gebeuren bij het vertalen van programma's voor bepaalde computers: de taal die de computer 'spreekt' kan uitdrukkingskracht missen, en dan moet je zelf die uitdrukkingen bedenken en toevoegen. Bedenk daarbij dat alle talen alles kunnen uitdrukken, alleen dat het er soms wat anders uitziet. Uitdrukkingen zijn manieren om in weinig woorden veel te zeggen, maar die zijn niet letterlijk vertaalbaar en moeten dus wat omstandiger worden vertaald. Wat op welke manier vertaald moet worden, verschilt per computer, net als de voorraadkast die per huishouden andere zaken bevat, waardoor het bakken van de pannakoeken iedere keer anders gaat.

Het zoeken van verschillen is dus de stap die constateert dat er geen pannekoekmeel in huis is. Zo komt er een lijstje met op te lossen zaken naar voren, die in de volgende stappen wordt opgelost.

Het handige van mijn oplossing is de orthogonaliteit: ik heb de beschrijving van alle mogelijke machines orthogonaal gemaakt. Dit betekent dat de verschillen die ik ontdek automatisch onafhankelijk van elkaar zijn. Als ik het ene verschil oplos, dan zullen andere verschillen daar niet anders van worden, net als de straalkracht die ik in de orthogonale mengkraan kon bijregelen zonder de temperatuur te beïnvloeden.

A.6.2 Bedenk oplossingen

Er zijn verschillende manieren te bedenken om pannekoekmeel na te maken. Niet iedereen doet er bijvoorbeeld boekweitmeel in, en ook het aantal eieren dat je toevoegt verschilt nog wel eens. Zo zijn er vaak meerdere oplossingen denkbaar voor de verschillen die moeten worden weggewerkt. In mijn opdracht betekent dit dat er meerdere aanvullingen op het originele programma denkbaar zijn, waaruit ik een keus moet maken.

Sommige oplossingen voldoen gewoonweg niet. Zo kun je iemand die een natrium-arm dieet volgt geen pannekoeken met zout voorzetten. Zo ook bestaan er programma-aanvullingen die te weinig doen om het verschil helemaal op te vangen.

Andere oplossingen voldoen wel, maar het is maar de vraag welke de handigste keuze vormt. Dat wordt bepaald in de volgende stap. Aan die volgende stap wordt daarvoor een lijstje doorgegeven met oplossingen, voor elk verschil dat de vorige stap vaststelde minimaal één.

Ook hier is orthogonaliteit weer handig, want dat zorgt dat ik de keuze van een oplossing voor een probleem vrij mag kiezen. Als er voor een verschil meerdere oplossingen bestaan, dan kan ik daaruit kiezen zonder dat dit de keuze van een andere oplossing beïnvloedt.

Helaas echter, is niet iedere combinatie even zuinig met de rekenkracht van de computer. En net als de werkgever die de mensen aan het werk wil houden, willen wij ook voorkomen dat de computer naar stoplichten gaat zitten staren.

A.6.3 Geen cent te veel hoor!

Om een zo zuinig mogelijk programma te maken, is het nodig om enige afwegingen te maken. Hierbij wordt dan uitgegaan van de in de vorige stap gegenereerde groepen oplossingen, waarvan ingeschat wordt hoe duur ze zijn.

Dit inschatten is moeilijk, omdat het erg ingewikkeld is, er speelt heel veel mee. We moeten ook hier een manier bedenken om het hanteerbaar te maken. Dit is in mijn opdracht de kostenmeting geworden. Een kostenmeting probeert, door te kijken naar onderdelen van de computer te kijken, te meten wat nu eigenlijk zorgt dat een oplossing duur wordt. Als bijvoorbeeld twee computers samenwerken moeten, maar één van de twee is constant aan het werk terwijl de andere er de kantjes afloopt, dan is dat duur. Daarom heb ik kostenmetingen voor 'niksdoen' en 'omkomen in het werk' geïntroduceerd.

Op basis van zulke kostenmetingen kan ik, na enig gereken, inschatten hoe duur een bepaalde oplossing is. Zo kan ik allerlei combinaties van oplossingen uit de groepen van de vorige stap uitproberen, tot ik de zuinigste heb gevonden. Dat is dan de oplossing die ik op de computer zal laten lopen.

Gelukkig voor mij, zijn de kostenmetingen ook weer orthogonaal te maken. Dit heeft tot gevolg dat het bijstellen van één knop een heel directe invloed heeft op de kosten van een totaaloplossing. Ik kan hierdoor zelfs terugredeneren, dus ik kan besluiten dat ik kosten wil drukken, en dan uitrekenen wat ik daarvoor zou kunnen doen, welke kostenmetingen daarvoor omlaag moeten. Net als een huishoudboekje dat met verlies draait: je kunt dan aangeven dat er bijvoorbeeld zuiniger met gas moet worden omgespringen in plaats van willekeurig maar wat veranderingen uitproberen.

A.6.4 Koffiedikkijken

Het voorspellen van wat het zuinigst is tijdens het vertalen is soms een beetje koffiedikkijken. Om te voorkomen dat het daardoor helemaal fout gaat, is het verstandig om een vinger aan de pols te houden als het programma eenmaal op de computer loopt. Als het dan uit de hand loopt, bijvoorbeeld doordat één computer veel te veel werk heeft en een ander niet, dan kun je wat werk doorschuiven. Die samenwerking zorgt ervoor dat het werk sneller geklaard is.

Ik doe dit door, net als bij het vertalen, een paar kostenmetingen te doen voor de meest relevante zaken. Tijdens het vertalen was het schatten geblazen, maar nu komt er meer zekerheid. Net als een telefoontikker: vantevoren weet je ongeveer hoe duur een gesprek zal worden, maar pas als je aan het bellen bent wordt het steeds zekerder.

Het betekent dus dat we niet alles vantevoren met zekerheid weten, maar dat komt doordat alles zo ingewikkeld met elkaar samenwerkt, dat we niet alles vooraf kúnnen voorzien. We nemen daarom genoegen met een zo goed mogelijke schatting.