

# De complexiteit van eenvoudige computersystemen

dr ir M. van Steen

Rede uitgesproken bij aanvaarding van het ambt van hoogleraar grootschalige gedistribueerde systemen, aan de faculteit der Exacte Wetenschappen, afdeling Informatica aan de Vrije Universiteit op 10 januari 2003.



# Inhoudsopgave

<b>1</b>	<b>Complexiteit in computersystemen</b>	<b>6</b>
1.1	Het perspectief van een gebruiker . . . . .	6
1.2	Het perspectief van een programmeur . . . . .	8
<b>2</b>	<b>Het beheersen van complexiteit</b>	<b>11</b>
2.1	Het traceren van mobiele objecten . . . . .	11
2.2	Onder de moterkap . . . . .	14
<b>3</b>	<b>Het beheersen van eenvoud</b>	<b>17</b>
3.1	Epidemische computersystemen . . . . .	17
3.2	Kleine-wereld computersystemen . . . . .	21
<b>4</b>	<b>Conclusies</b>	<b>22</b>
<b>5</b>	<b>Tot slot</b>	<b>24</b>
<b>6</b>	<b>Dankwoord</b>	<b>25</b>



Mijnheer de rector magnificus  
Dames en heren,

We kennen het verschijnsel maar al te goed: de diepe zucht die ontstaat bij het gebruik van een of ander apparaat en die langzaam maar zeker overgaat in ergernis. Laat ik een aantal voorbeelden noemen:

- De beruchte videorecorder die om volstrekt onbegrijpelijke redenen dat ene programma *niet* heeft opgenomen terwijl u toch nadrukkelijk die tiental instructies opgevolgd hebt die het gebruiksvriendelijke *showview* met zich meebrengt.
- Het handige mobiele telefoontje dat u ten tweede male doet schrikken met de mededeling dat u vlak daarvoor een telefoontje gemist heeft. Het vervolgens wissen van deze mededeling heeft althans mij enige kopzorgen gevergd, laat staan het automatisch terugbellen (wat ik overigens nog steeds niet beheers).
- De electronische tijdschakelaar die, als u er tijdelijk geen gebruik van wil maken, het beste uitgeschakeld kan worden door de stekker van het te regelen apparaat direct in het stopcontact te steken.
- De thuiscomputer die voor de zoveelste keer een blauw scherm vertoont en waarbij de enige oplossing voor de doorstart de *reset* knop is, die voor uw gemak sinds een aantal jaren aan de vóórzijde van het apparaat geplaatst is. Overigens krijgt u daarna wel bij het opstarten de mededeling dat u toch echt de volgende keer fatsoenlijk moet afsluiten om te voorkomen dat er een tijdrovende schijfcontrole plaatsvindt.

En daarmee ben ik beland bij computers, mijn vakgebied.

Mijn ergernis over de ogenschijnlijke complexiteit van computersystemen ligt diep. Ik zal het maar bekennen: in veel gevallen snap ik helemaal niets van deze systemen, hoewel ik wel geacht wordt expert te zijn. Het aantal uren dat ik besteed heb om een computersysteem weer in de lucht te krijgen is ontelbaar. Mijn familie, en vooral ook mijn schoonfamilie heeft dankbaar gebruik gemaakt van mijn schijnbare expertise. Daartegenover

staat dat er ook dikwijls terecht geklaagd is over de intensiteit van reparatie-inspanningen die onverwijld tot sociale afwezigheid leidt.

Toegegeven, de voldoening is groot als na een aantal uren worstelen het uiteindelijk tegen middernacht toch lukt om de speciale geluidschip aan de praat te krijgen nadat de juiste stuurprogramma's van het Internet zijn geplukt en hun plekje hebben gekregen in het systeem. Mijn neef was blij, zijn vader eveneens, zijn moeder wat minder. En ik had mijn eer gered. Maar laten we eerlijk zijn: dergelijke praktijken zijn te gek voor woorden en ik weet zeker dat er velen onder u zijn die zich regelmatig afvragen of het niet allemaal wat eenvoudiger kan.

Het antwoord op deze laatste vraag is simpelweg "ja".

Het probleem echter is dat die eenvoud doorgaans niet vanzelf komt. Sterker, er moet behoorlijk wat moeite in het ontwerp en de realisatie van computersystemen gestoken worden om die eenvoud te krijgen. Anders gezegd:

*Het is relatief eenvoudig om een complex systeem te maken, maar het vergt een nogal complexe inspanning om tot een eenvoudig systeem te komen.*

Mijn leeropdracht betreft grootschalige gedistribueerde computersystemen. Dit zijn systemen die bestaan uit aan elkaar gekoppelde computers waarbij grootschalig zich voornamelijk vertaalt naar veel gebruikers en veel computers, en gedistribueerd betrekking heeft op het feit dat deze computers verspreid liggen over het hele Internet. Het is duidelijk dat complexiteit hier een belangrijke rol speelt. Het is wellicht minder duidelijk dat eenvoud nog belangrijker is. Zonder de *beheersing van eenvoud*, d.w.z. de kunst om eenvoud te identificeren en vast te houden, zullen we nooit de wellicht inherente complexiteit van grootschalige gedistribueerde computersystemen kunnen bevatten.

In deze rede zal ik trachten u duidelijk te maken waarom eenvoud zo'n belangrijke rol speelt, en in het bijzonder bij onderzoek naar de ontwikkeling van computersystemen. Het komen tot eenvoudige oplossingen wordt zwaar onderschat, niet in de laatste plaats omdat bij een eenvoudige oplossing de complexiteit van de weg daar naartoe alleen begrepen zal worden door experts.

# 1 Complexiteit in computersystemen

Laten we eerst eens kijken hoe complexiteit in computersystemen zich manifesteert. Om het probleemgebied af te bakenen beperk ik mij tot zogeheten besturingssystemen. Een zeer bekend besturingssysteem is bijvoorbeeld Windows, waar weer verschillende varianten van bestaan. In het geval van gedistribueerde computersystemen, kan een besturingssysteem beschouwd worden als een verzameling programma's die tezamen de onderliggende computers en de wijze waarop ze met elkaar verbonden zijn afschermen voor gebruikers. Besturingssystemen zijn belangrijk omdat ze feitelijk het "gezicht" bepalen van een computersysteem.

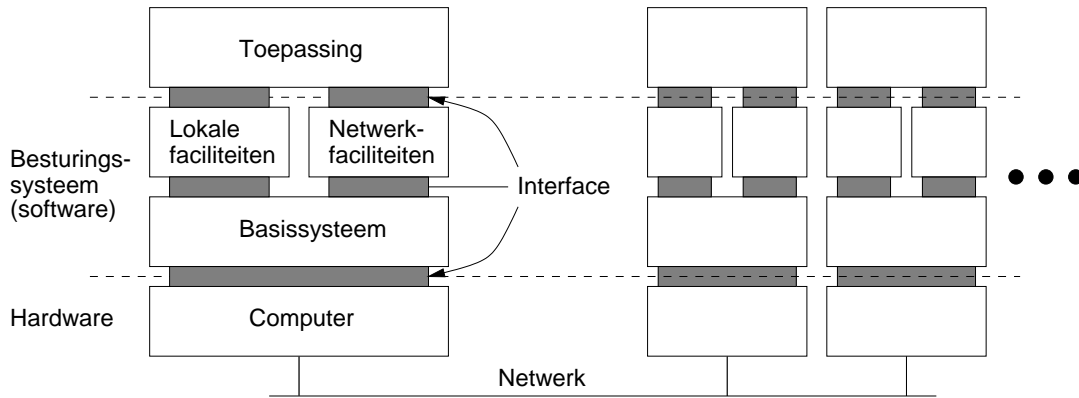
## 1.1 Het perspectief van een gebruiker

Voor de meesten van ons zal de complexiteit van een computersysteem zich in eerste instantie manifesteren aan de buitenkant. Daarmee doel ik op de complexiteit van het *gebruik*. Zo worden bijvoorbeeld Apple Macintosh computers door grote groepen als bijzonder gebruikersvriendelijk ervaren, waar tegenover staat dat UNIX gebaseerde systemen doorgaans als ingewikkeld bestempeld worden. De zeer veel gebruikte Windows producten van Microsoft lijken hier een middenpositie in te nemen.

