SOFTWARE ENGINEERING VAN GEDISTRIBUEERDE APPLICATIES

BERT-JAN DE GIER

DOCTORAALSCRIPTIE
ERASMUS UNIVERSITEIT ROTTERDAM
FACULTEIT DER ECONOMISCHE WETENSCHAPPEN

MAART 2000

Erasmus Universiteit Rotterdam

ORTEC Consultants bv

SOFTWARE ENGINEERING VAN GEDISTRIBUEERDE APPLICATIES

DOCTORAALSCRIPTIE

Auteur: L.J. de Gier
Studie: Bestuurlijke Informatica
Instelling: Erasmus Universiteit Rotterdam
Faculteit der Economische Wetenschappen
Vakgroep Informatica

***Scriptiebegeleiders**:*

Erasmus Universiteit Rotterdam: Dr. Ir. J. van den Berg
Drs. M. Polman
ORTEC Consultants BV: Drs. L. Vermaas

VOORWOORD

De invloed van het Internet begint laat zich steeds meer gelden in het dagelijks leven en werk van een hoop mensen. Naast een uitstekende manier om de tijd te verdrijven, blijkt het Internet een gigantische bron van informatie te zijn. En meer. Het besef begint bij meer en meer mensen door te dringen dat er geld te verdienen is met het Internet.

Maar tegelijk met deze interesse daagt het besef dat de daarvoor benodigde kennis berust bij slechts een handvol specialisten. Het ontwikkelen van 'gezonde' traditionele applicaties is een specialiteit, en dit geldt al helemaal voor een nieuw vakgebied als gedistribueerde applicaties. Zijn de ervaringen die mensen hebben met traditionele applicatieontwikkeling ook van toepassing op de ontwikkeling van gedistribueerde applicaties? Bestaat er een overzicht met de aandachtspunten die van belang zijn bij de ontwikkeling van gedistribueerde applicaties?

Met deze vraag werd ik geconfronteerd in mijn aanstelling als student-assistent bij ORTEC Consultants BV. Dit bedrijf bood mij de kans om onderzoek op dit interessante vakgebied te verrichten, en tegelijkertijd de opgedane kennis in praktijk te brengen bij een aantal projecten. Dit vulde het enigszins theoretische karakter van deze scriptie op een uitstekende wijze aan.

Ik bevond mij tijdens het schrijven van deze scriptie in de luxe situatie van drie begeleiders, namelijk Jan van den Berg en Mark Polman van de Erasmus Universiteit Rotterdam, en Luc Vermaas van ORTEC Consultants BV. Elk had zijn eigen invalshoek op het onderwerp van deze scriptie, en dat heeft de totstandkoming ervan sterk bevorderd. Hun inzet en inbreng heeft een grote invloed gehad op het uiteindelijke resultaat. Jan, Mark, Luc, bedankt!

Bert-Jan de Gier,
Gouda, maart 2000

1 INLEIDING

1.1 Verleden

Halverwege de jaren ‘60 besloot de Amerikaanse overheid een computernetwerk op te zetten dat bestand zou zijn tegen een nucleaire aanval. Een eis was, dat als één communicatiepunt zou uitvallen, niet het hele netwerk stil zou komen te liggen. Daartoe werd een web gebouwd met netwerkverbindingen als webdraden en speciale computers, zogenaamde *routers*, om de kabels met elkaar te verbinden en het gegevensverkeer de goede kant op te sturen. Dit netwerk, initieel ‘ARPANET’ genaamd, groeide later uit tot het Internet.

Aanvankelijk werd het Internet alleen gebruikt door de Amerikaanse overheid en Amerikaanse onderzoeksinstellingen, zoals universiteiten, om gegevens uit te wisselen met andere instellingen. Ondanks het feit dat het netwerk bleef groeien, bleef het gebruik beperkt tot deze selecte groep. Dit kwam voornamelijk door het gebrek aan gebruiksvriendelijke software.

1.2 World Wide Web

Hier kwam verandering in met de komst van het World Wide Web. Het World Wide Web is het gedeelte van het Internet dat mensen kunnen verkennen met zogenaamde *browsers*. Een browser is een programma waarmee mensen op een voornamelijk grafische manier (*point-and-click*) van de ene Internet locatie naar de andere locatie kunnen “springen” en de gegevens op die locatie op kunnen vragen. Door op een *hyperlink*, een verwijzing naar een andere Internet locatie, te klikken, vraagt de browser de gegevens van die locatie op waarna ze getoond worden aan de gebruiker.

Met de komst van het World Wide Web werd het Internet voor de ‘massa’ toegankelijk. En daardoor werd het Internet ook bruikbaar voor bedrijven. Enerzijds kwam dit door het gemak waarmee het Internet gebruikt kan worden voor de uitvoering van interne bedrijfsprocessen. Daarnaast nam het aantal particuliere gebruikers (voornamelijk van het World Wide Web) sterk toe. Hierdoor werd het Internet ook op commercieel gebied interessant voor bedrijven.

1.3 Gedistribueerde applicaties

Het Internet ondersteunt het gebruik van gedistribueerde applicaties. Deze eigenschap is in het kader van deze scriptie heel belangrijk. Het is tevens een eigenschap die het Internet interessant maakt voor bedrijven. Een *gedistribueerde applicatie* is een applicatie die in stukken opgedeeld is, zogenaamde componenten, en waarvan de verschillende componenten op verschillende computers kunnen staan. Uitgangspunt van deze scriptie is gedistribueerde applicaties waarvan de componenten via het Internet met

elkaar kunnen communiceren. Hierdoor is het mogelijk om vanaf willekeurige locaties handelingen uit te voeren en programma's te gebruiken, die voorheen alleen te gebruiken waren door fysiek aanwezig te zijn bij de computer waar het programma op staat.

Er zijn diverse manieren om gebruik te maken van gedistribueerde applicaties. Vooral op het gebied van E-business liggen veel mogelijkheden. Zo is het mogelijk om potentiële klanten te informeren over de aanwezigheid of eigenschappen van bepaalde producten, met als doel de verkoop te stimuleren. In een verder ontwikkelde 'business-to-consumer' of 'business-to-business' applicatie kunnen die producten direct verkocht worden via het Internet. Daarnaast bestaat de mogelijkheid om bedrijfsprocessen gedistribueerd uit te voeren, bijvoorbeeld op het gebied van *supply chain management*: bedrijven werken virtueel samen in een keten ten einde grondstoffen, halffabrikaten en eindproducten op tijd op een gewenste locatie te krijgen.

Van de genoemde categorieën zijn voorbeelden te over te vinden. Een representatief voorbeeld van de eerste categorie is te vinden op de site van El Cheapo (www.elcheapo.nl). Hier wordt aan bezoekers de mogelijkheid geboden om te zoeken naar de laagste prijzen voor muziek, auto's, wijn en software. Bij het zoeken naar tweedehands auto's bijvoorbeeld kan de gebruiker een bepaald merk en type invullen, waarna verschillende websites afgezocht worden en tenslotte vermeld wordt bij welke dealer welke occasions te vinden zijn. Tevens worden gegevens als bouwjaar, kilometerstand, kleur en prijs getoond.

Het klassieke voorbeeld voor categorie twee is Amazon.com. Amazon.com is een boekhandel op Internet. Je kunt er zoeken naar boeken, reacties van anderen op boeken lezen, en natuurlijk boeken kopen. Dit gebeurt door middel van een creditcard transactie. Een creditcard transactie gebeurt door het verstrekken van gegevens over de creditcard van de koper (zoals het nummer en de expiratiedatum) aan de verkoper. De verkoper controleert dan of deze gegevens kloppen. Indien dat het geval is, wordt het bedrag afgeschreven van de rekening van de koper en kan het product naar de koper verzonden worden.

Supply chain management is van toepassing op bedrijven die elk een onderdeel van een informatie- of goederenstroom verzorgen. Het doel van supply chain management is het integreren van de bedrijfsprocessen van deze bedrijven teneinde de totale 'productieproces' efficiënter te maken. Dit gebeurt door de bedrijfsprocessen van de verschillende organisaties op elkaar af te stemmen. Een voorbeeld is de verkoop van een product waarvan de 'supply chain' bestaat uit vier onderdelen. Eén bedrijf wint en verkoopt de grondstoffen, één bedrijf maakt van die grondstoffen halffabrikaten, één bedrijf maakt van die halffabrikaten eindproducten, en de laatste stap in de supply chain wordt gevormd door de detailhandelaars die het product verkopen aan de consument. Het bereiken van een bepaalde voorraad bij de detailhandelaar kan automatisch doorgegeven worden aan diens leverancier.

Als deze leverancier niet voldoende halffabrikaten heeft, kunnen deze automatisch besteld worden bij de producent van de halffabrikaten. Indien nodig kan die dan weer grondstoffen bestellen bij de grondstofleverancier. Deze acties kunnen dankzij het wereldwijde bereik van het Internet gemakkelijker op elkaar afgestemd worden dan voorheen.

1.4 Probleemstelling

Het ontwikkelen van een complexe stand-alone applicatie is beslist geen sinecure. Echter, indien de applicatie – naast complex – gedistribueerd wordt, is er nog een aantal additionele factoren die invloed hebben op de wijze van ontwikkeling.

Allereerst is er de extra functionaliteit die door gedistribueerde applicaties geboden kan worden. Deze extra functionaliteit moet, om succesvol gebruikt te kunnen worden, op de juiste wijze worden geïmplementeerd. Het gedistribueerde karakter van dergelijke applicaties vraagt om speciale aandacht voor de structuur van de applicatie. Ook wordt de ontwikkeling beïnvloed door de specifieke eigenschappen van communicatie via het Internet (zie hoofdstuk 3). Dit blijkt onder meer uit het belang dat aan beveiliging van de gegevensstromen moet worden gehecht.

Naast deze invloeden dient er rekening gehouden te worden met de keuze voor een bepaalde ontwikkeltechniek. Een ontwikkeltechniek is een programmeertaal of een bepaalde technologie waarmee applicaties ontwikkeld kunnen worden. Voorbeelden hiervan zijn C++ en CGI (Common Gateway Interface). Er is een aantal verschillende technieken voorhanden, elk met zijn eigen sterke en zwakke punten.

In deze scriptie wordt geprobeerd een theoretische onderbouwing te geven van de eisen die aan een ontwikkeltechniek voor gedistribueerde applicaties moeten worden gesteld. Dit leidt tot de mogelijkheid om de geschiktheid van verschillende bestaande ontwikkeltechnieken te beoordelen. Op basis van de eigenschappen van de te ontwikkelen applicatie, en de invloed die uitgaat van het gedistribueerde karakter, moet de ontwikkelaar een gemotiveerde keuze kunnen maken voor een bepaalde ontwikkeltechniek. Ook zal deze scriptie aangeven hoe deze eisen elkaar, en de beslissingen met betrekking tot de ontwikkeling van een gedistribueerde applicatie, beïnvloeden.

Voor de ontwikkeling van stand-alone applicaties bestaat al een wetenschapsgebied dat zich bezighoudt met de ontwikkeling en de kwaliteit van applicaties. Deze discipline, Software Engineering genaamd, is een systematische aanpak voor de ontwikkeling van software. Dit wordt behandeld in hoofdstuk 5. In deze scriptie wordt het brandpunt van dit wetenschapsgebied verplaatst van stand-alone applicaties naar gedistribueerde applicaties. Zodoende onstaat de discipline *Software Engineering van Gedistribueerde Applicaties*, die zich richt op de ontwikkeling van

gedistribueerde software. Deze scriptie richt zich op de volgende probleemstelling:

> Wat moet worden verstaan onder Software Engineering van Gedistribueerde Applicaties? Meer specifiek gaat het om de beantwoording van de volgende vragen:
>
> - Welke criteria zijn van belang voor de beoordeling van ontwikkeltechnieken van gedistribueerde applicaties?
> - Welke procedure kan worden gevolgd om – op basis van deze criteria – een beslissing te nemen ten aanzien van de te gebruiken ontwikkeltechniek?
> - Welke methodiek kan worden afgeleid ten behoeve van de ontwikkeling van gedistribueerde applicaties?

In het vervolg van deze scriptie wordt getracht antwoorden te geven op de in de probleemstelling geformuleerde vragen. De beantwoording van deze vragen beperkt zich tot gedistribueerde applicaties die gebruik maken van het Internet. Het doel is een overzicht te geven van de problematiek van Gedistribueerde Software Engineering, en daarom is deze scriptie meer een verkenning in de breedte dan in de diepte.

Bij het uitwerken van de probleemstelling is een descriptief gedeelte en een normatief gedeelte te onderkennen. Het descriptieve gedeelte betreft de volgende begrippen:

- Gedistribueerde applicaties
- Het Internet
- Beveiliging
- Ontwikkeltechnieken

In het normatieve gedeelte worden voornamelijk uitspraken gedaan over:

- Eisen die gesteld moeten worden aan de structuur van gedistribueerde applicaties
- Eisen aan de ontwikkeling van gedistribueerde applicaties
- Criteria voor de beoordeling van mogelijke ontwikkeltechnieken

1.5 Uitgangspunt

Deze scriptie gaat uit van de volgende situatie:

De ontwikkelaar heeft een functionele specificatie van de te ontwikkelen applicatie. Deze betreft de gewenste functionaliteit. Dit wil zeggen dat voor de ontwikkelaar precies bekend is wat de wensen zijn voor wat de applicatie moet kunnen doen. Op basis van die specificatie kan de ontwikkelaar bepalen:

- Wat de aan te raden structuur van de applicatie is
- Welke beveiligingsmethode het meest geschikt is
- Welke ontwikkeltechniek gekozen moet worden voor de ontwikkeling

De in deze scriptie gebruikte begrippen worden zoveel mogelijk toegelicht op de plek waar ze voor het eerst genoemd worden. Enige kennis en affiniteit met het vakgebied is echter wel een pre.

1.6 Aanpak

Het nut, en soms zelfs de noodzaak, van het aanbieden van de mogelijkheid tot het gedistribueerd uitvoeren van processen, begint zich in de dagelijkse beleving van de auteur steeds vaker te manifesteren. Ook daar is de vraag om een gestructureerde aanpak van gedistribueerde applicatieontwikkeling steeds vaker te horen.

Bij het bepalen van functionaliteiten van gedistribueerde applicaties (hoofdstuk 2) is uitgegaan van in de literatuur erkende eigenschappen van gedistribueerde applicaties.

De beschrijving van de architecturen van gedistribueerde applicaties (hoofdstuk 3) is voornamelijk gebaseerd op wetenschappelijke publicaties over netwerken en netwerkstructuren.

Voor het bepalen van eisen aan, en methoden voor beveiliging van applicaties en gegevensstromen (hoofdstuk 4) is uitgegaan van door vele auteurs onderkende beveiligingsaspecten, aangevuld met een literatuurstudie over de huidige technieken.

De eisen ten aanzien van de ontwikkeling van gedistribueerde applicaties (hoofdstuk 5) zijn afgeleid uit algemene Software Engineering eisen en ervaringsfeiten van de auteur.

Criteria voor de ontwikkeltechnieken (hoofdstuk 6) zijn opgesteld aan de hand van de in hoofdstuk 2, 3, 4 en 5 geformuleerde eisen. De beoordeling van de technieken op basis van die criteria (hoofdstuk 7) is gebaseerd op een literatuurstudie, en gedeeltelijk op ervaring met een aantal technieken. Bij het formuleren van uitspraken, gebaseerd op deze beoordeling, dient rekening gehouden te worden met een bepaalde mate van subjectiviteit die in de beoordeling aanwezig is. De methodiek die gebruikt kan worden in het beslissingstraject bij het bouwen van een gedistribueerde applicatie (hoofdstuk 8) is gebaseerd op de analyse van een applicatie bij ORTEC Consultants BV. Vervolgens is van deze concrete applicatie geabstraheerd om een – hopelijk – algemeen geldig beeld te kunnen geven. Deze methodiek moet (buiten deze scriptie) nog verder worden getoetst.

1.7 Structuur van de scriptie

In hoofdstuk 2 wordt allereerst ingegaan op welke functionaliteiten gedistribueerde applicaties dienen te bieden. Daarna wordt in hoofdstuk 3 meer verteld over de mogelijke architecturen van gedistribueerde applicaties, en wordt een uitspraak gedaan worden over welke architectuur gekozen dient te worden. Hoofdstuk 4 gaat in op de noodzakelijke beveiligingsaspecten die aanwezig dienen te zijn in gedistribueerde applicaties, en over methoden om die aspecten te implementeren. In hoofdstuk 5 worden eisen ten aanzien van de ontwikkeling van gedistribueerde applicaties gedefinieerd.

In hoofdstuk 6 worden uit de voorwaarden, die in de hoofdstukken 2 tot en met 5 onderkend zijn, criteria voor de beoordeling van ontwikkeltechnieken afgeleid. Dit leidt tot een reeks criteria waarop in hoofdstuk 7 een selectie van veelgebruikte ontwikkeltechnieken beoordeeld wordt. Hieruit volgt een algemene aanbeveling over de te kiezen techniek voor een specifieke applicatie.