Wat opvalt aan dit rijtje van meest populaire besturingssystemen, is dat Macintosh en UNIX al jarenlang veel dichterbij elkaar staan dan dat elk tot Windows staat. Sterker, met de meest recente versie van het Macintosh besturingssysteem, OS X, wordt zowel de traditionele Macintosh gebruiker op z'n wenken bediend, maar ook de traditionele UNIX aanhanger. Mac OS X verenigt simpelweg beide werelden.

Deze vereniging is op zich niet verwonderlijk. Wat ten grondslag ligt aan beide systemen is een relatief eenvoudig model van het onderliggende computersysteem. Beide besturingssystemen proberen de onderliggende hardware in slechts een paar concepten te vangen. Ter illustratie: in UNIX speelt bijvoorbeeld het begrip "bestand" een cruciale rol. Geheugen is niets anders dan een bestand, een harde schijf is een bestand, een netwerkverbinding gedraagt zich als een bestand, en zo voort.





Figuur 2: De globale organisatie van een gedistribueerd computersysteem.

een gebruiker gemakkelijk handelingen verrichten door het gebruik van een muis; bij een commandoregel verloopt alles in principe door het intypen van commando's.

Hoewel het een makkelijker te leren is dan het andere, is het mijn stellige overtuiging dat de uiteindelijke complexiteit zoals een gebruiker die ervaart, veruit bepaald wordt door de eenvoud en de consistentie van het model dat voorgeschoteld wordt en niet door de pracht en praal dat zichtbaar is op het beeldscherm. In dit opzicht valt er nog te veel te verbeteren bij menig besturingssysteem.

## 1.2 Het perspectief van een programmeur

Hoe manifesteert de complexiteit van een computersysteem zich aan een programmeur? Voor de begripsvorming is het belangrijk om te weten dat veel van de programmatuur die u als eindgebruiker te zien krijgt doorgaans ontwikkeld is voor een specifiek besturingssysteem. Zo zal een programma dat voor Windows ontwikkeld is niet zomaar geschikt zijn om ook onder UNIX te draaien.

Deze verschillen hebben te maken met de zogeheten *interfaces* waar een programmeur mee te maken krijgt. Indien we de organisatie van een gedistribueerd computersysteem schematisch weergeven, zoals in Figuur 2, dan

kunnen er grofweg vier logische lagen onderscheiden worden. De onderste laag wordt gevormd door de hardware: de letterlijk tastbare delen van een systeem.

De volgende laag bestaat uit een basissysteem dat de meest elementaire programmatuur bevat om de hardware aan te sturen. Stuurprogramma's vormen typisch een onderdeel van het basissysteem.

Interessanter voor de programmeur zijn die delen van het besturingssysteem die zorgen voor o.a. communicatie over een netwerk, alsmede allerlei faciliteiten om gebruik te maken van lokale functies. Tot die laatste categorie behoren functies voor het bewerken van bestanden en het manipuleren van grafische objecten op het beeldscherm.

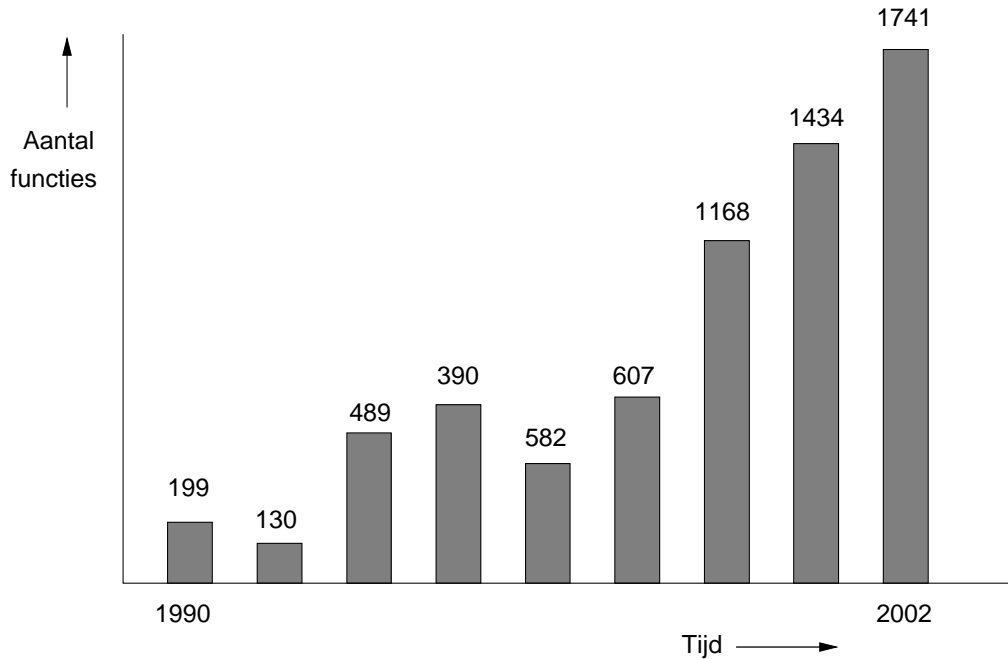
De lagen worden van elkaar onderscheiden door interfaces. Een interface is niets anders dan een verzameling functies die aangeboden worden door de direct onder die interface gelegen laag. Er zijn geen andere functies uit die laag beschikbaar. De complexiteit van een interface bepaalt dus feitelijk de complexiteit die een programmeur te zien krijgt van de onderliggende laag. Door interfaces te vergelijken van verschillende computersystemen krijgen we aldus een inzicht in de relatieve complexiteit van verschillende systemen. Laten we ons beperken tot de interfaces die een programmeur nodig heeft om toepassingen te kunnen ontwikkelen, d.w.z. de hoogstgelegen interfaces uit Figuur 2.

Een aantal onder u weet dat ik een liefhebber ben van het UNIX besturingssysteem. Laat we daarom naar de interfaces van UNIX kijken. Daarbij loop ik direct tegen het probleem aan dat er nogal wat varianten van UNIX bestaan. Voor het gemak beperk ik mij tot de standaardisatie van functies die moet leiden tot wat ook *The Single Unix Specification* genoemd wordt. Figuur 3 laat het effect van deze werkzaamheden zien in de loop van de tijd.<sup>1</sup>

Het is duidelijk dat hier sprake is van een groei van het aantal functies. Het is echter twijfelachtig of deze groei ook daadwerkelijk een groei van het aantal mogelijkheden met zich meegebracht heeft. Zo is het verschil tussen de kleinste en de grootste interface ruim een factor 13. Dit staat in geen verhouding tot de groei in daadwerkelijke functionaliteit van computersys-

---

<sup>1</sup><http://www.unix-systems.org/>



Figuur 3: Het aantal functies in de interfaces van gestandaardiseerd UNIX.

temen in de afgelopen 10 jaar. Het kan niet anders zijn dan dat we hier te maken hebben met een onevenredige vergroting van complexiteit ten koste van de kracht van eenvoud en overzichtelijkheid.

Overigens wil ik nog wel kwijt dat er een familie van populaire besturings-systemen bestaat die 100 maal zoveel functies aanbiedt dan de kleinste UNIX interface. U hoort mij goed: 13.000 functies. Als u nagaat dat de meesten van ons de grootste moeite zullen hebben om zonder haperen tot 13.000 te *tellen*, hoe moet het dan wel niet zijn voor de programmeur die geacht wordt zoveel functies te kunnen *gebruiken*.

Andrew Tanenbaum, een internationaal gerenommeerde expert op het gebied van besturingssystemen en hoogleraar aan de Vrije Universiteit, heeft de effecten van een dergelijke groei van complexiteit eenvoudig verwoord in zijn eerste wet van software:

*Meer programmatuur leidt tot meer fouten*

Meer functies betekent meer programmatuur en betekent dus meer fouten [10]. Ik durf de stelling aan dat wat de programmeur tegenwoordig voorgeschoteld krijgt aan meer complexiteit, voornamelijk bestaat uit meer fouten.

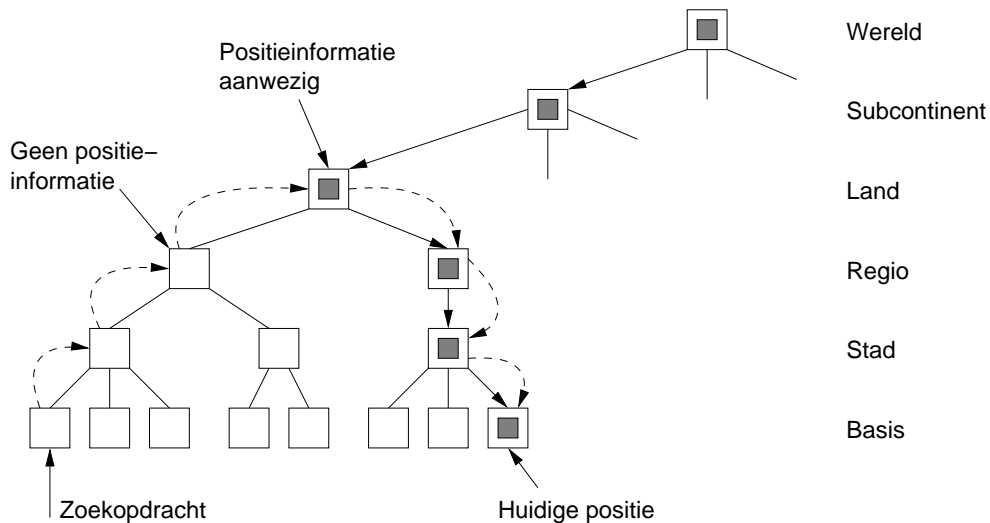
## **2 Het beheersen van complexiteit**

Willen we komen tot computersystemen die begrepen en onderhouden kunnen worden, dan speelt eenvoud een cruciale rol. Ik zie hierbij twee aspecten waar aandacht aan moet worden besteed: enerzijds het beheersen van complexiteit, en anderzijds het beheersen van eenvoud. Laat ik beginnen met de beheersing van complexiteit eens nader onder de loep te nemen door naar een systeem te kijken wat wij de afgelopen jaren ontwikkeld hebben.

### **2.1 Het traceren van mobiele objecten**

Een van de mogelijkheden die het Internet met zich mee heeft gebracht is dat het gemakkelijker wordt om overal ter wereld naar informatie te zoeken. Eén vorm waarin dit kan gebeuren is door middel van zogeheten digitale agenten [3]. Zo'n agent is feitelijk een stukje programma dat zich bijvoorbeeld verplaatst van de ene databank naar de andere om telkens relevante informatie op te halen. Een probleem dat zich met deze agenten voordoet is zij op hun reis door *cyberspace* wel weer opgespoord moeten kunnen worden.

Een andere mogelijkheid die het Internet ons biedt is het al dan niet illegaal delen van populaire bestanden. De eerste generatie programmatuur voor deze zogeheten *peer-to-peer* systemen regelde dat delen veelal op een ge-centraliseerde manier, zoals Napster. De huidige generatie volgt een volledige gedecentraliseerde aanpak waarbij een bestand in principe overal naar toe verplaatst kan worden om van daaruit weer doorgekopieerd te kunnen worden [7, 8]. Er achter komen waar nou precies een kopie van een specifiek bestand ligt is niet helemaal eenvoudig te bepalen, vooral niet als die bestanden continu verplaatst worden.



Figuur 4: Het zoeken van een mobiel object.

Wij zijn een aantal jaren geleden onderzoek gaan doen naar een oplossing voor het wereldwijd traceren van dergelijke mobiele objecten. De belangrijkste doelstelling was schaalbaarheid: het resulterend computersysteem zou miljarden objecten moeten kunnen ondersteunen die verspreid liggen over het hele Internet en die regelmatig verplaatst worden. Wat je ziet bij een dergelijk systeem is dat beheersing van complexiteit essentieel is. Laten we eens kijken naar het resultaat.

In wezen is onze oplossing zeer eenvoudig en berust op het principe dat je eerst om je heen moet kijken voordat je verder zoekt. Wat we doen is de wereld indelen in een aantal geografische basiseenheden, veelal ter grootte van bijvoorbeeld een bedrijventerrein, universiteitscampus, stadswijk, etc. Deze basiseenheden groeperen we vervolgens in grotere domeinen zoals bijvoorbeeld een stad. Steden worden weer gegroepeerd in provincies, die op hun beurt weer samengenomen worden tot een land, etc. Op deze wijze ontstaat een hiërarchie van domeinen waarbij het hoogstgelegen domein de hele wereld omvat. Dit principe is weergegeven in Figuur 4.

In elk domein plaatsen we vervolgens een computer, in ons geval een *positie server* geheten. Om een object te kunnen vinden dient het geregistreerd te zijn. Ter illustratie en om het eenvoudig te houden neem ik aan dat we personen willen opzoeken. In ons systeem gebeurt dit in principe als

volgt. Stel dat een persoon, laten we haar Elke noemen, zich verplaatst naar de campus van de Vrije Universiteit in Amsterdam. Stel dat ze daar een bezoekersbadge krijgt. Registratie verloopt als volgt:

1. Elke geeft aan de positie server op de VU haar badgenummer door.
2. De positie server op de VU geeft aan de server voor het Amsterdam domein door dat Elke zich op de VU bevindt. Let op: haar badgenummer wordt *niet* doorgegeven; alleen het feit dat ze zich op VU bevindt.
3. De Amsterdam server geeft op zijn beurt aan de Noord-Holland server door dat Elke zich in Amsterdam bevindt.
4. De Nederland server krijgt vervolgens te horen dat Elke zich in Amsterdam bevindt.
5. De server voor West Europa verneemt dat Elke in Nederland is.
6. En tenslotte krijgt de Wereld server te weten dat Elke zich in West Europa ophoudt.