In hoofdstuk 8 wordt aan de hand van een (deels fictieve) applicatie een beslissingstraject voor de ontwikkeling beschreven, waarna hier een algemene methodiek in de vorm van een stappenplan van afgeleid wordt.

Tenslotte wordt in hoofdstuk 9 deze scriptie besloten met een aantal algemene conclusies en enkele aanbevelingen.

2 GEDISTRIBUEERDE APPLICATIES

In dit hoofdstuk wordt inleidend naar gedistribueerde applicaties gekeken. Eerst wordt behandeld wat gedistribueerde applicaties zijn en welke eigenschappen ze hebben. Daarna wordt een aantal redenen genoemd om gedistribueerde applicaties te gebruiken. Tenslotte wordt getracht een zo algemeen mogelijk beeld te geven van de structuur en functionaliteiten van gedistribueerde applicaties.

2.1 Wat zijn gedistribueerde applicaties?

Voordat we dieper in kunnen gaan op de gevolgen van gedistribueerde applicaties op het ontwikkeltraject, is het verstandig om na te gaan wat een gedistribueerde applicatie is. Allereerst proberen we een definitie af te leiden, waarna we naar eigenschappen van gedistribueerde applicaties kijken.

2.1.1 Definitie gedistribueerde applicaties

In de Nederlandse taal betekent gedistribueerd simpelweg “verspreid”. Maar wat houdt dit in voor applicaties?

In hun boek “Distributed Systems” omschrijven Coulouris en Dollimore [COUL, 1988] de uitvoering van gedistribueerde processen als volgt:

> *“Distributed processing enables […] applications to be based on several computers, with processing and other system responsibilities distributed amongst them”*.

Dit houdt in dat applicaties op meerdere computers tegelijkertijd kunnen draaien, dus dat *processing and other system responsibilities* over meerdere computers verdeeld zijn. Hierin wordt een aanwijzing gegeven over de eigenschappen van gedistribueerde applicaties. Dit impliceert namelijk dat er tussen de verschillende onderdelen van de applicatie gegevens moeten worden uitgewisseld.

Overigens wordt in deze definitie gesproken over *distributed processing* en niet over *distributed applications*. Dit geeft aan dat er onderscheid gemaakt wordt tussen een gedistribueerde applicatie en het uitvoeren van de taken van een gedistribueerde applicatie (distributed processing). In deze scriptie wordt niet het verspreiden van een applicatie over verschillende computers bedoeld, maar het tweede begrip, namelijk de gedistribueerde *uitvoering van een taak* door een applicatie. In het vervolg van deze scriptie worden de termen ‘gedistribueerde applicaties’ en ‘gedistribueerde processen’ door elkaar gebruikt.

De uitvoering van een applicatie wordt vaak een *proces* genoemd. In het geval van een gedistribueerde applicatie als geheel betreft het een proces

dat verdeeld is over meerdere computers. Een (computer)proces wordt in "Computer and Network Organization" [STEE, 1995] als volgt gedefinieerd:

> *"A process […] is a description of a series of actions or operations […]"*

Een proces is dus een reeks bewerkingen die leiden tot de uitvoering van een taak. Uitgaande van deze definitie, kunnen we de werking van de afzonderlijke onderdelen deelprocessen noemen, waarbij elk deelproces bijdraagt aan het uitvoeren van de overkoepelende taak.

Dat tussen deelprocessen van een applicatie gegevens uitgewisseld moeten worden, bakent het begrip 'gedistribueerde applicatie' niet voldoende af. Neem als voorbeeld van een gedistribueerde applicatie een financiële-rapportage applicatie. Op één machine staan tekstbestanden met daarin financiële gegevens. Op een tweede machine staat een gedeelte van de applicatie die op basis van die gegevens totalen, gemiddelden, enzovoorts uitrekent. Op een derde machine staat een ander gedeelte van die applicatie, dat de totalen en gemiddelden door middel van grafieken aan de gebruiker toont. Volgens bovenstaande omschrijving is dit een gedistribueerde applicatie. Maar is dat nog steeds het geval als alle delen van de applicatie op dezelfde machine staan?

Deze vraag kan beantwoord worden aan de hand van een uitspraak van Leslie Lamport over gedistribueerde systemen, die op het eerste gezicht weinig relevant lijkt:

> "*A distributed system is one that stops you from getting any work done when a machine you've never even heard of crashes.*"

In deze uitspraak, ongetwijfeld komisch bedoeld, zit het antwoord op de vraag verborgen. Tanenbaum maakt dit antwoord expliciet als hij het verschil aangeeft tussen een computer netwerk en een gedistribueerd systeem [TANE, 1996]:

> "*The key distinction is that in a distributed system, the existence of multiple autonomous computers is transparent (i.e. not visible) to the user.*"

Deze eigenschap van gedistribueerde systemen is even relevant voor gedistribueerde applicaties: de gebruiker start de applicatie op, en de applicatie draait, ongeacht hoe de verdeling over verschillende computers is. Deze eigenschap, bekend als transparantie, houdt in dat het voor de gebruiker niet uitmaakt waar de verschillende onderdelen van een applicatie zich bevinden. Een applicatie is dus ook gedistribueerd als de onderdelen op dezelfde machine staan; belangrijk is niet of de applicatie

over meerdere computers verdeeld is, maar of de applicatie dat *kan* zijn. Dit moet transparant zijn. Dit besef leidt dan tot de volgende definitie van gedistribueerde applicaties, een definitie waar in de rest van deze scriptie vanuit wordt gegaan:

> *Een gedistribueerde applicatie is de uitvoering van een proces dat mogelijk verspreid is over verschillende computers, en waarvan de deelprocessen met elkaar samenwerken.*

Samenwerken betekent in deze context, dat de verschillende deelprocessen van een gedistribueerde applicatie gegevens uitwisselen en elkaar op een zodanige manier ondersteunen, dat de applicatie de gewenste taak uitvoert.

2.1.2 Eigenschappen

In deze definitie zijn twee belangrijke eigenschappen van gedistribueerde applicaties bevat, namelijk de verspreiding over verschillende computers, en de samenwerking tussen de verschillende deelprocessen.

De niet-triviale eis dat bij de ontwikkeling rekening gehouden moet worden met de mogelijkheid om de applicatie in delen te splitsen, is onderdeel van de algemene Software Engineering eisen. Deze eisen worden in hoofdstuk 5 behandeld.

Een voorwaarde voor de deelprocessen om samen te kunnen werken is de mogelijkheid tot onderlinge communicatie. Deze communicatie wordt in deze scriptie beperkt tot communicatie via het Internet, dus op basis van het TCP/IP protocol. In hoofdstuk 3 wordt verder ingegaan op de mogelijkheden van netwerkcommunicatie.

2.2 Voor- en nadelen van gedistribueerde applicaties

Een stand-alone applicatie is een applicatie die slechts uit één proces bestaat. Er is een aantal redenen aan te voeren waarom een gedistribueerde applicatie te prefereren is boven een *stand-alone* applicatie. Er wordt hier niet getracht een uitputtende lijst te geven, maar eerder een opsomming van wat belangrijke redenen kunnen zijn. Eerst worden een paar argumenten vóór het gebruik van gedistribueerde applicaties gegeven, waarna we stil staan bij mogelijke argumenten tegen het gebruik.

Als eerste reden kan *intrinsieke gedistribueerdheid* genoemd worden. Voor sommige taken is het niet mogelijk een stand-alone applicatie te gebruiken. Een voorbeeld is een applicatie waarbij meerdere gebruikers vanaf elke willekeurige locatie contact moeten kunnen krijgen met een centrale database.

Daarnaast kan de mogelijkheid tot *resource sharing* een reden zijn om een gedistribueerde applicatie te gebruiken. Dit houdt in dat meerdere gebruikers de voorzieningen van één computer(systeem) gebruiken. De deelprocessen die op dat computersysteem uitgevoerd worden zullen gespecialiseerd zijn in taken die karakteristiek zijn voor het computersysteem. Een voorbeeld hiervan is het gebruiken van een deelproces op een supercomputer voor het uitvoeren van ingewikkelde berekeningen.

Een ander argument is *parallel processing*. Parallel processing betekent dat meerdere computers tegelijkertijd aan één taak werken. Hierdoor wordt de rekenkracht van die computers gebundeld en wordt de taak sneller uitgevoerd. Niet alleen de rekenkracht kan zo gebundeld worden: met behulp van *load balancing* kan de belasting van de CPU en het geheugen verdeeld worden over meerdere machines. Naast load balancing is ook *scalability* een aandachtspunt tijdens de ontwikkeling, indien gebruik wordt gemaakt van parallel processing. Scalability (schaalbaarheid) van applicaties houdt in dat een gedistribueerde applicatie voorbereid is op een van tevoren onbekend aantal simultane gebruikers. Indien nodig, kunnen bijvoorbeeld makkelijk additionele computers toegevoegd worden voor extra opslagruimte of extra rekenkracht. Om parallel processing succesvol toe te passen moeten load balancing en scalability goed geregeld zijn.

Availability (beschikbaarheid) kan ook een reden zijn. Indien het bedrijfskritische processen of gegevens betreft, kunnen de deelprocessen van een applicatie gerepliceerd zijn op meerdere machines, zodat bij het uitvallen van een computer, een andere computer die taak over kan nemen.

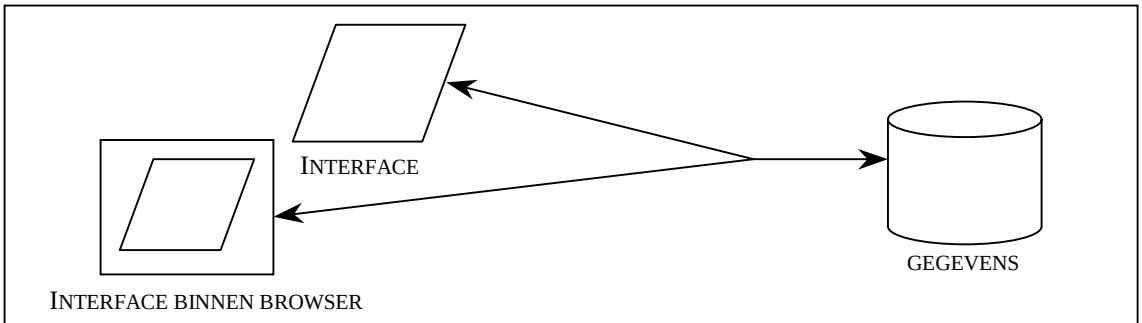
Hiertegenover staat ook een aantal nadelen. Als belangrijk nadeel kan een *grotere mate van complexiteit* genoemd worden. Deze toename van complexiteit is aan een aantal factoren toe te schrijven. De toename van de complexiteit van *communicatie* wordt veroorzaakt doordat een deelproces mogelijk moet communiceren met een deelproces op een andere computer. Daarnaast heeft de communicatie via een openbaar netwerk een aantal eigenschappen die de complexiteit op het gebied van *beveiliging* nadelig beïnvloeden (hiervoor wordt verwezen naar hoofdstuk 3). Tenslotte is de transparantie van locatie, die voor de gebruiker zo wenselijk is, de oorzaak dat het *beheer* lastiger wordt. Voor de beheerder is het, door die transparantie, moeilijker om te achterhalen waar mogelijke problemen vandaan komen. Het *debuggen* (opsporen van fouten) van gedistribueerde applicaties is een stuk lastiger dan van stand-alone applicaties.

2.3 Architecturen

In deze paragraaf wordt dieper ingegaan op de opbouw van een gedistribueerde applicatie, de architectuur. Door het noemen van een aantal

uiteenlopende verschijningsvormen van gedistribueerde applicaties, wordt getracht een beeld te krijgen van de mogelijkheden.

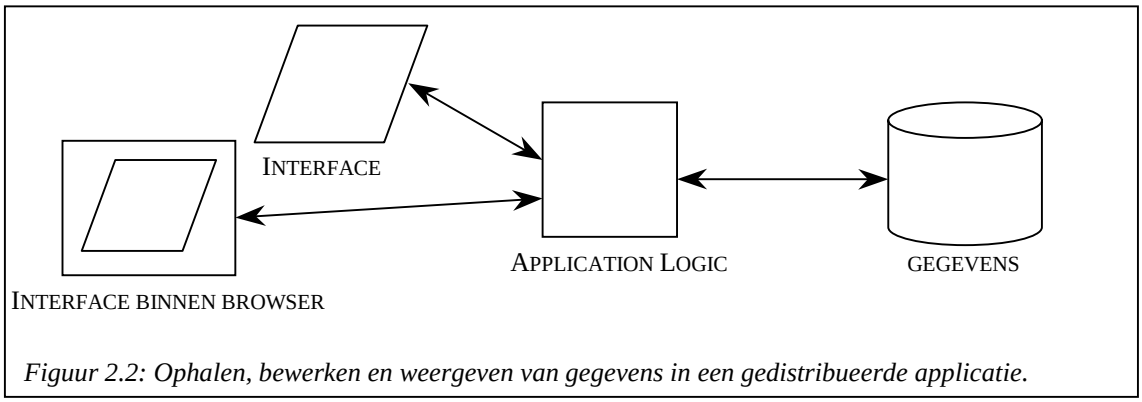
Veel gedistribueerde applicaties op het World Wide Web hebben de taak gebruikers de informatie te geven, waar ze om vragen. De applicatie bestaat doorgaans uit een webbrowser die de *user-interface* naar de gebruiker verzorgt. De user-interface is de manier waarop de gebruiker met het programma 'communiceert', bijvoorbeeld door commando's in te voeren (denk bijvoorbeeld aan de userinterface van het Windows operating system. Dit is een 'graphical user interface (GUI) om op een grafische manier ('point-and-click'; met de muis en het toetsenbord) met een programma te communiceren). Die webbrowser maakt verbinding met een webserver die op een andere machine staat, die de opgevraagde informatie aan de client verschaft. Die gegevens worden uit een database gehaald, die op weer een andere machine kan staan. De gegevens kunnen ook uit statische webpagina's bestaan (tekstbestanden die via het HTTP protocol (zie hoofdstuk 3) van de webserver naar de client worden verstuurd en in een browser ingelezen). Dit is het meer traditionele WWW gebruik. Deze taken kunnen overigens ook uitgevoerd worden door een applicatie waarvan de interface niet in een webbrowser draait. In de onderstaande figuur staan beide situaties afgebeeld. In deze figuur (en verdere figuren) worden pijlen gebruikt om berichtenoverdracht weer te geven.



Figuur 2.1: Ophalen en weergeven van gegevens in een gedistribueerde applicatie.

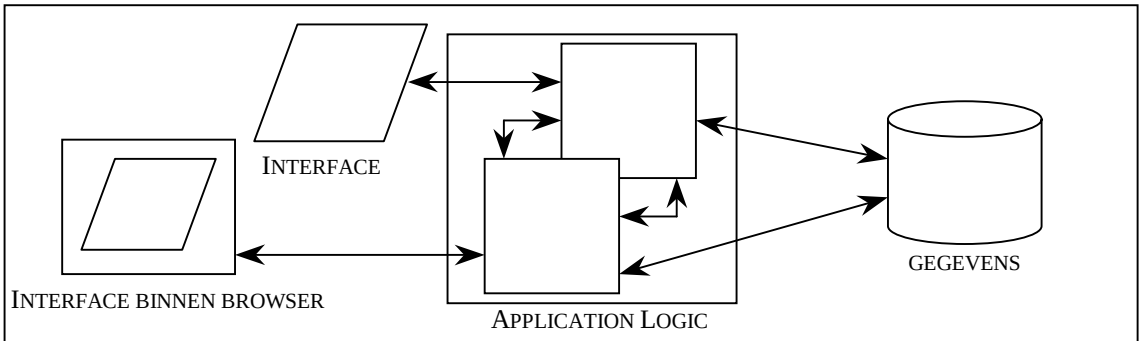
Het is ook mogelijk, dat de gegevens *bewerkt* moeten worden, voordat ze aan de gebruiker getoond worden. Hierbij kan bijvoorbeeld gedacht worden aan het uitvoeren van berekeningen, zoals het maken van een lange-termijn planning, het valideren van gegevens, of het uitrekenen van de rente voor een lening op basis van de gegevens die de gebruiker ingevoerd heeft. Meestal zijn bij dit soort berekeningen additionele gegevens nodig als invoer voor de berekening (zogenaamde *parameters*). Een voorbeeld van een parameter voor de berekening van de rente van een lening is de huidige rentestand op de geldmarkt. De waarde daarvan staat vast, en wordt bijvoorbeeld niet door de gebruiker ingevoerd, maar dient uit een database of uit tekstbestanden gehaald te worden. Het gedeelte van de applicatie dat genoemde bewerkingen uitvoert,

wordt de ‘Application Logic’ genoemd. Ook voor dit soort applicaties is het mogelijk, maar niet noodzakelijk, een browser-interface te gebruiken.



Figuur 2.2: Ophalen, bewerken en weergeven van gegevens in een gedistribueerde applicatie.