Stel nu dat een ander persoon, laten we hem Max noemen, op zoek gaat naar Elke. Max is thuis, en wel in de Burgemeesterswijk in Leiden. Het zoekproces verloopt als volgt:

1. Max vraagt aan de server in de Burgemeesterswijk waar Elke is.
2. De positie server voor de Burgemeesterswijk heeft geen positie-informatie over Elke en speelt het verzoek door naar de server voor Leiden.
3. De Leiden server weet eveneens van niets en geeft het verzoek door aan de Zuid-Holland server.
4. De Zuid-Holland server heeft ook geen positie-informatie over Elke waardoor het verzoek terecht komt bij de Nederland server.

Bij de Nederland server is wel bekend waar Elke zich bevindt, namelijk in Noord-Holland. Vanaf dat moment wordt het verzoek systematisch doorgestuurd naar lager-gelegen positie servers, net zolang totdat het eindelijk

bij de server voor de VU komt. Deze laatste weet het badgenummer waaronder Elke te bereiken is. Deze informatie wordt uiteindelijk doorgestuurd aan Max die vervolgens exact weet waar zij te bereiken is.

Deze oplossing heeft een aantal prettige eigenschappen. Vanuit het oogpunt van schaalbaarheid is de belangrijkste wel dat we kunnen garanderen dat de communicatie die nodig is om iemand te traceren beperkt blijft tot het kleinste domein waarin de zoeker en de gezochte zich bevinden.

Een andere belangrijke eigenschap is dat het een simpele oplossing lijkt te zijn. Illustratief is dat een van onze collega's de oplossing zo simpel vond dat ze er van overtuigd was iets over het hoofd te hebben gezien. Een groot compliment, maar gelijk had ze wel. Wat we namelijk niet beschreven hadden is hoe het een simpele oplossing kon blijven. Het antwoord op die vraag heeft in elk geval Gerco Ballintijn, één van onze promovendi, vijf jaar onderzoek gekost. Laat ik eens een tip van de sluier oplichten.

## **2.2 Onder de moterkap**

### **Falende servers**

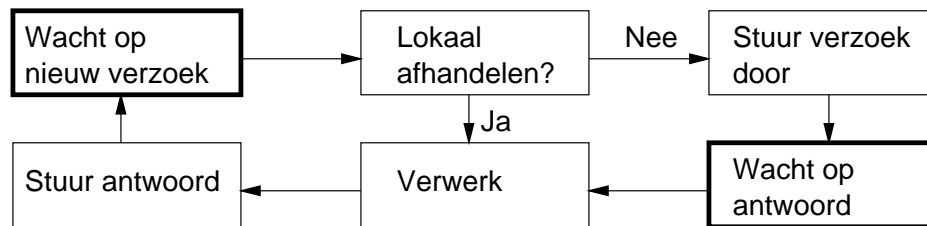
Een aspect waar we rekening mee dienen te houden is dat servers wel eens zouden kunnen falen. Zoiets merk je doorgaans als het onmogelijk wordt een verzoek door te sturen, of omdat er wel erg lang op een antwoord gewacht moet worden. Ernstiger is echter dat een gefaalde server doorgaans bij het herstel niet meer weet waar die gebleven is. Er zal dus ergens informatie vandaan gehaald moeten worden om volledig te herstellen.

Nu zouden we de zojuist beschreven oplossing voor het zoeken van geregistreerde objecten kunnen aanpassen en uitbreiden om deze gebreken aan te kunnen. Let wel: dit betekent onherroepelijk een complexere oplossing. Dat willen we niet. In plaats daarvan dient idealiter het herstellen van fouten als een apart en onafhankelijk probleem behandeld te kunnen worden. Ik heb het hier over een essentieel principe voor het beheersen van complexiteit:

*Principe: scheid zaken van elkaar*

In het Engels wordt dit principe aangeduid als *separation of concerns*.

Computersystemen zo inrichten dat zaken gescheiden kunnen worden ligt niet altijd voor de hand, maar is essentieel. In ons geval is het als volgt gelukt. Het afhandelen van een verzoek door een server ziet er in z'n meest simpele vorm uit zoals in het onderstaande stroomschema (waarbij het startpunt het vak linksboven is):



Merk op dat de vakken die vet omlijnd zijn wachtoestanden voorstellen: de server kan niets anders doen dan wachten op respectievelijk een binnenkomend verzoek of een antwoord.

We kunnen vasthouden aan deze eenvoud door in deze wachtoestanden het afhandelen en herstellen van fouten te verwerken. Het principe is wederom eenvoudig. Ten eerste richten we elke server zo in dat een eerder ingediend verzoek zonder problemen nog een keer ingediend kan worden. In het bijzonder laten we toe dat een object zijn huidige positie herhaald kan laten registreren; het eindresultaat is altijd hetzelfde als een eenmalige succesvolle registratie. Analooog geldt dat een opsporingsverzoek herhaald ingediend kan worden zonder dat dit tot administratieve fouten leidt. In het ergste geval moeten er dubbel werk verricht worden.

Ten tweede zorgen we ervoor dat als een server een verzoek doorstuurt het een kopie van het oorspronkelijk verzoek bij zich houdt. In combinatie met het toestaan van herhaalde verzoeken kunnen we nu een eenvoudige herstelstrategie introduceren. Als een server faalt en vervolgens weer opkomt, wordt gewoon gevraagd om nog uitstaande verzoeken opnieuw in te dienen. Dit principe blijkt uitstekend te werken, zelfs als meerdere servers achter elkaar falen, of zelfs weer falen terwijl ze nog niet hersteld zijn [2].

Voor alle duidelijkheid: het zojuist besproken stroomschema blijft *ongewijzigd*. Het hele mechanisme van herstel speelt zich af onder de motorkap

van de wachttoestanden. We hebben logisch gezien het herstelmechanisme volledige geïsoleerd. Daarmee is wel iets extra's geïntroduceerd, maar is niet de eenvoud van de oorspronkelijke oplossing aangetast.

## **Servers voor grote domeinen**

Met de introductie van de oplossing kregen we onmiddellijk hardnekkige kritiek te verduren: de servers voor grote domeinen zoals die voor de wereld of subcontinenten raken binnen de kortste keren overbelast. Uiteraard klopt die kritiek als er maar één server per domein is. Dat moet je dus ook niet doen. Wat je wel moet doen is weer een belangrijk principe toepassen voor het beheersen van complexiteit:

***Principe:** scheid logisch ontwerp van feitelijke realisatie*

Dit voor de handliggend principe wordt mijns inziens verrassend vaak slecht toegepast. Enerzijds zie ik dat informatici logische ontwerpen maken die door hun als feitelijke realisatie beschouwd worden. Het gevolg is dat het ontwerp onnodig complex wordt als er nieuwe eisen gesteld worden. Anderzijds zijn er hele hordes informatici die alleen maar realisaties maken, maar welk probleem ze feitelijk oplossen is niet altijd duidelijk.

Hoe zit het nou met die positie server in bijvoorbeeld ons Wereld domein. Heel simpel: daar zijn er duizenden van. Elke positie server in het Wereld domein is verantwoordelijk voor een klein gedeelte van alle mogelijke objecten. Tezamen zijn deze servers verantwoordelijk voor alle objecten. Met andere woorden, de positie servers die ik in eerste instantie beschreef zijn alleen maar *logisch*. In werkelijkheid wordt elk zo'n logische positie server gerealiseerd door mogelijksterwijs duizenden echte servers.

Pak je dit goed aan, dan kan het resultaat verbluffend zijn. Zonder in details te willen treden blijkt het mogelijk om zoveel positie servers te introduceren, en de verantwoordelijkheden zo te verdelen dat *alle* servers ongeveer even zwaar belast worden. Bovendien kan het geheel zo ingericht worden dat alle gunstige eigenschappen van het logische ontwerp, met name het lokaliteitsprincipe, volledig bewaard blijven [11].

Weer voor alle duidelijkheid: het vinden van een goede oplossing voor dit onafhankelijk deelprobleem heeft ons wel degelijk enige hoofdbrekens gekost. Eenvoud heeft een prijs, zullen we maar zeggen.

### **3 Het beheersen van eenvoud**

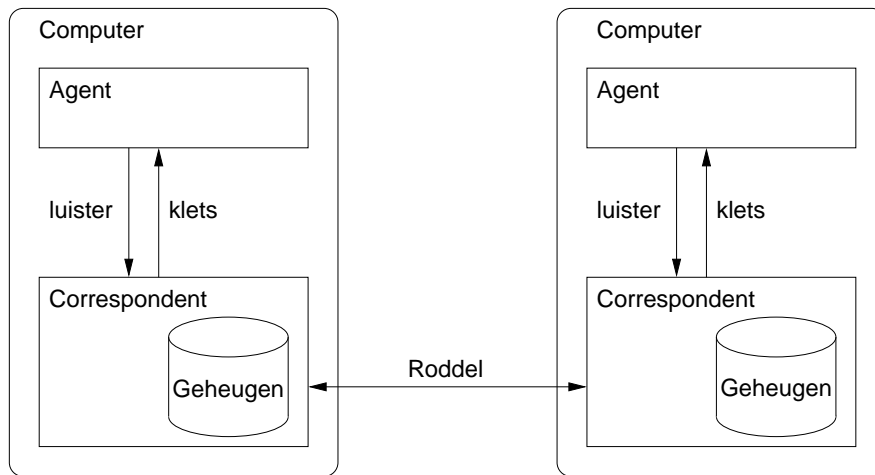
Complexiteit kan op eenvoudige wijze geïntroduceerd worden bij het ontwikkelen van gedistribueerde computersystemen. Door bijvoorbeeld alleen naar oplossingen voor falende componenten te kijken kun je een elegant en simpel ontwerp snel indrukwekkend complex maken. Het kost echter meer moeite om bij dat elegante en simpele ontwerp te blijven en zaken van elkaar te scheiden.

In plaats van alleen maar complexiteit proberen te beheersen kunnen we ook een andere insteek nemen. Laten we er eens vanuit gaan dat bepaalde uitingen van complexiteit inherent zijn aan gedistribueerde systemen. De vraag wordt dan of we de oorzaken van die complexiteit kunnen identificeren om daar vervolgens onze aandacht naar uit te laten gaan. Ik noem dit het beheersen van eenvoud.

#### **3.1 Epidemische computersystemen**

Om dat beheersen van eenvoud uit te leggen heb ik een voorbeeld nodig. Ruim een halfjaar geleden kwam een collega naar mij toe met een intrigerend probleem uit de hoek van de evolutionaire algoritmes. Wat dat zijn ga ik niet uitleggen, maar ik volsta met op te merken dat deze beestjes helemaal niets met gedistribueerde computersystemen te maken hebben. Althans, zo leek het.

Het probleem van Márk Jelasity, de bewuste collega, was er een van informatieverbreiding: hoe kan ik op eenvoudige wijze een computer op het Internet vinden die een evolutionaire berekening voor mij uit kan voeren? De manier waarop hij voorstelde dat te doen vertoonde veel verwantschappen met zogeheten epidemische algoritmes. Inmiddels zijn we al een aantal maanden aan het samenwerken – op het gebied van gedistribueerde com-



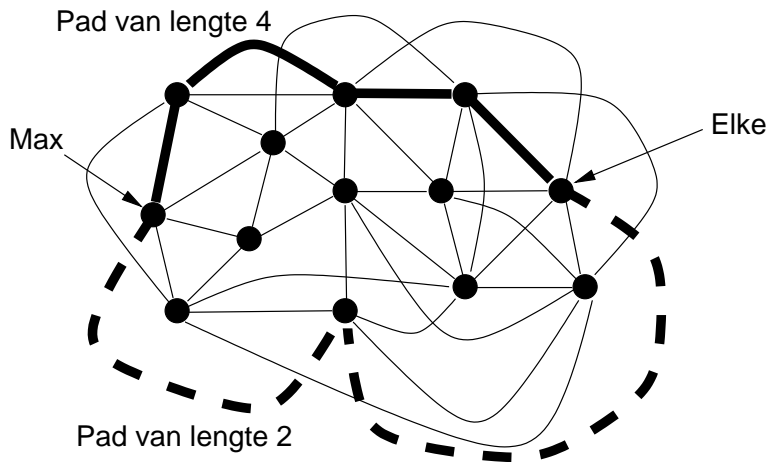
Figuur 5: Het principe van roddelende correspondenten.

putersystemen wel te verstaan. En waar het om draait is weer: vasthouden aan eenvoud.

Waar gaat het om? Eigenlijk niets meer of minder dan ordinair roddelen. Stelt u zich een flinke verzameling van computers voor. Op elke computer draait een programma dat ik voor het gemak een agent noem. Aan agenten stellen we minimale eisen: ze communiceren met een zogeheten *correspondent*, en wel door middel van twee operaties. Als de correspondent *kletst*, dan worden er  $k$  nieuwe berichten doorgegeven aan de agent. Als de correspondent *luistert*, dan moet de agent één nieuw bericht doorgegeven aan de correspondent. Elke agent heeft z'n eigen privé correspondent die altijd op dezelfde computer als die van de agent blijft.

Correspondenten wisselen berichten met elkaar uit volgens een zogeheten epidemisch protocol. In de gewone mensenwereld wordt dit protocol ook wel roddelen genoemd. Wij informatici zijn ook maar gewone mensen, dus laten we dingen maar ook gewoon bij hun naam noemen: correspondenten roddelen met elkaar. Dit principe, dat ik maar even het *roddel model* noem, is weergegeven in Figuur 5.

Hoe gaat dat roddelen in z'n werk? Elke correspondent onthoudt  $k$  berichten. Elk bericht bestaat uit drie onderdelen: het tijdstip waarop de agent het doorgegeven heeft, het adres van de agent, en het feitelijke bericht. Stel dat correspondenten Max en Elke (daar zijn ze weer), gaan roddelen. Dat



Figuur 6: Een voorbeeld van een communicatienetwerk.

doen ze door hun onthouden berichten met elkaar uit te wisselen. Nu heeft elk dus  $2 \cdot k$  berichten. De  $k$  oudste worden vervolgens weggegooid. Dit uitwisselen van berichten tussen twee correspondenten heet ook wel een roddelsessie.

Een paar observaties zijn belangrijk. Elke correspondent  $c$  heeft met  $z$ 'n  $k$  berichten in principe het adres van  $k$  andere correspondenten waarmee hij kan roddelen. We noemen deze  $k$  correspondenten ook wel de *buren*  $B(c)$  van  $c$ . Als er dan even niet geroddeld wordt, dan hebben we een *communicatienetwerk* van correspondenten, waarbij elke correspondent  $c$  in directe verbinding staat met zijn  $k$  buren  $B(c)$ . Een voorbeeld van zo'n netwerk is te zien in Figuur 6.