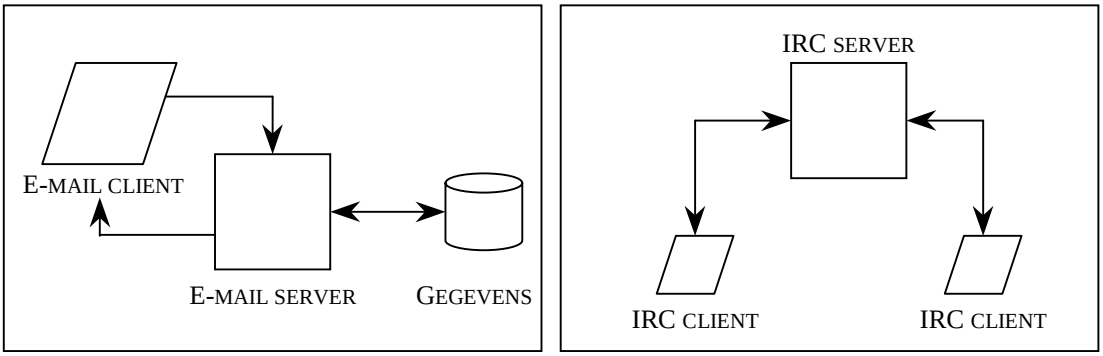
Het kan ook zijn, dat niet de representatie van gegevens het belangrijkste onderdeel van de applicatie is, maar de berekening die uitgevoerd wordt. Die berekening kan zo complex zijn, dat het verstandig is om die door meerdere computers tegelijkertijd uit te laten voeren, het eerdergenoemde *parallel processing*. Een voorbeeld hiervan is het voorspellen van het weer: ingewikkelde berekeningen in weermodellen worden door meerdere computers tegelijkertijd uitgevoerd. Dit is in de onderstaande figuur gevisualiseerd door meerdere Application Logic deelprocessen. De pijlen tussen de Application Logic deelprocessen geven aan, dat communicatie onderling mogelijk, en soms noodzakelijk is. Belangrijk hierbij is dat deze communicatie efficiënt is.



Figuur 2.3: Parallel processing: meerdere Application Logic gedeeltes.

Bij veel applicaties wordt er door de meeste gebruikers niet bij stilgestaan dat het ook gedistribueerde applicaties zijn (transparantie). Een voorbeeld is het programma waarmee iemand e-mail kan lezen. Dit programma (de e-mail client) maakt verbinding met de mailserver (het ‘elektronische postkantoor’) en haalt de mailtjes over die voor de gebruiker bestemd zijn. Ook voor het populaire Internet Relay Chat (IRC) wordt een gedistribueerde applicatie gebruikt. IRC is een systeem voor het *chatten* (kletsen) met andere mensen via het Internet. Om IRC te gebruiken moet men een programmaatje downloaden

waarmee je dan in kan typen wat je wil zeggen, en kunt zien wat anderen zeggen. Het programmaatje maakt verbinding met een speciale server die de boodschappen tussen de gebruikers heen en weer stuurt. Er is een aantal ondernemingen dat deze diensten aanbiedt.



Figuur 2.4: Structuur van e-mail en IRC applicaties.

2.4 Prestaties

De prestaties van een gedistribueerde applicatie zijn het meest tastbaar in de vorm van de responstijd van de applicatie, kortom, de snelheid waarmee de gedistribueerde applicatie zijn taak uitvoert. De snelheid is afhankelijk van drie dingen: de tijd die verstrijkt voordat de applicatie opgestart is, de snelheid waarmee de componenten met elkaar kunnen communiceren en de snelheid waarmee elke component zijn taak uit kan voeren, kortom: *laadtijd*, *communicatietijd* en *totale executietijd*.

Laadtijd is voor de ene applicatie meer van belang dan voor een andere. Voor een applicatie die HTML pagina's naar de client stuurt, komt het neer op het opstarten van een browser. Voor een applicatie die op de client uitgevoerd wordt komt het neer op de tijd die nodig is om de applicatie naar de client te versturen en de applicatie daar op te starten. Om de laadtijd zo kort mogelijk te houden, is het verstandig om de client applicatie zo klein mogelijk te houden. Dit heet een 'thin client'. Het alternatief is een applicatie die veel functionaliteit heeft aan de client-zijde, een zogenaamde 'thick client'. Deze situatie vermindert vaak de benodigde mate van communicatie, maar verhoogt de laadtijd (omdat de omvang van de client groter is).

De *communicatietijd* is grotendeels afhankelijk van de gebruikte communicatie architectuur. TCP/IP is sneller dan HTTP, en dat heeft grote invloed op de prestaties van de applicatie.

Tenslotte is de *executietijd* afhankelijk van de techniek die gebruikt wordt om de applicatie te ontwikkelen (zie ook hoofdstuk 7). Afhankelijk van de

eigenschappen van de techniek is de ene techniek beter geschikt om ingewikkelde berekeningen mee uit te voeren dan de andere. De *totale* executietijd is afhankelijk van de vraag of de deelprocessen wel of niet parallel worden uitgevoerd. Bij niet-parallele uitvoering is de totale executietijd als volgt weergegeven worden:

$$T_{executie} = \sum_i T_i$$

waarbij T_i de executietijd van component i is. Bij parallele uitvoering is deze relatie als volgt:

$$T_{executie} = \max_i(T_i)$$

De totale tijd die een applicatie nodig heeft om zijn taken uit te voeren kan (op een versimpelde manier) worden weergegeven als:

$$T_{totaal} = T_{laad} + T_{communicatie} + T_{executie}$$

Om de totale tijd terug te brengen, moeten de afzonderlijke onderdelen worden geminimaliseerd. Dit kan echter tegenstrijdige gevolgen hebben. Zo is het mogelijk dat men kiest voor een ‘thick client’, om de communicatietijd te vergroten. Dit resulteert echter in een hogere laadtijd. Bij het optimaliseren van de prestaties moet gekeken worden naar de meest beperkende factor. In veel gevallen is dit de communicatietijd. Deze is in grote mate afhankelijk van de mate van distributie van de applicatie. Hoe meer de applicatie gedistribueerd is (dus op verschillende machines staat), des te meer gegevens moeten tussen de verschillende componenten via het (relatief) trage Internet verzonden worden. Het is daarom niet onverstandig om te trachten de omvang van de gegevens die over een trage verbinding verzonden moeten worden, zo beperkt mogelijk te houden. Dit kan door componenten zoveel mogelijk op één machine te plaatsen. Dit kan tegenstrijdig zijn met de gewenste architectuur (zie hoofdstuk 3) of met de eerdergenoemde voordelen van gedistribueerde applicaties zoals *parallel processing*. Om de communicatietijd te verlagen is het verstandig om, waar mogelijk, te zorgen voor communicatiekanalen met een hoge bandbreedte.

2.5 Conclusie

In de voorgaande paragrafen zijn enkele eigenschappen van gedistribueerde applicaties en voorkomende architecturen behandeld. Op basis daarvan hebben we enkele uitspraken gedaan over de algemene *functionaliteiten* van gedistribueerde applicaties. Het begrip ‘functionaliteit’ kan afgeleid worden uit de betekenis van het begrip ‘*Requirements specification*’ (specificatie van vereisten). Deze term is afkomstig uit de Software Engineering (zie ook hoofdstuk 5). In de *requirements specification* wordt aangegeven wat de applicatie precies moet doen. Hierin staan de gewenste functionaliteiten van de applicatie beschreven. We kunnen dus zeggen dat de functionaliteit van een

applicatie aangeeft welke specifieke werking de software bezit (met het oog op het bereiken van een bepaald doel). In veel gevallen zal gegevensbewerking een (gewenste) functionaliteit zijn.

In de paragraaf over voor- en nadelen werden enkele mogelijkheden genoemd, waardoor gedistribueerde applicaties een meerwaarde kunnen hebben boven stand-alone applicaties. Deze mogelijkheden zijn:

- Resource sharing
- Parallel processing (in combinatie met load balancing en scalability)
- Availability

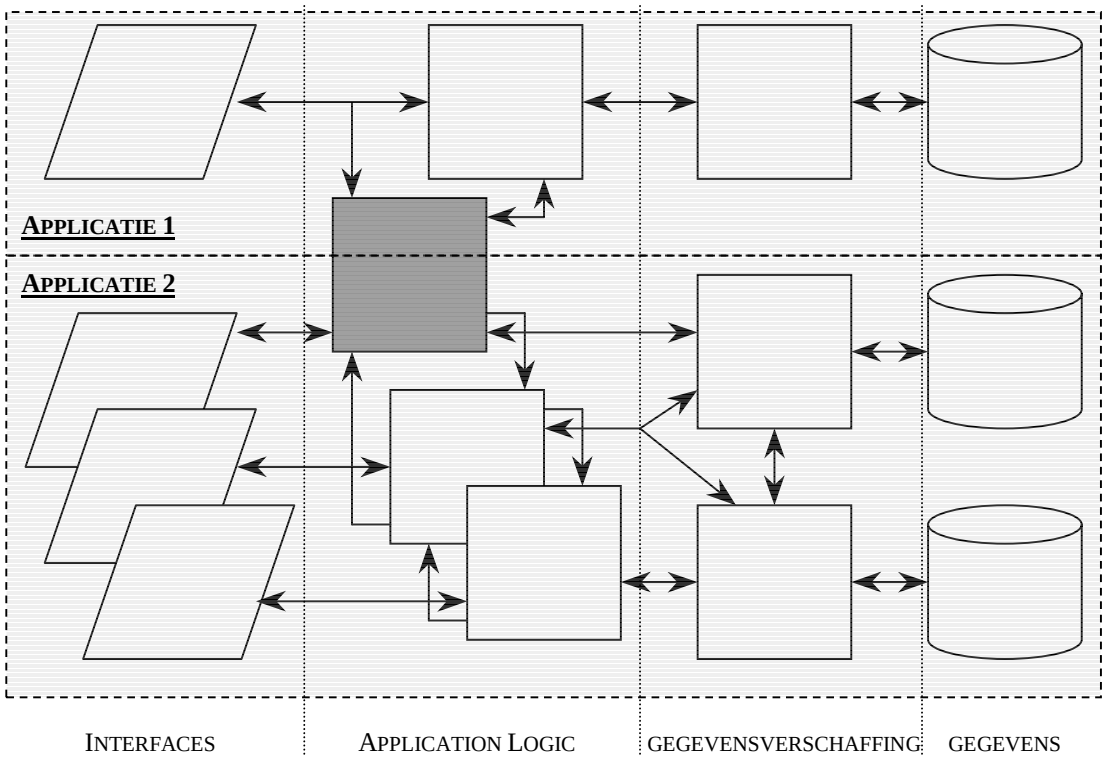
In de paragraaf over architecturen werd de invloed van het gedistribueerd uitvoeren van een proces op de structuur van de applicatie behandeld. De indrukken die ontleend kunnen worden aan de genoemde voorbeelden van bestaande toepassingen, leiden tot een algemeen beeld van gedistribueerde applicaties en de mogelijke functionaliteiten daarvan, zoals:

- *Communicatie.*
  Communicatie met andere deelprocessen van hetzelfde programma, of met (deelprocessen van) andere programma's, wordt in deze scriptie wordt beperkt tot communicatie via het TCP/IP protocol, of via een protocol dat gebruik maakt van TCP/IP. Hierbij is het nog interessant om op te merken dat niet alleen data, maar ook 'code' overgestuurd kan worden. Hierbij kan bijvoorbeeld aan Java applets worden gedacht.
- *Application Logic*.
  Hieronder wordt verstaan: alle bewerkingen die op gegevens uitgevoerd worden. Als voorbeeld hiervan werd het uitrekenen van een rentepercentage genoemd. Vaak is dit een belangrijk deel van de werking van een programma.
- *Gegevensverschaffing.*
  Om gegevens te bewerken, dienen die gegevens allereerst ergens vandaan gehaald te worden. De gegevens kunnen uit verschillende bronnen komen. Veelgebruikte bronnen zijn een database of bestanden die op een server staan.
- *Interface.*
  Door een Interface aan de gebruiker te verschaffen, geeft de applicatie de gebruiker een manier om gegevens in te voeren, bepaalde opties te kiezen of uitvoer van gegevens te bekijken.

Een algemene structuur van gedistribueerde applicaties is af te leiden aan de hand van de bovenstaande gegevens. Hoewel niet voor alle gedistribueerde applicaties relevant, is deze structuur voor veel applicaties wel van toepassing. Uit het voorgaande kunnen we afleiden dat een gedistribueerde applicatie typisch bestaat uit een interface met de gebruiker, een Application Logic

gedeelte dat zorgt voor berekeningen en dat de juiste gegevens op de juiste plaats komen, en een gedeelte dat voor de gegevensverschaffing zorgt. In figuur 2.5 is deze structuur weergegeven. Tevens is getracht de eigenschappen van gedistribueerde applicaties weer te geven. Het gebruik van het gearceerde Application Logic deelproces door meerdere applicaties, duidt op *resource sharing*. Het gebruik van een aantal Application Logic deelprocessen geeft *parallel processing* weer. De replicatie van de gegevens en gegevensverschaffing leidt to een grotere beschikbaarheid (*availability). Load balanding* en *scalability* zijn moeilijk te visualiseren, en zijn daarom niet weergegeven in de figuur.

De verschillende functies hoeven niet in verschillende deelprocessen van de applicatie te zijn ondergebracht. Het is heel goed mogelijk om twee of meerdere functies (of allemaal) onder te brengen in één deelproces. Dit wordt verder uitgewerkt in het volgende hoofdstuk, net zoals hoe de communicatie precies verloopt tussen de deelprocessen, en wat de eigenschappen van die communicatie zijn.



Figuur 2.5: Mogelijke architecturen van twee uitgebreide gedistribueerde applicaties.

In paragraaf 2.4 zijn de prestaties van gedistribueerde applicaties behandeld. Er werden drie aspecten genoemd met betrekking tot de snelheid van gedistribueerde applicaties. Dit waren:

- Laadtijd
- Communicatietijd
- Totale executietijd

Tevens werd, in versimpelde vorm, de formule voor de totaal benodigde tijd voor de uitvoering van de taken van een gedistribueerde applicatie gegeven:

$$T_{totaal} = T_{laad} + T_{communicatie} + T_{executie}$$

3 EIGENSCHAPPEN VAN DE COMMUNICATIE

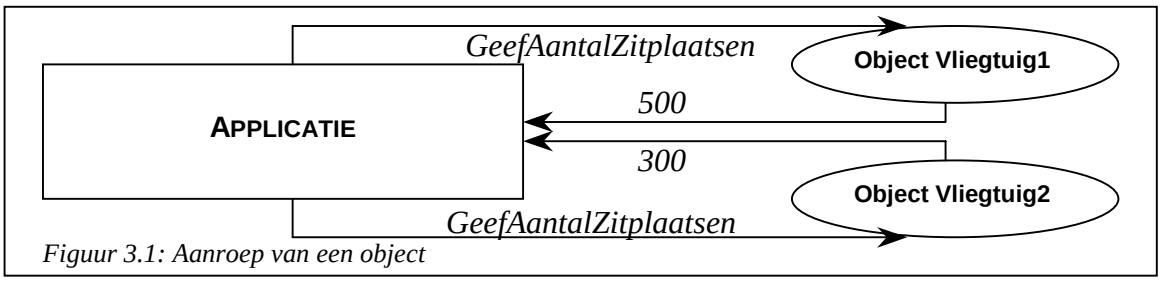
Nu wat meer bekend is over de globale structuur van gedistribueerde applicaties, kunnen we ons bezighouden met de communicatie tussen de deelprocessen daarvan. Zonder de mogelijkheid tot communicatie kunnen de verschillende onderdelen geen gegevens uitwisselen. Het worden dan geïsoleerde 'eilandjes', allemaal in staat om hun eigen taak uit te voeren (voor zover daar de gegevens voor zijn), maar niet in staat om de resultaten naar de andere eilandjes te sturen.

3.1 Terminologie

Voordat we echter beginnen, zullen we eerst de gebruikte termen scherper definiëren. In dit hoofdstuk, en de hoofdstukken hierna, zal gesproken worden over objecten, componenten en tiers.

3.1.1 Objecten

De term 'object' is afkomstig van het principe 'Object Oriented' programmeren. Het idee achter een object is om methoden (bewerkingen op gegevens) en attributen (eigenschappen) op een logische wijze te groeperen. Om een voorbeeld te geven: stel dat een programmeur het gedrag van vliegtuigen in een programma wil verwerken. Dan kan hij een *class* (klasse) 'vliegtuig' aanmaken met attributen zoals gewicht, spanwijdte, aantal zitplaatsen, etc. Dan kunnen van de algemene class 'vliegtuig' meerdere instanties gecreëerd worden, om zo verschillende vliegtuigen weer te geven (bijvoorbeeld één met 300 zitplaatsen en één met 500 zitplaatsen). Zo'n instantie van een class heet een object. Elk object heeft een 'interface' ten opzichte van de buitenwereld. Zo bepaalt de programmeur hoe een instantie van een object aangesproken moet worden (voorbeeld: om het aantal zitplaatsen van een vliegtuig te weten, moet de methode *GeefAantalZitplaatsen* aangeroepen worden). Zie ook figuur 3.1.



Figuur 3.1: Aanroep van een object

3.1.2 Componenten

Wat tot nu toe als 'onderdelen van een gedistribueerde applicatie' werd genoemd, is beter te omschrijven met de term component. Een component voert zodoende een deelproces uit, zoals dat in het vorige hoofdstuk werd

gedefinieerd. Met het creëren van componenten wordt bedoeld het opbouwen van een applicatie uit diverse losstaande onderdelen. Indien een applicatie uit meerdere componenten bestaat, kan die applicatie verdeeld worden over verschillende locaties. Het verschil met objecten, is dat componenten op een hoger niveau staan. Zo kan een component meerdere objecten bevatten.

'Component' is een beladen term. Er is veel te doen omtrent het begrip 'Component based development'. Er is geen eenduidige definitie van het begrip 'component'. Veelal worden componenten omschreven aan de hand van ruime definities, zoals die van Clemens Szyperski in zijn boek 'Component Software' [SZYP, 1998]:

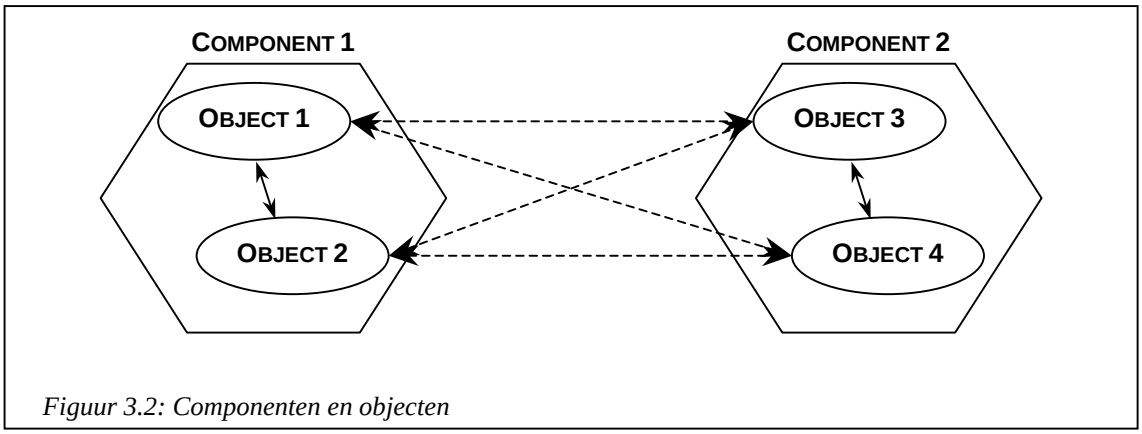
> "*Software components are binary units of independent production, acquisition, and deployment that interact to form a functioning system.*"

De term 'component', zoals die in deze scriptie gebruikt wordt, gaat niet uit van een enkele definitie, maar is gebaseerd op het algemene begrip. Met de term 'component' wordt in deze scriptie bedoeld:

> *"Een gedeelte van een applicatie dat, indien uitgevoerd op een computer, zelf in staat is een deelproces uit te voeren."*

Deze scriptie beperkt zich tot componenten die communiceren met componenten op andere machines op basis van het TCP/IP protocol, of een protocol dat hier 'bovenop' ligt"

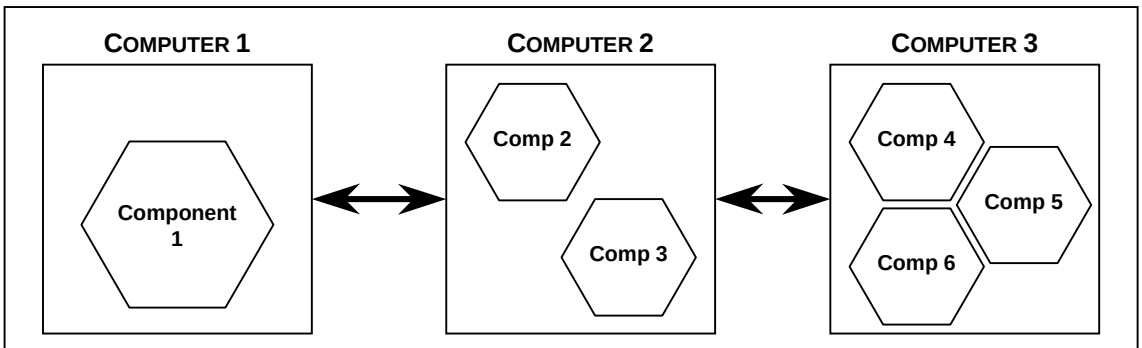
De definitie kan worden toegelicht aan de hand van figuur 3.2. De gestippelde pijlen stellen communicatie voor tussen objecten in verschillende componenten, de 'normale' pijlen communicatie binnen de componenten. De communicatie tussen de objecten vindt plaats op basis van het TCP/IP protocol of een protocol dat hiervan gebruik maakt, zoals HTTP (zie hiervoor het TCP/IP model in de paragraaf 'Eigenschappen van communicatie via het Internet', verderop in dit hoofdstuk).



Figuur 3.2: Componenten en objecten

3.1.3 Tiers

Door een applicatie te verdelen in meerdere componenten, ontstaat de mogelijkheid om de applicatie te verdelen over verschillende lagen. Deze lagen, die vaak horizontaal naast elkaar worden afgebeeld, zijn andere lagen dan die verderop genoemd zullen worden bij de bespreking van de OSI en TCP/IP modellen, die vaak verticaal boven op elkaar worden afgebeeld. Om verwarring te voorkomen, worden de 'horizontale' lagen, overeenkomstig de benaming in de literatuur, tiers genoemd. Een tier komt dus overeen met een brok software. Elke tier bestaat dan uit één of meerdere componenten. Er bestaan verschillende mogelijkheden om applicaties over tiers te verdelen. Hiervoor wordt verwezen naar het gedeelte verderop over tier structuur in de paragraaf 'Component architectuur'. De tier structuur staat in grote mate los van de gekozen opdeling in componenten. Zo kan de applicatie in honderd componenten zijn onderverdeeld, die allemaal in één tier zijn bevat. Er kunnen uiteraard niet méér tiers zijn dan componenten. Figuur 3.3 geeft dit weer, waarbij elke tier op een aparte computer staat. Dit is echter niet noodzakelijk.



Figuur 3.3: Tiers en componenten

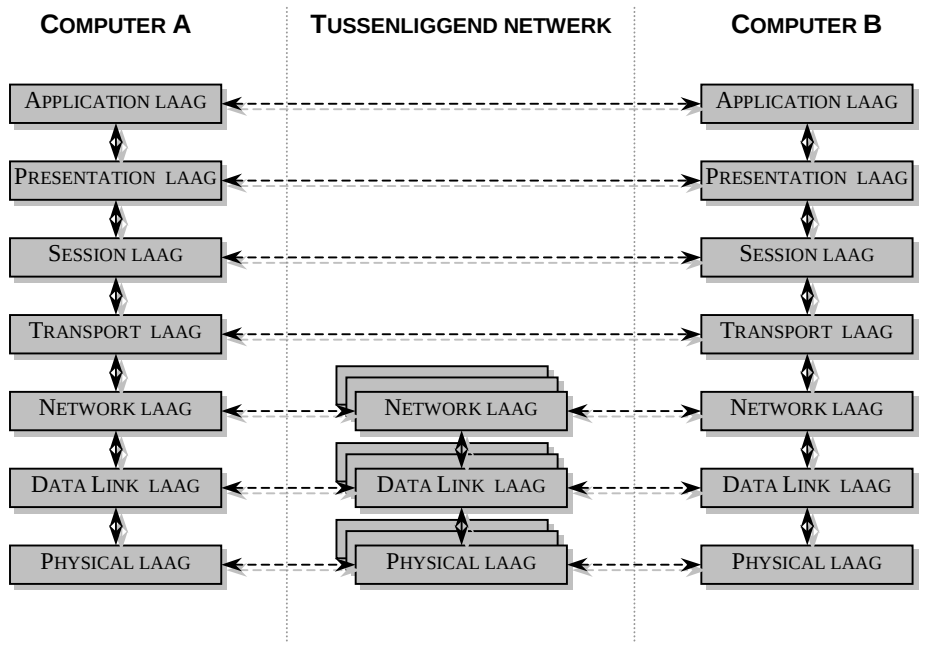
3.2 Eigenschappen van communicatie via het Internet

Om wat meer te vertellen over de eigenschappen van de communicatie zelf, gaan we eerst dieper in op de onderliggende structuur van het Internet. Daaruit kunnen we dan enkele kenmerkende eigenschappen van de communicatie afleiden.

De structuur van het Internet bestaat uit vele 'lagen'. De onderste laag is de fysieke verbinding tussen twee punten. Elke laag die daar 'bovenop' ligt, maakt gebruik van die fysieke verbinding, maar voegt zelf nog wat extra functionaliteit toe. Twee belangrijke netwerk-architecturen, die de werking van deze lagen definiëren, zijn het ISO-OSI Reference Model, en het TCP/IP Reference Model. ISO staat voor International Standards Organization en OSI staat voor Open Systems Interconnection. Dit model, vaak kortweg het OSI model genoemd, richt zich op het standaardiseren van de communicatie tussen systemen. Het TCP/IP Reference Model is genoemd naar zijn belangrijkste twee protocollen: TCP en IP. Beide modellen zijn opgemaakt uit verticale lagen. Elk model specificeert een *protocol* voor elke laag. Een protocol is een reeks afspraken over hoe de communicatie plaats dient te vinden. Het kan gezien worden als een afspraak om een bepaalde taal te spreken. Een verzameling van die lagen wordt in de literatuur vaak als een stapel afgebeeld. Zo'n stapel lagen wordt dan een *protocolstack* genoemd.

3.2.1 Het ISO – OSI Reference Model

Het OSI model bestaat uit zeven lagen. Elke laag bouwt voort op de functionaliteit die de onderliggende laag biedt, en hoeft zodoende niets te weten over de lagen die daar weer onder liggen. Elke laag biedt zijn diensten aan de laag die daar direct boven ligt aan. Zoals in figuur 3.4 staat geschetst, lijkt het voor elke laag alsof direct met dezelfde laag op de andere computer wordt gecommuniceerd. Hoe de onderliggende lagen met elkaar communiceren is transparant voor de bovenliggende laag. Een laag biedt zijn gegevens aan een onderliggende laag aan, waarna die laag zijn eigen informatie toevoegt. Deze informatie kan worden begrepen door dezelfde laag op de andere computer. Het toevoegen van die informatie in de vorm van zogenaamde 'headers' kan vergeleken worden met het versturen van een brief in een enveloppe. Op de enveloppe staat waar die naartoe moet (de adressering), en mogelijk extra informatie (zoals een stempel 'deze kant boven'), zodat de postdienst weet wat er precies moet gebeuren. Deze nieuwe gegevens worden doorgegeven aan de volgende laag, waarna die weer zijn specifieke informatie toevoegt. We lichten elke laag kort toe. Meer informatie is te vinden in Tanenbaum's 'Computer Networks' [TANE, 1996].



Figuur 3.4: ISO – OSI Reference Model

De *Physical* laag zorgt voor de communicatie over het fysieke medium (de kabel): nulletjes en eentjes worden hier verstuurd. De taak van de *Data Link* laag is om te zorgen dat het medium waarover gecommuniceerd wordt, vrij van fouten lijkt. Als resultaat van deze maatregelen kan de Data Link laag een foutvrije communicatie aanbieden als dienst aan de *Network* laag. De *Network* laag zorgt voor de werking van het *subnet*. Het subnet is het netwerk dat de computers met elkaar verbindt. Deze laag zorgt voor de *routing* (zorgen dat de gegevens uiteindelijk bij de goede computer aankomen), de *congestion control* (zorg dragen dat het netwerk niet 'verstopt' raakt door overmatig 'netwerkverkeer') en voor de overgang van gegevens tussen verschillende soorten netwerken. De *Transport* laag zorgt dat de manier waarop de communicatie plaatsvindt, volkomen transparant is voor de Session laag.

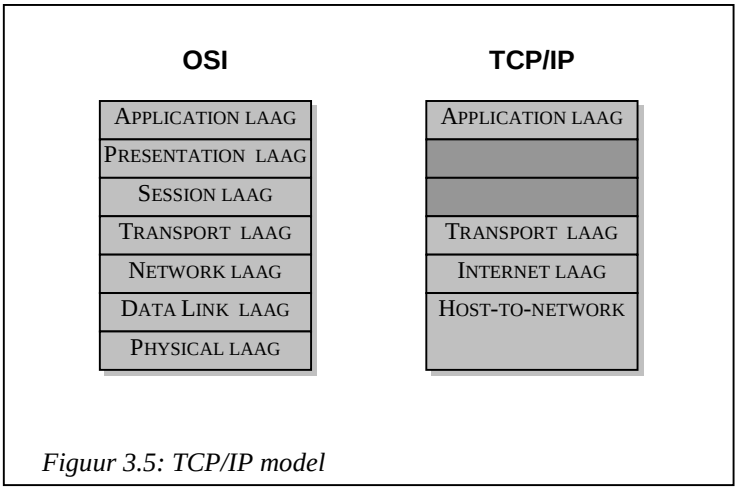
De communicatie van de Transport laag vindt plaats op een *end-to-end* niveau. Dit houdt in, dat de Transport laag op Computer A (zoals in figuur 3.4 te zien is) direct met de Transport laag op Computer B communiceert. In de lagere niveaus communiceren de lagen met de lagen op de volgende computer in het pad naar de uiteindelijk ontvanger. Dit principe is geïllustreerd als het 'tussenliggende netwerk' in figuur 3.4.

De *Session* laag staat gebruikers op verschillende machines toe een 'sessie' te openen. Een sessie geeft de mogelijkheid voor normaal gegevensverkeer, maar ook voor wat meer geavanceerde functionaliteit, zoals het inloggen op een andere machine, of het transporteren van een bestand tussen twee machines. De Session laag fungeert als een soort

verkeersagent die het verkeer regelt. De *Presentation* laag houdt zich bezig met de opmaak en inhoud van de informatie die overgestuurd wordt. Een typisch voorbeeld van een dienst die de Presentation laag aanbiedt, is om de gegevens op een bepaalde manier op te maken. De gegevens die overgestuurd worden zijn van een bepaald type: een tekstregel, een getal, et cetera. Verschillende soorten computers hebben verschillende manieren om die gegevens weer te geven. Om dan toch gegevens tussen verschillende computers uit te kunnen wisselen, gebruikt de Presentation laag een standaard manier om gegevens over het netwerk te vervoeren, waarna dezelfde laag de gegevens zodanig weergeeft, dat de computer ze begrijpt. Andere diensten zijn bijvoorbeeld encryptie en compressie. In de *Application* laag zit een aantal protocollen die algemeen gebruikt worden. Voorbeelden hiervan zijn het HTTP protocol (HyperText Transfer Protocol) dat op het World Wide Web gebruikt wordt, maar ook andere, zoals protocollen om e-mail te kunnen lezen.

3.2.2 Het TCP/IP Reference Model

Het TCP/IP model is ontstaan tijdens de totstandkoming van het ARPANET (zie hiervoor de inleiding en Tanenbaum). Het TCP/IP model bestaat ook uit lagen, maar minder dan het OSI model. De functionaliteit van enkele lagen komt overeen met die van bepaalde lagen in het OSI model. Andere lagen, die wel in het OSI model aanwezig zijn, zijn totaal niet in het TCP/IP model aanwezig. Figuur 3.5 verduidelijkt dit.



Figuur 3.5: TCP/IP model

Binnen het TCP/IP model is geen specificatie gegeven over hoe de lagen onder de Internet laag ingevuld moeten worden. Voor de *Host-to-network* laag is geen protocol gedefinieerd, en het verschilt van netwerk tot netwerk. De *Internet* laag houdt het hele model bij elkaar. De Internet laag stelt computers in staat pakketjes het netwerk op te sturen, en zorgt dat ze, los van elkaar, op de juiste plek aankomen. Er wordt dus niet

gegarandeerd dat de pakketjes in de juiste volgorde aankomen; hier moet een hogere laag voor zorgen. Het protocol dat bij de Internet laag hoort is het *Internet Protocol* (IP). De taak van de TCP/IP *Transport* laag is dezelfde als van de Transport laag in het OSI model: computers de mogelijkheid geven om 'direct' datapakketjes naar elkaar te sturen, zonder dat de laag zich bezig houdt met het onderliggende subnet. In deze laag zijn twee protocollen gedefinieerd: TCP (Transmission Control Protocol) en UDP (User Datagram Protocol). TCP is een betrouwbaar protocol dat zorgt dat een reeks bytes van de ene machine 'onbeschadigd' op de andere machine aankomt. UDP is een protocol dat geen garantie geeft voor betrouwbare communicatie. Onbetrouwbaar houdt hier in, dat niet gegarandeerd wordt dat pakketjes goed bij de andere computer aankomen. Het wordt gebruikt in toepassingen waarbij het belangrijker is dat pakketjes op tijd aankomen, dan dat ze juist aankomen. Een voorbeeld hiervan is video via Internet. In het kader van gedistribueerde applicaties is het van belang dat de gegevensstroom betrouwbaar is, dus zal geen gebruik worden gemaakt van het UDP protocol. In het TCP/IP model zijn geen Session en Presentation lagen aanwezig, omdat de noodzaak van die lagen niet ingezien werd. Volgens Tanenbaum [TANE, 1996] heeft ervaring met het OSI model uitgewezen, dat deze opvatting juist was. In de *Application* laag zijn alle high-level protocollen aanwezig, variërend van HTTP tot SMTP (Simple Mail Transfer Protocol, een protocol voor e-mail berichten). De Application lagen van het TCP/IP model en het OSI model hebben een vergelijkbare functie.