In Figuur 6 staan ook twee *paden* aangegeven tussen correspondenten Max en Elke. Het ene pad heeft een lengte 4; het andere een lengte 2. Het begrip pad en vooral padlengte gaat zo dadelijk een belangrijke rol spelen. Overigens heb ik opzettelijk het een en ander uit dit communicatienetwerk weggelaten. Tenslotte moeten we het niet complexer maken dan het al is.

Laten we nu eens aannemen dat er telkens maar één roddelsessie tegelijk plaatsvindt. Met andere woorden, op elk tijdstip is er maximaal één paar van correspondenten dat berichten met elkaar uitwisselt. Na een roddelsessie zal een flink aantal berichten uit het geheugen van elk van de communicerende correspondenten vervangen zijn. Omdat elk bericht de oor-

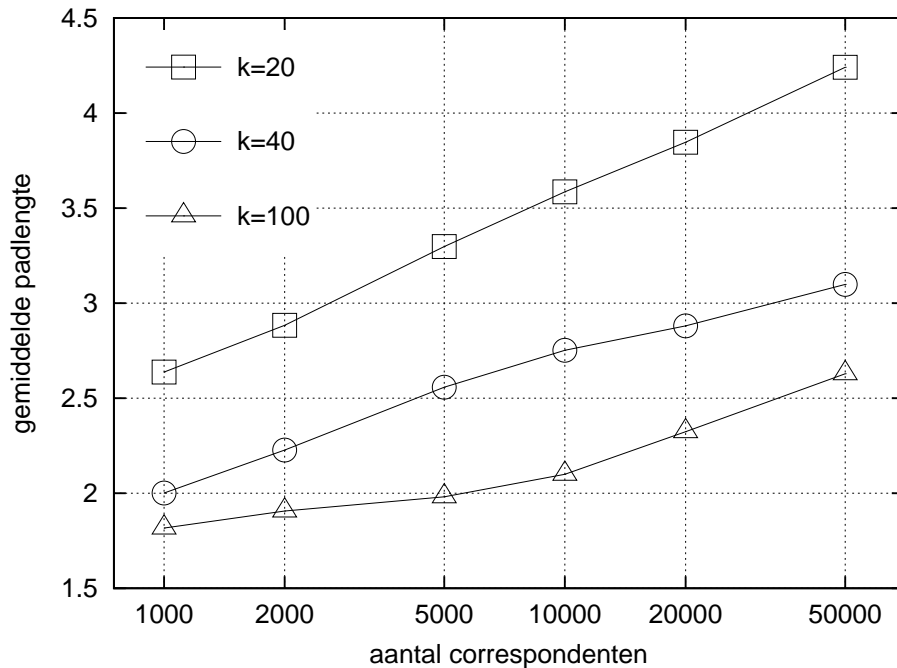
spronkelijk zender identificeert, dat wil zeggen de agent die het bericht in eerste instantie als nieuwtje aan zijn correspondent meegaf, verandert met een uitwisseling ook de verzameling burens voor elke communicerende correspondent. Een correspondent krijgt dus telkens nieuwe burens, en vergeet compleet z'n oude. Maar dat betekent dus dat met elke roddelsessie het communicatienetwerk verandert. Laten we nu eens kijken over welke veranderingen we praten.

Stel dat ons startpunt een willekeurig communicatienetwerk is, aangeduid als  $G(0)$ . We kiezen vervolgens een willekeurige correspondent en laten die een roddelsessie starten. Dit leidt tot een volgend netwerk, aangeduid als  $G(1)$ . We kiezen vervolgens weer een willekeurige correspondent voor de volgende roddelsessie, en zo voort. Het moge duidelijk zijn dat we op deze manier een *serie* van netwerken  $G(0), G(1), G(2), \dots$  krijgen.

Wat blijkt? Indien het aantal berichten dat een correspondent onthoudt voldoende groot is, dan is elk paar van correspondenten altijd met elkaar verbonden door een pad. Dat betekent dat als Max een bericht wil sturen naar Elke, dat dit eigenlijk altijd kan, desnoods via een aantal andere correspondenten. Het echt opmerkelijke is dat het aantal berichten dat elke correspondent moet onthouden om een netwerk met maar liefst 50.000 correspondenten met elkaar verbonden te laten zijn slechts ongeveer 20 is. Dat is wel erg weinig. We vermoeden zelfs dat voor een netwerk met miljoenen correspondenten, dit getal niet veel groter dan 40 hoeft te zijn.

Hoe ver zit Max van Elke verwijderd? Laten we eens kijken naar de gemiddelde padlengte tussen Max en elke andere correspondent in het netwerk. Figuur 7 laat het resultaat van onze simulatie-experimenten zien. Voor een netwerk met 50.000 correspondenten en 20 onthouden berichten per correspondent is de gemiddelde padlengte ongeveer 4. Indien elke correspondent 100 berichten onthoudt, komen we niet verder dan 2,5.

Merk op dat de grootte van het netwerk logaritmisch weergegeven is. Dat betekent dat met wat ruw schatwerk, een netwerk met zes miljard correspondenten en 100 onthouden berichten per correspondent, de gemiddelde padlengte ongeveer 5,5 is. Denkt u zich eens in: gemiddeld minder dan vijf correspondenten scheiden Max en Elke in een netwerk ter grootte van de wereldbevolking. Inderdaad, een opmerkelijk resultaat, dat we overigens ook in echte sociale netwerken tegenkomen.



Figuur 7: De gemiddelde padlengte vanuit één correspondent naar alle andere correspondenten.

### 3.2 Kleine-wereld computersystemen

Waar we met dit onderzoek tegenaan gelopen zijn wordt ook wel het verschijnsel van *kleine werelden* genoemd, een fenomeen dat we ook in ons dagelijks leven tegenkomen [1, 12]. Een uitspraak als “Je raadt nooit wie ik tegen ben gekomen”, is typisch voor een kleine wereld. Het is een prettige wereld: ondanks de enorme grootte en complexiteit kun je nog steeds het gevoel hebben in een overzichtelijk gedeelte daarvan te leven, en in de wetenschap dat je tegelijkertijd niet geïsoleerd bent. Het zojuist beschreven gedistribueerd computersysteem voldoet in meerdere opzichten aan het model van een kleine wereld [5].

Ons kleine-wereld computersysteem is vooral belangrijk omdat er we met zeer geringe inspanning zeer complexe problemen mee kunnen oplossen. Eén voorbeeld is het versturen van een bericht dat bij alle agenten aan moet komen. Door hier en daar speciale agenten te maken die binnenkomende berichten gewoon weer doorsturen, kunnen we zeer hoge garanties geven

dat een bericht ook daadwerkelijk alle agenten bereikt. Traditionele technieken zijn hiervoor ongeschikt [6].

Dit gezegd hebbende lopen we direct tegen een probleem. We hebben nu dus een zeer eenvoudig gedistribueerd computersysteem dat feitelijk complexe communicatiestructuren voortbrengt. Deze structuren hebben op hun beurt de prettige eigenschappen die kleine werelden ook hebben. Hoe zorgen we er nu voor dat die complexe structuren behouden blijven? Een cruciale observatie hierbij is dat die structuren *voortgebracht* worden, en wel louter en alleen door de eenvoudige wijze waarop correspondenten roddelen. We moeten dus die eenvoud onder controle zien te houden.