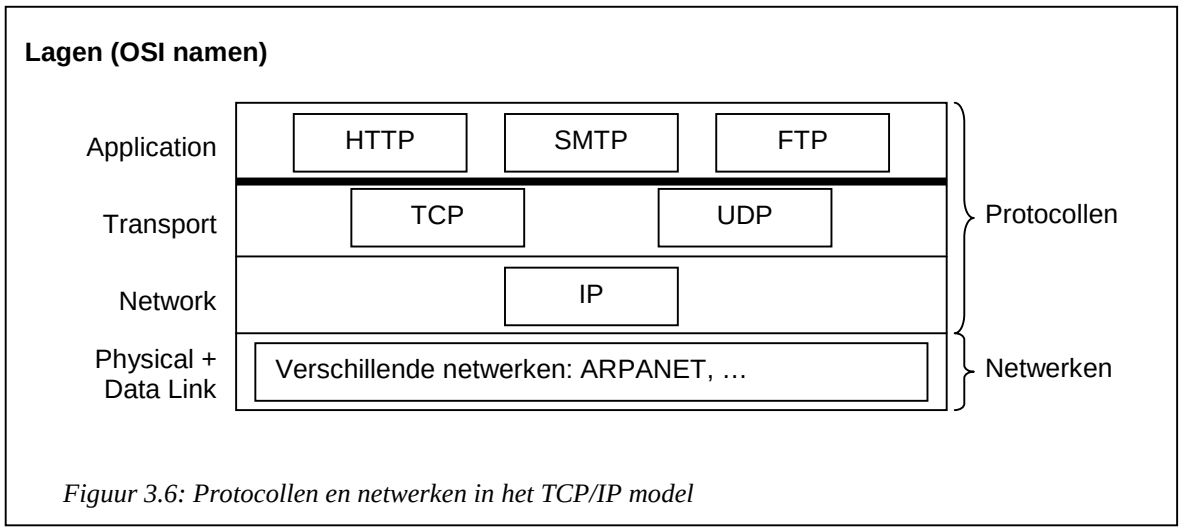
3.2.3 OSI versus TCP/IP

Zonder in details te treden, kan over het OSI *model* gezegd worden, dat het goed te gebruiken is om op basis daarvan de opbouw van een computer netwerk uiteen te zetten. De OSI *protocollen* daarentegen, die voor elke laag in het model zijn gedefinieerd, zijn niet populair geworden. Voor TCP/IP geldt het omgekeerde: het TCP/IP *model* bestaat nagenoeg niet, maar de *protocollen* zijn veel gebruikt. Voor de beargumentering van bovenstaande conclusies wordt verwezen naar [TANE, 1996].

In de inleiding werd al vermeld, dat het Internet gebruik maakt van het TCP/IP protocol. Andere protocollen, die in het model in een hogere laag zitten, worden echter ook veel gebruikt bij gedistribueerde applicaties. Hiervan is HTTP een uitstekend voorbeeld. Het verband tussen die protocollen, en het onderliggende netwerk, is weergegeven in figuur 3.6.

In figuur 3.6 staat ook een dikke lijn als scheiding tussen de Transport laag en de Application laag. Deze lijn geeft een kader aan voor deze scriptie. Onder deze lijn liggen de diensten die de TCP en IP protocollen leveren. Die diensten kunnen we dus als een vast gegeven beschouwen. De diensten die door de lagen geleverd worden, die boven deze scheidslijn liggen, dienen door de applicatie zelf verzorgd te worden,

indien gewenst. De aard van de diensten die het TCP/IP protocol levert, wordt hieronder toegelicht.



Figuur 3.6: Protocollen en netwerken in het TCP/IP model

3.2.4 Eigenschappen van communicatie via TCP/IP

Zoals in de beschouwing van de Transport laag van het TCP/IP model al vermeld werd, zorgt het TCP/IP protocol voor een (bijna) fout-vrije communicatie tussen twee computers. Een eigenschap van het Internet waar het TCP/IP protocol geen oplossing voor biedt, is het feit dat de communicatie niet veilig is. De gegevens worden namelijk verstuurd via andere computers dan de zender en de ontvanger. Dit zorgt ervoor dat mensen die gegevensstromen kunnen ‘aftappen’. Die mensen kunnen de protocolstack die op hun computer aanwezig is, gebruiken om de gegevens te lezen. Dit is natuurlijk niet wenselijk. Het TCP/IP protocol zorgt er standaard niet voor, dat de gegevens op een zodanige manier versleuteld zijn, dat alleen de ontvanger ze kan lezen. Hier moeten dus additionele maatregelen voor getroffen worden. Maatregelen voor beveiliging van gegevensstromen (en andere aspecten van beveiliging) worden in hoofdstuk 4 uitvoerig besproken.

3.2.5 Middleware

TCP/IP verschaft de mogelijkheid om bijna foutloos gegevens over te sturen naar andere computers. Er zijn echter technieken die deze communicatie sterk vergemakkelijken. Doordat het niveau van abstractie hoger ligt bij deze technieken, worden de communicatie-protocollen transparanter. Deze technieken worden onder de naam ‘Middleware’ in appendix B uitvoerig behandeld, maar een kleine toelichting is hier op zijn plaats.

De drie meest voorkomende vormen van middleware op het Internet zijn CORBA, COM+ en RMI. Deze worden in de appendix dan ook nadrukkelijk genoemd. Het principe is echter gelijk voor deze drie technieken. Middleware zorgt er voornamelijk voor, dat het ontwikkelen van gedistribueerde applicaties makkelijker is. Het zorgt er namelijk voor, dat met objecten die in een andere component zitten, gecommuniceerd kan worden, alsof ze binnen dezelfde component zitten. Door deze eigenschap zal middleware een belangrijke rol spelen bij de ontwikkeling van complexe gedistribueerde applicaties.

Middleware zorgt er niet alleen voor, dat de ontwikkeling van gedistribueerde applicaties makkelijker is, maar ook dat de prestaties beter zijn. Dit is bijvoorbeeld mogelijk als twee componenten op dezelfde machine draaien, waardoor de netwerk-aanroepen eigenlijk niet nodig zijn. Dit kan door middleware gedetecteerd worden, waardoor een meer directe communicatie mogelijk wordt. Voor meer informatie wordt verwezen naar paragraaf 3.3.3 en appendix B.

3.3 Distributie architecturen

3.3.1 Twee invalshoeken

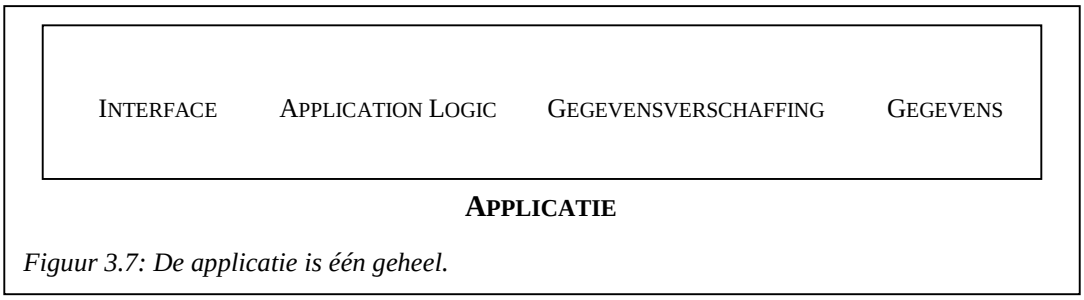
De mogelijke architecturen van gedistribueerde applicaties kunnen vanuit twee invalshoeken beschouwd worden. Allereerst is de verdeling van de componenten van belang. Daarnaast is de communicatie tussen de tiers een aandachtspunt. Beide aspecten worden hieronder behandeld.

3.3.2 Component architectuur

In deze paragraaf wordt de component architectuur behandeld. Aspecten waar aandacht aan besteed moet worden zijn de opsplitsing in een *n*-tier structuur, waarbij de *n* bepaald moet worden, en de positionering van de componenten.

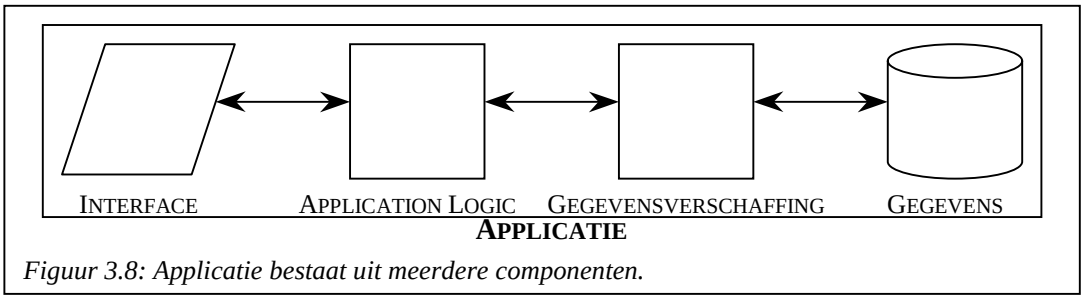
3.3.2.1 Component structuur

Historisch gezien waren in een programma de gegevens nauw verweven met het programma. Er was geen duidelijke scheiding aan te geven tussen programma en gegevens, alsmede tussen programma-onderdelen (componenten) onderling. Gedistribueerde applicaties zijn op deze manier niet mogelijk. De structuur van dit soort applicaties is weergegeven in figuur 3.7.



Figuur 3.7: De applicatie is één geheel.

Deze structuur heeft echter nadelen. Meer mensen willen tegelijkertijd met de gegevens werken. Dit betekent dat de gegevens (en gegevensverschaffing) op een centrale plaats moeten staan, terwijl de rest van de applicatie zich bij verschillende gebruikers bevindt. Indien meerdere gebruikers gelijktijdig van één systeem gebruik maken wordt dat een multi-user omgeving genoemd. Ook aan deze situatie kleven nadelen. Zo is er het beheersaspect. Indien een wijziging in de Application Logic moet worden aangebracht, zal het gedeelte van de applicatie dat bij de gebruikers staat, bij alle gebruikers aangepast moeten worden. Het is beter om deze Application Logic in een aparte component te plaatsen, en deze ook op een centrale machine te plaatsen, zodat het beheren van de Application Logic van de applicatie een stuk gemakkelijker wordt. Deze situatie (Interface, Application Logic en gegevens(verschaffing) in verschillende componenten) is weergegeven in figuur 3.8.



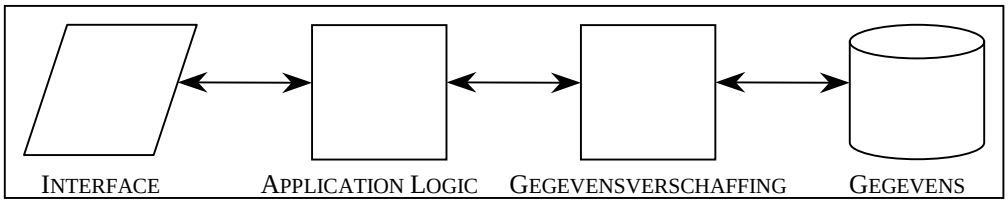
Figuur 3.8: Applicatie bestaat uit meerdere componenten.

3.3.2.2 Tier structuur

De tier structuur geeft aan in op welke wijze een applicatie over verschillende computers verdeeld kan worden, indien de applicatie uit meerdere componenten bestaat. Onderscheiden worden hier 1-tier, 2-tier en multi-tier (aantal tiers is 3 en groter) structuren.

1-tier

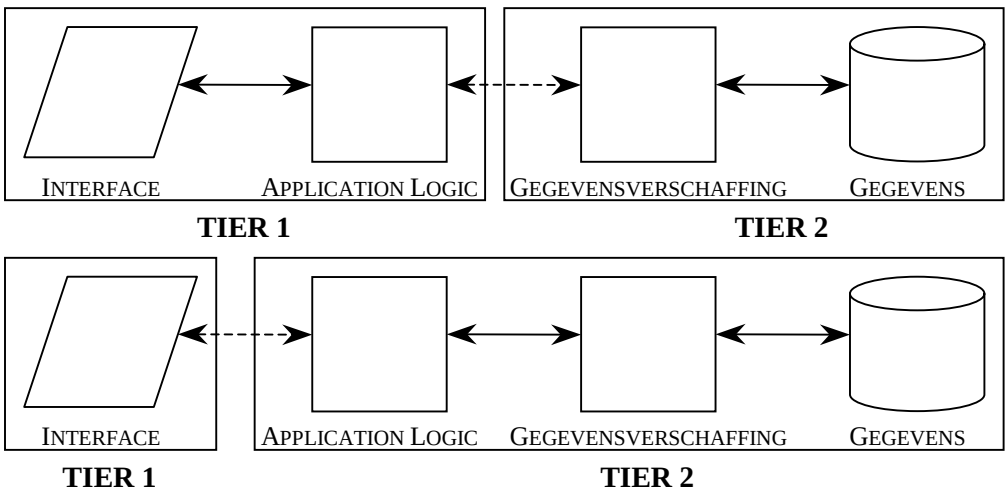
Een 1-tier structuur houdt in dat de gehele applicatie zich op één machine bevindt. Daardoor wordt geen van de voordelen van gedistribueerde applicaties gerealiseerd. Deze situatie is schematisch weergegeven in de volgende figuur. Het kader geeft aan dat alle onderdelen binnen één tier bevat zijn:



Figuur 3.9: 1-tier structuur.

2-tier

Indien voor de mogelijkheid wordt gekozen om de gegevens (en gegevensverschaffing) op een centrale computer te plaatsen, zijn er twee mogelijkheden voor de verdeling van de overige componenten. Deze twee situaties zijn in figuur 3.10 weergegeven. Bij de eerste manier zitten de gegevens en de gegevensverschaffing in één laag, en de rest van het programma in de andere. Bij de tweede manier is alleen de interface in één laag geplaatst, en de rest in de andere. Vandaar de naam 2-tier. Beide kunnen gezien worden als een vorm van Client/Server architectuur. Gestippelde pijlen tussen de tiers geven communicatie via een netwerk aan. De kaders geven aan dat de componenten verdeeld zijn over twee tiers:



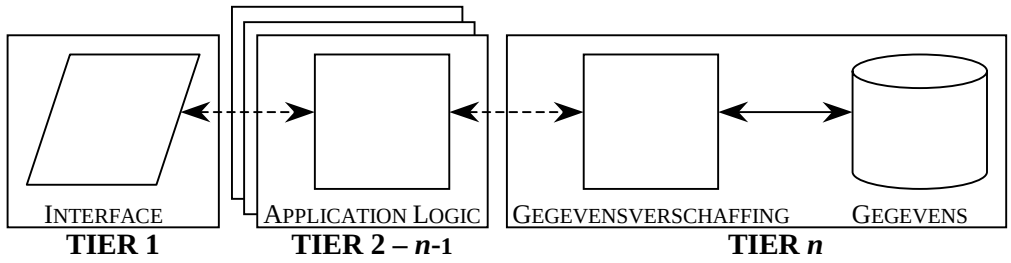
Figuur 3.10: 2-tier structuur.

Multi-tier

Ondanks de voordelen die een 2-tier structuur biedt, zijn er ook nadelen aan verbonden. Neem de structuur dat de interface en Application Logic op één tier staan (bij de client), en de rest op een andere. Deze situatie is uit beheer-oogpunt gezien niet wenselijk (zie ook paragraaf 3.3.2.1). Ook gezien de huidige snelheid waarmee bestanden over het Internet verstuurd kunnen worden, is

dit geen wenselijke situatie. Dit probleem is echter niet aanwezig indien de Application Logic op een andere tier staat dan bij de client. Dit is bij een 2-tier structuur mogelijk door de Application Logic op dezelfde tier te zetten als de gegevensverschaffing en de gegevens.

Als echter *parallel processing* gebruikt moet worden, dienen verschillende delen Application Logic op verschillende machines te staan. Uit oogpunt van flexibiliteit is het te verkiezen de Application Logic op een aparte tier te zetten. Aangezien deze structuur uit drie tiers bestaat, wordt het een 3-tier structuur genoemd. Als verschillende delen Application Logic op verschillende machines staan, ontstaan er meer dan 3 tiers. Dit wordt dan aangeduid met de term multi-tier. In de volgende figuur wordt deze situatie grafisch toegelicht.