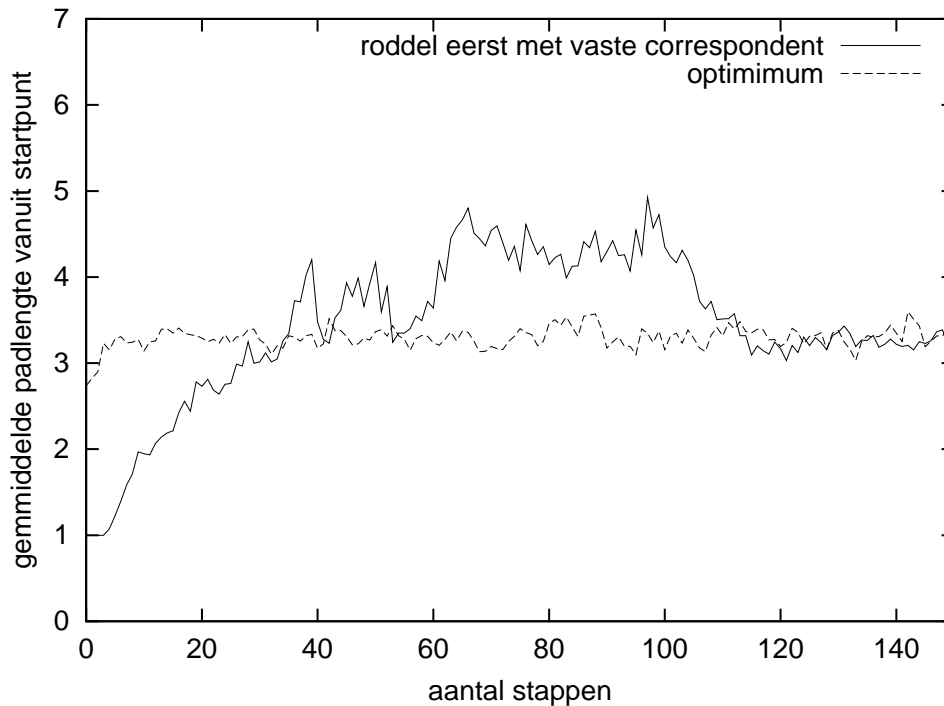
Er zijn verschillende zaken die onze eenvoud aan kunnen tasten. Eén daarvan is het toevoegen of verwijderen van correspondenten. Het blijkt dat, afhankelijk van welke communicatiestructuren je wilt hebben, je mogelijk-kerwijs een ingewikkelde procedure moet volgen om de verzameling correspondenten te wijzigen [4]. Dat soort complexiteit willen we juist *niet*.

In ons geval pakt het zeer gunstig uit. Een nieuwe correspondent kan zelfs gewoon contact zoeken met een speciaal daartoe aangewezen correspondent en vervolgens een roddelsessie starten. Binnen enkele roddelsessies blijken we weer een kleine-wereld gecreëerd te hebben. Een correspondent verwijderen is nog simpeler: we doen helemaal niets; we stoppen gewoon de roddelsessies met die correspondent.

Figuur 8 laat zien hoe snel we weer op ons gemiddelde padlengte zitten voor het geval dat elke correspondent 20 berichten onthoudt. Elke keer worden er 50 correspondenten toegevoegd aan de bestaande verzameling die elk eerst roddelen met een vaste correspondent. Nadat er 5000 correspondenten zijn (stap 100) worden geen nieuwe correspondenten toegevoegd. Na een klein aantal roddelsessies hebben de onderliggende communicatienetwerken weer de prettige eigenschappen van kleine werelden.

## 4 Conclusies

Laat ik, alvorens tot minder inhoudelijke zaken over te gaan, proberen enkele conclusies te trekken. Er valt uit het onderzoek dat ik zojuist verwoord



Figuur 8: Het effect op de padlengte bij het toevoegen van correspondenten.

heb het een en ander te leren. Een van de belangrijkste lessen is dat het flink wat moeite kan kosten om eenvoud in al haar glorie te identificeren, isoleren, en te behouden. Eenvoud is complexe materie zullen we maar zeggen. Voor het ontwikkelen van grootschalige gedistribueerde computersystemen is het echter cruciaal om met eenvoud verstandig om te gaan om niet te verzanden in nodeloze complexiteit.

Met het fenomeen van kleine werelden openbaart eenvoud zich echter ook op een andere manier: complexiteit kan ook als het ware uit het niets ontstaan. Afgezien van het fascinerende aspect hiervan, zal ook hier weer blijken dat het koesteren van de eenvoudige onderliggende principes essentieel is. Het is mij op dit moment onduidelijk hoe we met die eenvoud om moeten gaan, maar wel is duidelijk dat we vooral het generatieve proces moeten trachten te begrijpen. Zal ons dat lukken, dan komt het begrijpen van de voortgebrachte complexiteit een stap dichterbij.

Het begrip wordt in dit geval gevoed door experimenten. Alleen zo zijn we er bijvoorbeeld achter gekomen hoe ons epidemisch computersysteem zich gedraagt. We hadden het niet vantevoren kunnen bedenken. Dit voorbeeld verstevigt mijn overtuiging dat we de complexiteit van grootschalige computersystemen alleen dan kunnen begrijpen als we allerlei experimenten bedenken en uitvoeren om te kijken wat er gebeurt. Deze benadering van de informatica als een experimentele wetenschap is mijns inziens onvoldoende verankerd in de Nederlandse academische traditie. Dit is dan weliswaar geen gevolgtrekking van mijn betoog, maar het is u inmiddels wel duidelijk waar u mij kunt plaatsen ten opzichte van informatici die hun ontworpen systemen valideren door middel van het bedrijven van theorie en *daar* vervolgens conclusies uit trekken. Mijn argusogen glinsteren.

## 5 Tot slot

Ook is het duidelijk dat ik inmiddels de weg van harde wetenschap afgeslagen ben en terecht ben gekomen op het drassige terrein van opinies. Daar heb ik er ook een paar van die ik u beslist niet wil onthouden in deze openbare les.

Het lijkt bijna een gebruik om tijdens een oratie een kritische toon te laten horen ten aanzien van de rol die de landelijke politiek speelt bij de invulling van onderwijs- en onderzoeksprogramma's. Daar heb ik helemaal geen zin in, want de invloed die ik kan uitoefenen op de politieke besluitvorming met deze rede is minder dan nihil. Ik houd het dichterbij huis.

Er wordt veel geklaagd over de kwaliteit van ons universitair systeem. De studenten moeten het daarbij behoorlijk ontgelden. Ook door mij, laat daar geen misverstand over bestaan. Hoe kunnen we die kwaliteit verbeteren? We kunnen blijven afgeven op datgene wat buiten ons ligt, maar ik stel wat anders voor.

Het is wat ongebruikelijk in Nederland om op competitieve wijze met elkaar om te gaan. Feitelijk gebeurt dit voor het grootste gedeelte impliciet. Het lijkt me echter veel beter als we wat meer met open vizier gaan strijden. Het kan geen kwaad als universiteiten onderling proberen elkaar expliciet

de loef af te steken als het gaat om bijvoorbeeld het werven en goed opleiden van studenten. Kwaliteit van onderzoek en onderwijs is een belangrijk goed waar we niet alleen zuinig op moeten zijn, maar dat bovendien continu moet worden onderhouden. Soms betekent dat vertroetelen, soms snoeien.