Figuur 3.11: Multi-tier structuur.

Aangezien de gegevensverschaffing nauw verbonden is met de gegevens zelf (in veel gevallen een database), zal dit deelproces op dezelfde tier draaien als waar de gegevens staan. Een andere situatie zou in onnodig veel netwerkverkeer resulteren.

3.3.2.3 Conclusie

Uit bovenstaande beschouwing kan afgeleid worden, dat de 2-tier en multi-tier structuur beide gebruikt kunnen worden voor gedistribueerde applicaties. Met het oog op de beheersbaarheid van de applicatie (doorvoeren van wijzigingen in de programmatuur) en het benutten van de voordelen van gedistribueerde applicaties (zoals *parallel processing*, *multi-user situatie*), is de meer flexibele multi-tier structuur aan te bevelen.

3.3.3 Communicatie architectuur

Ook moet gekeken worden hoe de deelprocessen (componenten) met elkaar communiceren (welke kanalen, welk protocol). De componenten wisselen gegevens met elkaar uit op basis van protocollen, een vorm van gedragscodes waaraan beide componenten zich dienen te houden willen ze verstaanbaar zijn voor de ander. Protocollen zijn er op verschillende

niveaus. In paragraaf 3.2 werden de verschillende lagen van de OSI en TCP/IP modellen besproken en de daarbij behorende protocollen. Tevens werd in die paragraaf gezegd dat we uit gaan van het TCP/IP protocol. Dit protocol zorgt voor een bijna-foutloze, niet-veilige verbinding tussen twee computers, zoals reeds eerder werd vermeld.

Naast het TCP/IP protocol zijn er nog andere protocollen die van belang zijn voor deze scriptie. Eén daarvan is het al eerder genoemde HTTP protocol (HyperText Transfer Protocol). Dit protocol is gedefinieerd in de *Application* layer. Ook bieden de verschillende vormen van middleware hun eigen protocol. Een voorbeeld hiervan is het IIOP protocol (Internet InterOrb Protocol) wat door CORBA gebruikt wordt voor de communicatie tussen de ORB's (Object Request Brokers).

3.3.3.1 Werking van de verschillende protocollen

Voordat de voor- en nadelen van elk protocol behandeld worden, wordt eerst kort toegelicht hoe elk protocol werkt.

TCP/IP

Op basis van het IP-adres van een computer, kan het IP protocol die computer precies lokaliseren. Een IP adres in de zogenaamde 'dot-notatie' bestaat uit vier getallen tussen de 0 en 255, gescheiden door punten. De TCP/IP software (en ondersteunende software en hardware) zorgt voor 'virtuele communicatiekanalen', zogenaamde 'ports'. Door gegevens via het TCP/IP protocol naar een bepaalde port op een andere computer te sturen, kan direct met een component op die computer gecommuniceerd worden. TCP/IP kan, net zoals hieronder bij HTTP is beschreven, gebruik maken van het DNS om een IP adres te vinden.

HTTP

Op het World Wide Web communiceren computers met elkaar op basis van het HTTP protocol. HTTP maakt gebruikt van de diensten van het TCP/IP protocol. Zoals TCP/IP IP adressen gebruikt, gebruikt HTTP zogenaamde URL's (Uniform Resource Locator) om andere computers te vinden. Een URL bestaat uit drie delen: het protocol dat gebruikt wordt, de domeinnaam van de machine, en een gedeelte dat aangeeft welk document opgevraagd wordt. Een voorbeeld is *http://www.mijndomein.nl/index.html*. Hierbij geeft 'http' aan dat het HTTP protocol gebruikt wordt. Het domein van de computer waarmee verbinding wordt gemaakt is 'www.mijndomein.nl'. Voor een computer die met het Internet is verbonden kan een zogenaamde domeinnaam aangevraagd worden. Dit betekent dat het IP adres van die computer geregistreerd wordt, waarna het IP adres en de domeinnaam toegevoegd worden aan het DNS (Domain Name System). Als iemand verbinding wil maken met die computer, kan hij met behulp van DNS nagaan welk IP

adres bij de domeinnaam hoort. Vervolgens kan TCP/IP met dat IP adres verbinding maken met de computer. Het document dat opgevraagd wordt, ‘index.html’, wordt daarna via het HTTP protocol van de server naar de client toegestuurd.

De belangrijkste functionaliteit van HTTP is het op een standaard-manier uitwisselen en interpreteren van gestructureerde informatie. Hiervoor maakt HTTP gebruik van standaard *commando’s* (*methods*) om gegevens op te vragen of te versturen. Deze functionaliteit is opzettelijk algemeen opgezet met het oog op toekomstige ontwikkelingen. Hiermee is HTTP ook geschikt voor andere toepassingen dan het World Wide Web.

Middleware
In paragraaf 3.2.5 werden CORBA, COM+ en RMI genoemd als vormen van middleware. Het principe achter alle drie de vormen heeft voldoende overeenkomsten om te volstaan met de beschrijving van de werking van CORBA. Meer informatie over CORBA is te vinden op de website van de OMG (Object Management Group, [OMG, 2000]). Voor informatie over COM+ en RMI wordt verwezen naar respectievelijk Microsoft ([MS, 2000]) en Java websites ([SUN, 2000], [JSOFT, 2000]).

CORBA probeert zoveel mogelijk transparant te maken hoe de communicatie tussen twee componenten verloopt. Het belangrijkste onderdeel van CORBA is de ORB (Object Request Broker). Een ORB houdt verbindingen bij van een component met andere componenten. Het is voor een component hierdoor transparant waar de andere componenten staan, op dezelfde computer of aan de andere kant van de wereld. Voor een component lijkt het alsof hij direct met de andere component communiceert.

3.3.3.2 Voor- en nadelen
De keuze voor een bepaalde communicatie architectuur hangt af van de voor- en nadelen van de bovengenoemde mogelijkheden. Elke mogelijkheid biedt zijn eigen toegevoegde waarde. Per applicatie zal het gewicht dat aan die toegevoegde waarde toegekend wordt, verschillen.

TCP/IP
Van de genoemde mogelijkheden werkt TCP/IP op het laagste niveau. Dit betekent dat het de minste diensten verschaft, het is een ‘kaal’ communicatie-kanaal. Hierdoor is TCP/IP echter wel te optimaliseren voor een specifieke applicatie, waardoor het sneller is dan de andere protocollen. Dit is het belangrijkste voordeel.

HTTP

Omdat het HTTP protocol in een hogere laag gedefinieerd is dan het TCP/IP protocol, biedt het in ieder geval de diensten die TCP/IP ook biedt. Met behulp van URL's en het DNS is een hoge mate van transparantie te bereiken. Daarnaast biedt het een hoop additionele functionaliteit voor het verzenden en interpreteren van gestructureerde informatie. Deze algemene functionaliteit is de oorzaak van de wat grotere traagheid van HTTP ten opzichte van TCP/IP (generieke services zijn altijd trager dan specifieke). Maar naarmate de functionaliteit van TCP/IP uitgebreid wordt ten behoeve van een specifieke applicatie, kruipen de prestaties van deze twee protocollen naar elkaar toe.

Middleware

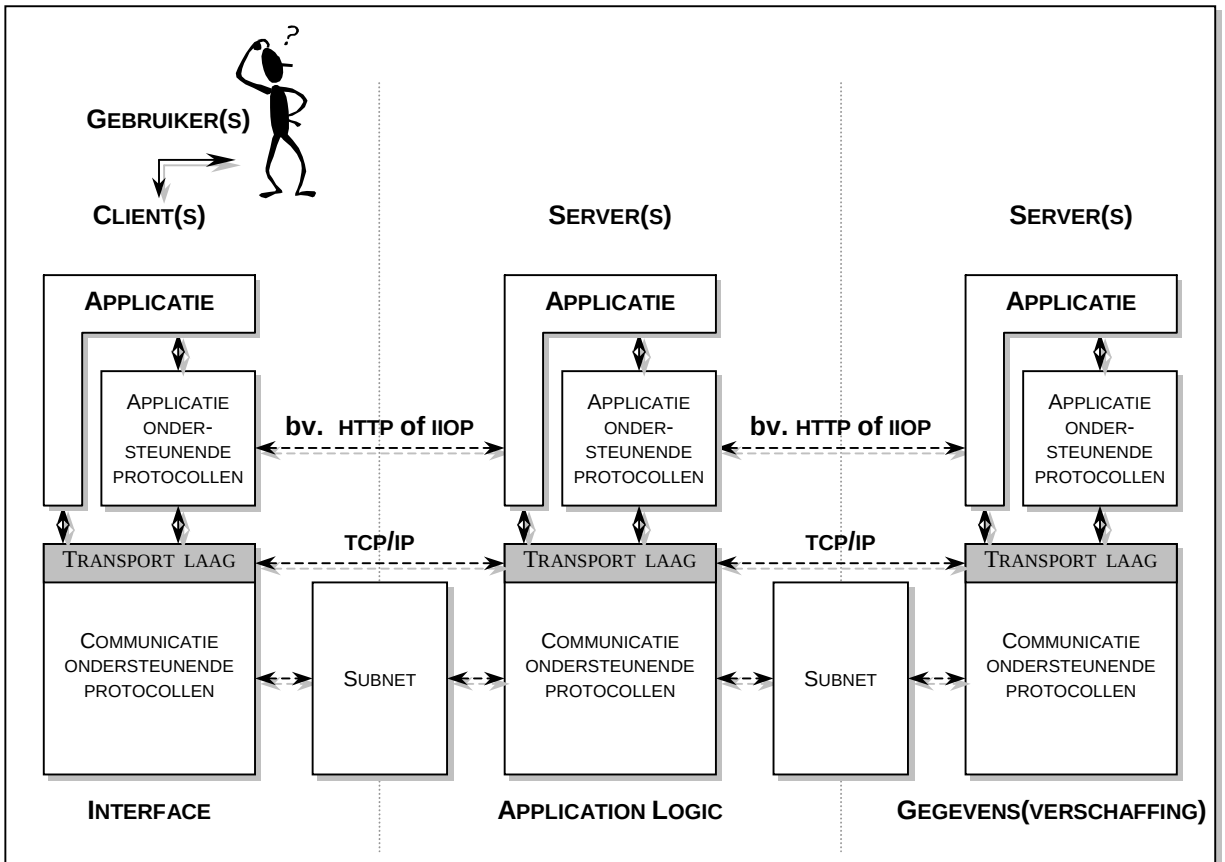
Middleware gebruikt protocollen die, evenals HTTP, gebruik maken van de diensten van TCP/IP. Evenals HTTP biedt middleware transparantie van locatie. Het niveau van transparantie is bij middleware echter hoger, omdat niet het adres van de andere computer(s) (zoals bij HTTP door middel van de URL wordt gerealiseerd) bekend hoeft te zijn. Daarnaast kan middleware helpen de mogelijke voordelen van gedistribueerde applicaties te benutten (zie ook hoofdstuk 2). Parallel processing, load balancing, scalability en robustness kunnen met behulp van een ORB gemakkelijker gerealiseerd worden, of zelfs automatisch door een ORB geregeld worden. Hetzelfde nadeel als hierboven voor HTTP genoemd werd geldt ook voor middleware: door de extra functionaliteit, is de communicatie tussen componenten op verschillende machines wat trager dan bij TCP/IP. Een keuze tussen HTTP en middleware is op voorhand moeilijk te maken. Hierbij moet wel gezegd worden dat middleware de communicatie tussen de componenten optimaliseert. Indien de componenten zich op dezelfde machine bevinden, wordt directe communicatie gebruikt in plaats van communicatie zoals die over een netwerk zou plaatsvinden. Dit komt de snelheid sterk ten goede, en in zo'n geval zou middleware snellere communicatie bieden dan TCP/IP. Aangezien we echter uitgaan van een 2-tier of multi-tier architectuur, zal er wel degelijk netwerk communicatie plaatsvinden, waardoor de snelheid van middleware beperkt wordt.

3.4 Conclusie

De resultaten van de voorgaande paragrafen kunnen goed weergegeven worden met behulp van de onderstaande figuur. We gaan uit van communicatie via het TCP/IP protocol, dus van de diensten van de *Transport* laag. De applicatie kan op verschillende niveaus binnen dit model gepositioneerd worden. De applicatie kan gebruik maken van de diensten van de lagen boven op de *Transport* laag, of alleen van de *Transport* laag. In dit

laatste geval moeten de diensten van de bovenliggende lagen (indien nodig) in de applicatie zelf verzorgd worden.

Op elke tier is dus een *protocolstack* aanwezig, met in ieder geval een *Transport* laag voor het TCP/IP protocol. Communicatie tussen de tiers zal plaatsvinden op basis van TCP/IP, en mogelijk op basis van 'hogere' protocollen. De diensten van de *Transport* laag en de lagen daaronder (samen aangeduid als *communicatie ondersteunende protocollen*) zijn in de figuur als één geheel weergegeven, om aan te geven dat deze lagen als gegeven beschouwd mogen worden. Van de overige lagen (de *applicatie ondersteunende protocollen*) is kortom niet zeker of ze door de applicatie gebruikt zullen worden. Dit is in de figuur aangegeven door de applicatie deels direct aan de *Transport* laag te laten grenzen, en deels aan de *applicatie ondersteunende protocollen*.



Figuur 3.12: Architectuur gedistribueerd applicaties

4 BEVEILIGING

Indien men over een openbaar netwerk, zoals het Internet, gegevens verstuurt, zijn die gegevens voor iedereen toegankelijk. Als een bedrijf met vertrouwelijke gegevens werkt, is het niet de bedoeling dat iedereen die gegevens kan inzien. Zoals in het vorige hoofdstuk werd vermeld, zorgt het TCP/IP protocol niet voor veilige verbindingen. Daarom dient beveiliging binnen een gedistribueerde applicatie zelf gerealiseerd te worden. In dit hoofdstuk wordt een korte schets gegeven ten aanzien van de problematiek van beveiliging. Daarnaast worden enkele methoden om gegevens te beveiligen kort toegelicht. Voor meer informatie over de onderwerpen die in dit hoofdstuk behandeld worden, en andere aspecten van beveiliging, wordt verwezen naar de scriptie van Ed Ridderbeekx ([RIDD, 1997]).

4.1 Aspecten van beveiliging

Beveiliging bestaat uit twee soorten maatregelen: preventieve en correctieve maatregelen. Preventieve maatregelen zijn gericht op het voorkomen van ongeautoriseerde handelingen. Correctieve maatregelen zijn bedoeld om te kunnen evalueren in hoeverre de preventieve maatregelen voldoende zijn geweest, en hoe ze bijgesteld moeten worden om het gewenste niveau van beveiliging te realiseren. Met de aspecten die in deze paragraaf worden behandeld, zijn vaak preventieve en correctieve maatregelen gemoeid. Het doel van deze maatregelen is niet alleen voorkomen dat gegevens door onbevoegden worden gemanipuleerd (gegevensbeveiliging), maar ook zorgen dat de werking van de applicatie niet aangetast kan worden.

Gegevensbeveiliging is zowel gericht op gegevens die op een computer zijn opgeslagen (lokale gegevensbeveiliging) als op gegevens die tussen twee computers verstuurd worden. Om een voldoende niveau van gegevensbeveiliging te realiseren, is een scala van maatregelen noodzakelijk. Niet alleen technische maatregelen (zie paragraaf 4.2) maar ook organisatorische maatregelen. Het belang van een dergelijk beveiligingsbeleid voor de organisatie wordt in paragraaf 4.1.1 beargumenteerd.

Gegevens beveiligen tegen onbevoegden heeft twee aspecten. Onbevoegden dienen de gegevens niet te kunnen manipuleren. Allereerst moet vastgesteld kunnen worden of iemand wel of niet bevoegd is bepaalde gegevens te manipuleren. Daarna moet gegevens dus verborgen of onleesbaar gemaakt worden voor die mensen die geen bevoegdheid hebben. Dit geldt zowel voor lokale gegevensbeveiliging als voor beveiliging van gegevens die verstuurd worden. Bij het versturen van gegevens dient echter ook rekening gehouden te worden met het feit, dat iedereen de gegevens kan ‘aftappen’. Er moet dus een manier gevonden worden om de gegevens onleesbaar te maken, behalve voor de zender en de ontvanger. Over dit principe, encryptie (versleuteling) genaamd, wordt in paragraaf 4.2 meer verteld.

De aspecten van gegevensbeveiliging vallen uiteen in twee categorieën: *algemene beveiligingsaspecten* en *beveiliging van gedistribueerde applicaties*. Deze worden hieronder behandeld.

4.1.1 Algemene beveiligingsaspecten

Deze paragraaf betreft algemene maatregelen die door organisatie genomen moeten worden om een succesvolle beveiliging te kunnen realiseren. Deze maatregelen betreffen het *beveiligingsbeleid* en *lokale gegevensbeveiliging*.