De afdeling Informatica aan de Vrije Universiteit is mijns inziens een van de beste van het land. Zowel wat betreft onderzoek als wat betreft onderwijs. Deze mening wordt vrij breed gedragen, ook door collega's van andere universiteiten. Uiteraard is niet iedereen het hiermee eens en valt er ook veel te nuanceren. Daar heb ik echter geen zin in. Het lijkt me veel aardiger als we veel meer mensen kunnen overtuigen van de hoge kwaliteit van Informatica aan de VU zonder dat al te veel nuancering nodig is. Ik hoop daarbij overigens van harte dat afdelingen aan andere universiteiten het beter proberen te doen. Het eindresultaat kan alleen maar grotendeels positief uitpakken.

Ik nodig het bestuur van de Faculteit Exacte Wetenschappen, maar vooral ook ons College van Bestuur uit om ons te steunen in ons streven om niet alleen tot de besten te behoren, maar zelfs de beste te worden. Het kan een aangenaam wedstrijdje worden, dat verzeker ik u.

## **6 Dankwoord**

Dames en heren, ik ben aan het eind gekomen van mijn rede. Ik wil mijn dank betuigen aan het bestuur van de Vereniging van de Vrije Universiteit, het College van Bestuur, het bestuur van de Faculteit Exacte Wetenschappen, en vooral ook het bestuur van de voormalige divisie Wiskunde en Informatica voor hun inzet en steun.

Kijk ik terug op de jaren sinds mijn afstuderen, inmiddels alweer zo'n 20 jaar geleden, dan valt het me op hoe veel mensen vertrouwen in mijn kunnen hadden terwijl ik daar zelf zo over kon twijfelen.

In Leiden kreeg ik de gelegenheid om daadwerkelijk de overstap van toegepaste wiskunde naar toepassingsgerichte informatica te maken. Luuk Groenewegen, mijn dagelijks begeleider van mijn promotie-onderzoek, heeft mij niet alleen de beginselen bijgebracht van wetenschappelijk onderzoek, maar ook in de wereld van moderne klassieke muziek geïntroduceerd. In-

middels heb ik ontdekt dat er veel meer bestaat dan Alban Bergs Lyrische Suite in een uitvoering van het Schönberg Ensemble.

Henk Sips ken ik van mijn TNO periode waar hij me van meet af aan flink aan het werk heeft gehouden, en met veel vertrouwen dat het allemaal wel goed zou komen. Hij heeft daar gelijk in gekregen. Inmiddels werken we al zo lang samen dat we bij wijze van spreken een onbehaaglijk gevoel krijgen als we eventjes geen gezamenlijk project hebben. Gelukkig kunnen we de komende jaren weer vooruit op het gebied van peer-to-peer netwerken. Daarna zien wel weer verder.

Speciaal moet ik Andy Tanenbaum noemen. Zijn mensenkennis acht ik omstreden. Zijn grenzeloos vertrouwen in mij vanaf dag één ook. Zijn kunnen om mij te coachen tot wat ik vandaag ben staat echter buiten kijf. Maar hoe kun je een goede leermeester zijn zonder mensenkennis? Iets klopt er niet. Het zijn de speciale kwaliteiten van Andy Tanenbaum die me wellicht nooit helemaal duidelijk zullen worden, maar dat geeft niet. Wat wel voor me duidelijk is, is dat ik zonder Andy hier zeer waarschijnlijk nu niet zou staan. De combinatie van intelligentie, onafhankelijkheid, scherpzinnigheid en een buitengewoon gevoel van humor maakt het samenwerken met hem tot een waar genoegen.

Luuk, Henk, en Andy, ik ben jullie veel dank verschuldigd.

De tijd gebiedt me veel mensen over te slaan in mijn dankwoord, maar niet de noeste werkers die helpen om tot echte resultaten te komen: Edwin, Wim, Mark, Philip, Gerco, Arno, Ihor, Guido, Bogdan, Spyros, Jan Mark, Giovanni, Elth, Michal en Swami. Tenslotte zijn het de promovendi die feitelijk altijd het echte werk doen. En dat wil ik wel eens expliciet gezegd hebben. Ik dank jullie.

Maar de invloed die al deze mensen op mij hebben gehad is niets vergeleken bij die van mijn lief, mijn Mariëlle. Want als het gaat om het vraagstuk van eenvoud en essentie, heeft zij me geleerd om op de cruciale momenten mezelf de simpele vraag te stellen: “waar gaat het nou eigenlijk om?” Het antwoord, kan ik u verklappen, is dikwijls verrassend eenvoudig. Het is echter ook verrassend hoe dikwijls ik het weer vergeten ben. Gelukkig ben jij daar altijd weer om me te helpen.

Ik heb gezegd.

## Referenties

- [1] R. Albert and A.-L. Barabási. “Statistical Mechanics of Complex Networks.” *Reviews of Modern Physics*, 74(1):47–97, Jan. 2001.
- [2] G. Ballintijn, M. van Steen, and A. S. Tanenbaum. “Simple Crash Recovery in a Wide-Area Location Service.” In *Proc. 12th Int’l Conf. on Parallel and Distributed Computing Systems*, pp. 87–93, Fort Lauderdale, FL, Aug. 1999.
- [3] B. Brewington, R. Gray, K. Moizumi, D. Kotz, G. Cybenko, and D. Rus. “Mobile Agents for Distributed Information Retrieval.” In M. Klusch, (ed.), *Intelligent Information Agents*, pp. 355–395. Springer-Verlag, Berlin, 1999.
- [4] A. Ganesh, A.-M. Kermarrec, and L. Massoulié. “Peer-to-Peer Membership Management for Gossip-based Protocols.” *IEEE Transactions on Computers*, 2003. To appear.
- [5] M. Jelasity and M. van Steen. “Large-Scale Newscast Computing on the Internet.” Technical Report IR-503, Vrije Universiteit, Department of Computer Science, Oct. 2002.
- [6] B. Levine and J. Garcia-Luna-Aceves. “A Comparison of Reliable Multicast Protocols.” *ACM Multimedia Systems Journal*, 6(5):334–348, 1998.
- [7] D. S. Milojicic, V. Kalogeraki, R. Lukose, K. Nagaraja, J. Pruyne, B. Richard, S. Rollins, and Z. Xu. “Peer-to-Peer Computing.” Technical Report HPL-2002-57, HP Labs, Palo Alto, CA., Mar. 2002.
- [8] A. Oram, (ed.). *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O’Reilly & Associates, Sebastopol, CA., 2001.
- [9] R. Pike, D. Presetto, K. Thompson, and H. Trickey. “Plan 9 from Bell Labs.” In *Proc. Summer 1990 UKUUG Conference*, pp. 1–9, London, July 1990.
- [10] A. Tanenbaum. *Modern Operating Systems*. Prentice Hall, Upper Saddle River, N.J., 2nd edition, 2001. p. 865.
- [11] M. van Steen and G. Ballintijn. “Achieving Scalability in Hierarchical Location Services.” In *Proc. 26th Int’l Computer Software and Applications Conference*, pp. 899–905, Oxford, UK., Aug. 2002. IEEE.
- [12] D. J. Watts. *Small Worlds, The Dynamics of Networks between Order and Randomness*. Princeton University Press, Princeton, NJ., 1999.