4.1.1.1 Beveiligingsbeleid

Als het beveiligingsbeleid van een organisatie niet afgestemd is op de overige maatregelen, krijgt men al gauw het idee van een gesloten voordeur en een openstaande achterdeur: de verbinding met het Internet zit goed dicht, maar dit heeft weinig nut indien indringers ook op andere manieren binnen kunnen komen, zoals via modems van gebruikers die vanaf de werkplek over het Web *surfen*. Een ketting is tenslotte zo sterk als zijn zwakste schakel.

Gebruikers dienen bewust gemaakt te worden van de risico's, en hun verantwoordelijkheden ten opzichte van het beveiligingsbeleid. Daarnaast is het een gezond idee om gebruikers te informeren van de maatregelen die getroffen worden, ten eerste om begrip af te dwingen voor eventuele ongemakkelijke constructies, en ten tweede om gebruikers de mogelijkheid te geven om zwakheden in of aanvallen op het beveiligingssysteem te herkennen en te melden. Ook is het verstandig om mensen bewust te maken van de eventuele vertrouwelijkheid van gegevens. Dit kan misverstanden voorkomen. Tenslotte is het voor een onderneming belangrijk om een zekere mate van expertise in de onderneming te houden of te ontwikkelen. Ook al wordt de beveiliging geregeld door externe specialisten, het is een goed idee om in ieder geval een zekere mate van kennis in huis te hebben.

4.1.1.2 Lokale gegevensbeveiliging

Gegevens die lokaal op een systeem bijgehouden worden dienen uiteraard ook beveiligd te worden tegen oneigenlijk gebruik. Deze gegevens kunnen op een aantal manieren benaderd worden door gebruikers, dus ook door onbevoegde personen. Gegevens die lokaal op een stand-alone computer staan, hoeven niet beveiligd te worden voor toegang via een externe verbinding. Maar dat betekent niet dat daarmee het risico verdwenen is. Natuurlijk is de mogelijkheid nog steeds aanwezig om fysiek de computer te benaderen. Bij veel kantoorgebouwen zijn inmiddels maatregelen genomen om bijvoorbeeld vleugels af te sluiten voor mensen die geen sleutel of chipcard hebben. Dit elimineert al wat risico's. Daarnaast is er nog het risico van inbraak, waarbij de computer gewoon meegenomen kan

worden. Indien een onbevoegde toch kans ziet om achter een computer te gaan zitten, dient voorkomen te worden dat de gegevens zomaar toegankelijk zijn. Op de meeste computers gebeurt dit door middel van passwords. Maar dat is redelijk simpel te omzeilen. Indien het bedrijfkritische gegevens betreft, dient overwogen te worden om de gegevens lokaal te versleutelen. RSA SecurPC is bijvoorbeeld een product dat gegevens versleutelt op basis van een 128 bit sleutel. Deze is dus zo goed als niet te breken, dat wil zeggen dat de gegevens (bijna) niet ontcijferd kunnen worden. Aangezien lokale gegevensbeveiliging niet specifiek van belang is voor gedistribueerde applicaties zal dit verder niet behandeld worden.

Indien wel een verbinding met het Internet of een ander extern netwerk aanwezig is, moet die beveiligd worden. Hiervoor dient een firewall gebruikt te worden. Een firewall is een combinatie van hardware en software, die tussen het externe netwerk en het interne netwerk wordt geplaatst. De functies van een firewall zijn grofweg het monitoren van het verkeer tussen het interne en externe netwerk, zodat onbevoegden geen toegang krijgen tot één van beide netwerken (mensen binnen de organisatie die geen toegang dienen te hebben tot het externe netwerk, en onbevoegden van buiten de organisatie die geen toegang tot het interne netwerk mogen hebben). Dit gebeurt op basis van verschillende technieken, waar hier niet op in wordt gegaan.

4.1.2 Beveiliging van gedistribueerde applicaties

Om gedistribueerde applicaties goed te kunnen beveiligen moet een aantal preventieve en repressieve maatregelen genomen worden. Preventieve maatregelen zijn bedoeld om misbruik te voorkomen. In deze categorie vallen *beschikbaarheid voor geautoriseerde personen*, *privacy en integriteit* en *authenticatie en non-repudiation*. Repressieve maatregelen zijn bedoeld om te kunnen achterhalen wat er fout is gegaan, als mensen de preventieve maatregelen toch omzeild hebben. *Controleerbaarheid* van in deze categorie.

4.1.2.1 Beschikbaarheid voor geautoriseerde personen

In deze categorie zijn twee punten van belang: *beschikbaarheid* (availability) en *autorisatie*.

Beschikbaarheid

De beschikbaarheid die hier wordt bedoeld is een enger begrip dan de beschikbaarheid die in hoofdstuk 2 werd beschreven. Het betreft hier voornamelijk de mogelijkheid om opzettelijke pogingen om de applicatie te overbelasten, het hoofd te bieden. Vaak is een firewall hierbij een goed hulpmiddel.

Autorisatie

Natuurlijk mogen alleen díe personen gebruik maken van de applicatie, die hiervoor toestemming hebben. In het geval van het Internet: sommige sites dienen niet voor iedereen toegankelijk te zijn. Om er zeker van te zijn dat alleen de mensen die daartoe gerechtigd zijn bepaalde sites bezoeken of bepaalde transacties doen, dienen bezoekers zich te identificeren, op basis waarvan ze dan bepaalde rechten krijgen. Dit principe heet autorisatie. Het verderop genoemde principe van authenticatie is een noodzakelijk onderdeel voor een goede autorisatie. Authenticatie gebeurt vaak op basis van de combinatie username en password, waarna aan de hand van de username de autorisatie tot het gebruik van bepaalde sites of transacties toegekend wordt.

4.1.2.2 Privacy en integriteit

Privacy en integriteit richten zich voornamelijk op de verzonden gegevens. Deze gegevens moeten onveranderd aankomen bij de ontvanger.

Privacy

De gegevens die via het Internet verstuurd worden, reizen via een aantal routers. Die informatie is zodoende voor iedereen te zien. Als het geheime of vertrouwelijke informatie betreft kan dat natuurlijk fataal zijn, maar ook indien het een onbeduidend berichtje aan een vriend is, is het niet de bedoeling dat iedereen dat zomaar kan lezen. Met andere woorden, de privacy van de communicatie is belangrijk. Vaak wordt encryptie gebruikt om ervoor te zorgen dat gegevens alleen door de bedoelde persoon verwerkt kunnen worden. Andere aspecten van beveiliging worden nutteloos indien de informatie die verzonden wordt door iedereen zomaar gelezen kan worden. Want het heeft geen nut om bijvoorbeeld in te moeten loggen om toegang tot een site te krijgen, terwijl iedereen die dat wil de verzonden username en password kan onderscheppen en zelf gebruiken.

Integriteit

De mogelijkheid bestaat dat kwaadwillende individuen de inhoud van een bericht aanpassen voordat het bij de ontvanger aangekomen is. Daarnaast kunnen ook technische factoren invloed uitoefenen op de inhoud van een bericht, zoals het omslaan van een nul naar een één door een magnetisch veld in de buurt van een kabel. In beide gevallen zijn de effecten in het minste geval uiterst vervelend. Om dit te voorkomen dient de integriteit van berichten gecontroleerd te worden. Dit heet verificatie van integriteit. Hiervoor bestaan een aantal technieken, zoals het versleutelen van het hele bericht (zodat er na het veranderen van bijvoorbeeld 1 bit alleen maar onzin ontstaat), of het inbouwen van een *checksum* (een controle-getal),

aan de hand waarvan gecontroleerd kan worden of het bericht niet veranderd is.

4.1.2.3 Authenticatie en non-repudiation

Naast het bericht zelf, dient ook de informatie over de communicerende partijen zelf betrouwbaar te zijn. Hiervoor zijn twee maatregelen van belang, namelijk authenticatie en non-repudiation

Authenticatie

Zoals gezegd moet de mogelijkheid bestaan onbevoegden te identificeren. Bij communicatie via het Internet dienen beide communicerende 'partijen' zichzelf te authenticeren. Wanneer transacties via het Internet plaatsvinden (zeker bij financiële transacties) is het van belang te weten met wie er gecommuniceerd wordt. Indien men een aankoop doet via het Internet, zal men zeker willen weten dat degene aan wie betaald wordt is wie hij zegt dat hij is. Dit is het principe van authenticatie. Bij het authenticatie proces dienen beide communicerende partijen zich te identificeren en zodanig te authenticeren dat er geen twijfel over elkaars identiteit kan bestaan.

Non-repudiation

Zoals bij E-commerce heel goed te begrijpen is, dienen sommige transacties onweerlegbaar plaats te hebben gevonden. Vooral bij financiële transacties is dit essentieel. Niet alleen dienen de beide partijen zich te authenticeren tegenover elkaar, ook moet onweerlegbaar vastgelegd kunnen worden dat een bepaald persoon (of bedrijf) een zekere transactie uitgevoerd heeft, zodat die persoon later niet kan ontkennen die transactie te zijn aangegaan. Hiertoe moet een vorm van 'digitale handtekening' aan een bericht toegevoegd kunnen worden. Hierover wordt in paragraaf 4.2 meer verteld.

4.1.2.4 Controleerbaarheid

Controleerbaarheid betreft het principe dat naderhand nagegaan kan worden waar er iets fout gegaan is. Dit gebeurt in de regel met behulp van zogenaamde *log-files*. In deze bestanden moeten alle gebeurtenissen bijgehouden worden, voorzien van een datum en een tijd, zodat gezien kan worden welke gebeurtenissen op welk moment plaatsgevonden hebben. Men kan aan de hand daarvan maatregelen nemen om ongewenste gebeurtenissen in de toekomst te voorkomen. Eventueel biedt de log-file ook de mogelijkheid om de dader op te sporen.

4.2 Beveiligingsmaatregelen

Bovenstaande beveiligingsaspecten berusten kortom op het principe dat gegevensstromen niet zomaar afgetapt en verwerkt kunnen worden. Om daarvoor te zorgen worden gegevens vaak versleuteld. Met behulp van geavanceerde algoritmen worden de gegevens omgezet naar een soort geheimschrift, dat alleen degene die een zogenaamde sleutel heeft kan vertalen naar het originele bericht. Er zijn een twee soorten encryptie (versleuteling): symmetrische en asymmetrische encryptie. Het verschil berust op de sleutel(s) die gebruikt worden voor het versleutelen en ontcijferen. Beide principes worden hieronder toegelicht. Daarna wordt een aantal beveiligingsmethoden beschreven die, gebruik makend van encryptie, mogelijkheden bieden voor het bekrachtigen van enkele (of alle) bovengenoemde beveiligingsaspecten.

4.2.1 Encryptie

4.2.1.1 Symmetrische encryptie – Secret Key

Symmetrische encryptie wordt ook wel private key of secret key encryptie genoemd. Een veel gebruikte methode hiervoor is het door de Amerikaanse overheid gesponsorde DES: Data Encryption Standard. Andere algoritmes zijn IDEA en RC4. Symmetrische encryptie berust op het principe dat dezelfde sleutel wordt gebruikt voor zowel het versleutelen als het ontcijferen. Deze sleutel dient geheim te zijn om te voorkomen dat iedereen alle berichten kan ontcijferen. Vandaar de naam Secret Key Encryptie. Tussen elke twee personen dient een secret key te bestaan om er zeker van te zijn dat alleen die twee personen de berichten kunnen lezen. Als het aantal gebruikers in een netwerk groeit, wordt dit dus in toenemende mate onhandig, duur en lastig om te beheren. Daarnaast dient het vertrouwen in de andere partij aanwezig te zijn dat die de geheime sleutel op een juiste manier zal beheren en aan niemand zal prijsgeven. Dus dient men iedereen met wie men communiceert een eigen geheime sleutel toe te vertrouwen. In praktijk gebeurt dit alleen indien al een relatie met de andere persoon aanwezig was, hetzij persoonlijk of professioneel. Aspecten als authenticatie en non-repudiation worden niet afgehandeld door DES. Het gebruik van gedeelde geheime sleutels voorkomt dat één van beide partijen kan bewijzen wat de ander gedaan zou hebben. Elk van beide kan gegevens veranderen, en omdat beide partijen dezelfde sleutel hebben, kan de schuldige niet aangewezen worden.

4.2.1.2 Asymmetrische encryptie – Public Key

De problemen die hierboven genoemd zijn werden theoretisch behandeld in 1976 door Whitfield Diffie en Martin Hellman, toen ze hun concepten publiceerden voor het uitwisselen van geheime berichten zonder uitwisselen van geheime sleutels. Het idee kwam in 1977 tot leven met de uitvinding van het RSA Public Key

Cryptosystem door Ronald Rivest, Adi Shamir en Len Adleman, destijds professoren aan het Massachusetts Institute of Technology.

In plaats van het gebruiken van dezelfde sleutel voor zowel het versleutelen als het ontcijferen van de gegevens, gebruikt het RSA systeem een gekoppeld paar van encryptie en decryptie sleutels. Elke sleutel voert een één-richtings transformatie uit op de gegevens. Elke sleutel is de omgekeerde functie van de andere: wat de één doet, kan alleen de andere ongedaan maken. Vandaar de naamgeving asymmetrische encryptie.

RSA werkt met een Public Key en een Private Key. De Public Key wordt door zijn eigenaar openbaar gemaakt, terwijl hij zijn Private Key geheim houdt. Om een privé bericht te versturen, versleutelt de zender het bericht met de Public Key van de bedoelde ontvanger. Eenmaal op die manier versleuteld kan het bericht alleen ontcijferd worden met de Private Key van de ontvanger.

Daarnaast kan de gebruiker ook gegevens met zijn Private Key versleutelen. Met andere woorden: RSA werkt in allebei de richtingen. Dit is het principe achter de “digitale handtekening”. Als de gebruiker een bericht kan ontcijferen met de Public Key van iemand anders, moet die andere wel zijn Private Key gebruikt hebben om het bericht te versleutelen. Aangezien alleen de eigenaar zijn Private Key kan gebruiken, moet dat degene wel zijn die het bericht verstuurd heeft. Zodoende wordt het versleutelde bericht een soort elektronische handtekening: een document dat niemand anders kan maken.

RSA kan dus heel goed gebruikt worden voor authenticatie en non-repudiation. Dat gaat als volgt. Een digitale handtekening wordt gemaakt door een tekstbericht door een hashing algoritme te halen. Het resultaat hiervan is een zogenoemde *message digest*. Een message digest heeft de volgende eigenschappen:

- De digest is moeilijk om te draaien. Het originele bericht is niet terug te krijgen uit de digest.
- Het is moeilijk om een ander bericht te vinden dat dezelfde digest waarde heeft als het originele bericht.

Nadat de message digest gegenereerd is, wordt die versleuteld met de Private Key van degene die het bericht verstuurt, zodat het een digitale handtekening wordt. Deze kan alleen ontcijferd worden met de Public Key van dezelfde persoon. Degene die het bericht ontvangt ontcijfert de digitale handtekening en rekent de message digest opnieuw uit. Vervolgens wordt de waarde van deze pas uitgerekende message digest vergeleken met de waarde van de message digest van de handtekening. Indien ze overeen komen, is er niet geknoeid met het bericht. Zo is dus vast te stellen dat het bericht komt van degene die het echt verstuurd

heeft, en is dat ook meteen te bewijzen, omdat diegene de enige is die de message digest met diens Private Key heeft kunnen versleutelen.

Het is belangrijk om te realiseren dat het gebruik van Public Key Encryptie zonder additionele maatregelen nog ruimte overlaat voor een aantal vervelende mogelijkheden voor kwaadaardige personen om de communicatie in de war te gooien. Voor de mogelijkheden, en de oplossingen hiervoor, wordt verwezen naar appendix A: Public Key Encryptie.

4.2.2 Beveiligingsmethoden

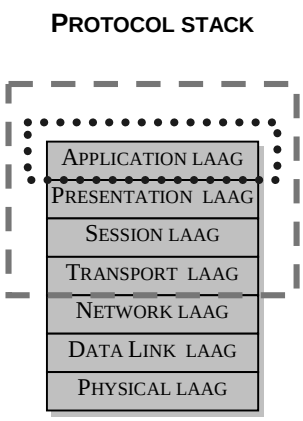
Voor het beveiligen van de communicatie tussen 2 punten bestaan verschillende mogelijkheden. Welke alternatieven mogelijk zijn wordt bepaald door een aantal factoren, zoals het soort applicatie dat beveiligd moet worden, het beveiligingsniveau dat nagestreefd wordt en de gekozen techniek voor het ontwikkelen. Hieronder worden verschillende methoden behandeld die in meer of mindere mate te maken hebben met bovengenoemde encryptie methoden.

4.2.2.1 SSL, TLS, SHTTP, PCT

SSL staat voor Secure Sockets Layer. SSL is een door de Netscape Communications Corporation ontwikkelde methode voor het beveiligen van web communicatie. Het SSL protocol biedt, op basis van encryptie, mogelijkheden voor server authenticatie, bericht integriteit en optioneel client authenticatie voor een TCP/IP verbinding. Omdat SSL in de belangrijkste browsers is ingebouwd, is het installeren van een digital certificate (zie Public Key Encryptie) genoeg om de SSL-mogelijkheden van de browsers ‘aan te zetten’. SSL is verkrijgbaar in 2 versies: 40-bit en 128-bit, wat verwijst naar de lengte van de ‘session key’ die voor het versleutelen van elke transactie wordt gebruikt en aangemaakt. Hoe langer de sleutel, hoe moeilijker het is om de code te breken. In 1995 heeft een Fransman bewezen dat 40-bit eigenlijk te kort is. Hij wist namelijk in 8 dagen met 2 supercomputers en een aantal PC’s een stuk 40-bit code te breken. De laatste versie van SSL, versie 3.1, heet Transport Layer Security (TLS). In figuur 4.1 is aangegeven dat SSL op TCP/IP niveau opereert.

Indien encryptie op een bepaalde laag in het OSI model wordt uitgevoerd, zijn automatisch alle bovenliggende lagen beveiligd, aangezien ze gebruik maken van de diensten van die beveiligde laag. Een voorbeeld van een protocol dat gebruik maakt van een beveiligde onderliggende laag is het door Netscape ontwikkelde HTTPS. HTTPS is het HTTP protocol over een TCP/IP verbinding die beveiligd is door middel van SSL (HTTPS gebruikt port 443 in plaats van HTTP port 80). Indien gebruikt wordt gemaakt van een HTTPS verbinding,

verschijnt in de URL in plaats van het protocol-voorvoegsel ‘http:’ het voorvoegsel ‘https:’. Zie ook figuur 4.1.



•••• SHTTP biedt encryptie op het niveau van de Application laag. Dit betekent dat de onderliggende lagen niet automatisch veilig zijn.

– – SSL biedt encryptie op het niveau van de *Transport* laag. In samenwerking met TCP/IP protocol wordt een veilige verbinding aangeboden, waarover andere protocollen zoals HTTP (wat dan HTTPS wordt genoemd) kan communiceren.

Figuur 4.1: Beveiligingsmethoden in de protocolstack

HTTPS dient niet te worden verward met SHTTP (Secure HTTP). SHTTP is ontworpen door EIT (Enterprise Integration Technologies) om HTTP verbindingen te beveiligen en biedt een groot aantal mechanismen om privacy, authenticatie en integriteit te verschaffen. Omdat de ontwikkelaars het belangrijk vonden om de specificatie en het mechanisme te scheiden, is SHTTP niet gebonden aan een bepaald cryptografisch systeem of encryptie algoritme. SHTTP is een superset van HTTP, waarbij berichten op verschillende manieren ingepakt kunnen worden (encryptie, digitaal tekenen, …). Dit kan recursief gebeuren. SHTTP ondersteunt verschillende methoden voor het uitwisselen van sleutels, zoals de bij Public Key Encryptie genoemde RSA methode. Net zoals bij SSL zijn client Public Keys (in certificates) niet verplicht. Om de gegevens die van de client naar de server worden verstuurd te beveiligen, is het voldoende om een certificate voor de server aan te vragen. SHTTP werkt op een hoger niveau dan SSL, namelijk op HTTP niveau in plaats van op TCP/IP niveau. Evenals SSL ondersteunt SHTTP authenticatie van zowel client als server, gegevens encryptie en digitale handtekeningen. SHTTP ondersteunt zowel symmetrische als asymmetrische encryptie. Zoals in figuur 4.1 te zien is worden onderliggende lagen niet beveiligd door SHTTP.

PCT (Private Communication Technology) is een andere specificatie voor veilige communicatie over het World Wide Web, ontwikkeld door Microsoft. PCT is in hoge mate vergelijkbaar met SSL en zal daarom verder niet apart genoemd worden.

4.2.2.2 SET

SET (Secure Electronic Transaction) is een systeem voor het beveiligen van financiële transacties over het Internet. Het wordt onder meer ondersteund door Mastercard, Visa, Microsoft en Netscape. Bij SET krijgt de gebruiker een elektronische creditcard (digital certificate, zie Public Key Encryptie) en wordt de transactie afgehandeld en geverifieerd door een combinatie te gebruiken van digital certificates en digitale handtekeningen van de koper, de verkoper en de bank van de koper. SET maakt gebruik van SSL en SHTTP.

Het werkt als volgt:

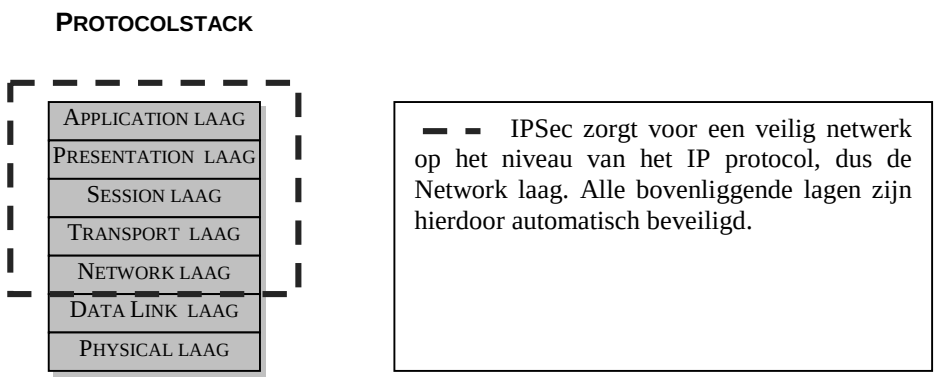
Neem aan dat de klant een browser heeft die SET ondersteunt, zoals Netscape's Navigator of Microsoft's Internet Explorer, en dat de verkoper of bank een server heeft die SET ondersteunt.

1. De klant opent een Mastercard of Visa bankrekening (iedere instantie die creditcards af kan geven is een soort bank).
2. De klant krijgt een digital certificate. Dit elektronische bestand dient als een creditcard voor on-line aankopen of andere transacties. Het bevat een Public Key met afloop datum en is door de bank digitaal 'getekend' om de geldigheid te waarborgen.
3. Verkopers ontvangen ook certificates van de bank. Deze certificates bevatten de Public Keys van de verkoper en van de bank.
4. De klant bestelt iets via een web pagina.
5. De browser van de klant ontvangt de certificate van de verkoper en bevestigt dat die geldig is.
6. De browser verstuurt de bestelling, versleuteld met de Public Key van de verkoper, informatie met betrekking tot betaling aan de bank, versleuteld met de Public Key van de bank zodat de verkoper het niet kan lezen, en informatie zodat de betaling alleen plaats kan vinden met betrekking tot deze bestelling.
7. De verkoper controleert of de klant de juiste is door de digitale handtekening op de certificate van de klant te controleren. Dat kan plaatsvinden door de certificate voor te leggen aan de bank of een 3e partij die het certificaat kan verifiëren.
8. De verkoper stuurt de bestelling door naar de bank. Hier zit dus de Public Key van de bank bij, de betalingsinformatie van de klant (wat de verkoper niet kan lezen), en de certificate van de verkoper.
9. De bank verifieert de verkoper en het bericht.
10. De bank zet een digitale handtekening en geeft de verkoper toestemming de bestelling te leveren.

4.2.2.3 IPSec

IPSec staat voor IP Security Protocol. IPSec is dé standaard voor authenticatie, integriteit en vertrouwelijkheid door middel van cryptografie op het niveau van de OSI *Internet* laag (Internet Protocol). Het zorgt ervoor dat de IP-pakketjes die tussen computers op een beveiligd netwerk worden rondgestuurd ongewijzigd, authentiek en privé zijn. IPSec is eigenlijk de mooiste oplossing voor het beveiligingsprobleem, omdat dit volkomen transparant is voor de gebruiker. Het is echter moeilijk te implementeren, omdat bij grote netwerken sleutelbeheer een grote opgave wordt. IPSec wordt daarom voornamelijk op LAN niveau gebruikt. Op een LAN worden IP pakketjes (met daarin gegevens betreffende de bestemmingsadressen, afkomstadressen en andere informatie) onbeveiligd rondgestuurd. Indien een kwaadwillend persoon de firewall weet te omzeilen kan hij de pakketjes onderscheppen, vervalsen of veranderen. Dit is te voorkomen met IPSec.

Het verschil met SSL is dat IPSec een veilig netwerk van computers over onveilige kanalen creëert. SSL werkt op het niveau van de Transport layer zodat de communicerende computer niet op hetzelfde netwerk hoeft te staan. SSL zorgt voor beveiliging tussen twee applicaties die over een netwerk communiceren, IPSec beveiligt een heel netwerk. Dit is te zien in figuur 4.2.



Figuur 4.2: De plaats van IPSec in de protocolstack

4.2.2.4 S/MIME

S/MIME staat voor Secure Multipurpose Internet Mail Extensions, een uitbreiding van het al bestaande MIME protocol. S/MIME is de standaard voor elektronische bericht-uitwisseling (e-mail). S/MIME is een belangrijk hulpmiddel voor E-business, doordat het de belangrijke aspecten van vertrouwelijkheid en authenticatie van gegevens aanpakt. S/MIME gebruikt Public Key Encryptie om berichten te beschermen tegen onderschepping en vervalsing. S/MIME biedt de mogelijkheid

om gegevens veilig op te slaan, te versturen en door te sturen. Op dat gebied is S/MIME de standaard, en daar ligt ook het nut van het protocol. Waar SSL de verbinding tussen een client en een server over een onveilig netwerk beveiligt, wordt S/MIME gebruikt om veilig berichten tussen gebruikers, applicaties en computers te versturen.

4.3 Conclusie

Met betrekking tot de ontwikkeling van gedistribueerde applicaties zijn de volgende aspecten genoemd, waarin een applicatie, afhankelijk van de gewenste functionaliteit, in meer of mindere mate dient te voorzien:

- Beschikbaarheid voor geautoriseerde personen
- Privacy en integriteit
- Authenticatie en non-repudiation
- Controleerbaarheid

Om deze aspecten te bewerkstelligen zijn een aantal methoden genoemd, alle op basis van encryptie. Hiervan zijn IPSec en S/MIME het minst van toepassing. IPSec is minder geschikt voor een netwerk met de omvang van het Internet (IPSec beveiligt een heel netwerk, en het Internet is daarvoor te groot). S/MIME is voornamelijk bedoeld voor e-mail, en niet alle gedistribueerde applicaties maken daarvan gebruik. Over het algemeen kan gezegd worden dat niet één methode zonder meer de beste is, al is SSL in de praktijk wel de meest gebruikte, voornamelijk bij E-Commerce applicaties.

Gezien de vele verschillende methoden die hierboven genoemd werden, is het niet moeilijk voor te stellen dat er binnen de beveiligingswereld grote behoefte is aan standaardisatie. Een succesvolle, wereldwijde implementatie van een beveiligingsmethode is in principe mogelijk, maar er moet rekening gehouden worden met een aantal aspecten. Het eerste aspect betreft de problematiek rond de distributie (en het beheer) van sleutels. Er is nog geen goedkope standaard om sleutels veilig te distribueren, waardoor de inspanningen voor een beveiligingsbeleid voor een groot deel tenietgedaan worden. Een ander aspect is dat het label ‘100% waterdicht’ nooit gehaald kan worden, ook als de sleutelproblematiek naar behoren is opgelost. Zogenaamde *hackers* (mensen die proberen zwakheden te vinden in de beveiliging en zodoende in te breken in computersystemen) zijn vindingrijk en het zal niet makkelijk zijn deze mensen buiten te houden.

Tenslotte dient nog gezegd te worden dat de beveiliging van een gedistribueerde applicatie een complexe zaak is. Het wordt dan ook aangeraden om bij het ontwikkelen van een gedistribueerde applicatie die met beveiliging te maken zal krijgen (wat bijna altijd het geval is), een expert in de arm te nemen.

5 EISEN T.A.V. GEDISTRIBUEERDE APPLICATIEONTWIKKELING

In de voorgaande hoofdstukken zijn de functionaliteiten en gewenste eigenschappen van de uiteindelijke applicatie behandeld. Dit hoofdstuk betreft de ontwikkeling zelf van gedistribueerde applicaties. Eerst worden de aspecten van traditionele Software Engineering besproken, en geëvalueerd in hoeverre ze van toepassing zijn voor gedistribueerde applicaties. Daarna wordt gekeken naar de ontwikkeling zelf en welke eigenschappen van ontwikkeltechnieken wenselijk zijn.

5.1 Software Engineering

In zijn boek 'Software Engineering, principles and practice' ([VLIE, 1993]) geeft Van Vliet een aantal definities voor Software Engineering. De standaard definitie, die in de *IEEE Standard Glossary of Software Engineering Terminology* [IEEE, 1983] staat, luidt als volgt:

> *Software Engineering is the systematic approach to the development, operation, maintenance and retirement of software.*

Vrij vertaald is Software Engineering een systematische aanpak voor het ontwikkelingsproces en voor de uiteindelijke applicatie zelf. Men probeert standaarden voor de ontwikkeling te vinden om de kwaliteit van software te waarborgen. Op basis van bevindingen uit dit zoekproces is een aantal criteria opgesteld waar een goed ontwikkeltraject aan dient te voldoen. In deze paragraaf worden eerst de relevante Software Engineering aspecten toegelicht. Daarna wordt het belang daarvan voor gedistribueerde applicatie ontwikkeling behandeld.

5.1.1 Software Engineering aspecten

De Software Engineering eisen zijn in eerste instantie opgesteld voor *stand-alone* applicaties. Hierdoor gelden ze allemaal in meer of mindere mate ook voor gedistribueerde applicaties. Sommige aspecten zijn echter meer relevant voor gedistribueerde applicaties dan andere. Daarom worden in deze paragraaf slechts de Software Engineering aspecten behandeld die in het licht van gedistribueerde applicatieontwikkeling een andere (of extra) betekenis krijgen dan voor stand-alone applicatie ontwikkeling. Voor meer informatie over deze (en andere) Software Engineering aspecten wordt verwezen naar van Vliet ([VLIE, 1993]). Gezien het globale karakter van de gebruikte termen, wordt de engelse naamgeving voor deze aspecten gehandhaafd.

- **Reusability** zorgt dat software objecten of componenten (zie ook hoofdstuk 3) gebruikt kunnen worden als bouwstenen voor toekomstige software ontwikkeling. Dit hergebruik van

componenten bevordert de snelheid en het gemak van ontwikkeling.

- **Extensibility** verschaft de mogelijkheid om op simpele wijze functionaliteiten aan bestaande software toe te voegen.
- **Compatibility** is de eigenschap die software componenten de mogelijkheid geeft om gecombineerd te worden met andere componenten.
- **Robustness** is de eigenschap van applicaties om zelfs onder extreme omstandigheden (overmatige belasting door bijvoorbeeld veel gelijktijdig gebruik van een multi-user applicatie) normaal te functioneren.
- **Efficiency** is de eigenschap dat een applicatie niet bovenmatig beslag legt op de (schaarse) *resources* van een computer, zoals processor-belasting of geheugengebruik.
- **Portability** geeft de mogelijkheid voor een applicatie aan om in verschillende software en hardware omgevingen te draaien.

5.1.2 Software Engineering aspecten en gedistribueerde applicaties

Niet voor alle bovengenoemde aspecten is de relevantie voor gedistribueerde applicaties direct duidelijk. Daarom wordt in deze paragraaf ingegaan op de toepasbaarheid van die eisen op het ontwikkeltraject van gedistribueerde applicaties.

- **Reusability**
  Aangezien veel technieken voor de ontwikkeling van gedistribueerde applicaties nog vrij jong zijn, is reusability zeker niet triviaal. Vaak is een gestructureerde opzet van het programma bevorderlijk (misschien wel noodzakelijk) voor de reusability. Object Oriented programmeren heeft vaak een gunstige uitwerking op de gestructureerdheid van een programma.
- **Extensibility**
  Indien applicaties modulair opgesteld zijn, is het in veel gevallen vrij eenvoudig om de componenten aan te passen. Bij gedistribueerde applicaties dient echter rekening gehouden te worden met het feit dat componenten mogelijk op diverse tiers geplaatst zijn (zie hoofdstuk 3).
- **Compatibility**
  Indien componenten hergebruikt kunnen worden voor andere applicaties als ze te combineren zijn met meerdere componenten, kunnen ontwikkeltijd en ook testtijd sterk teruggebracht worden, en daarmee tegelijkertijd de kosten. De toegevoegde waarde voor gedistribueerde applicaties is, dat mogelijk één component door meerder applicaties gebruikt kan worden.
- **Robustness**

Aangezien het feit dat het nut van een gedistribueerde applicatie veelal gelegen is in het gelijktijdig gebruiken van de applicatie door meerdere gebruikers (i.e. een multi-user omgeving), is het wenselijk dat de applicatie berekend is op zware belasting door veel gelijktijdig gebruik.

- **Efficiency**
  Wat hierboven voor robustness geldt, geldt ook voor efficiency: de kans is aanwezig dat veel mensen tegelijkertijd de applicatie zullen gebruiken. Het is daarom zeer wenselijk als de applicatie efficiënt omspringt met de beschikbare systeembronnen als processortijd en geheugenruimte. Stel je voor dat de applicatie voor elke gebruiker 1 MB geheugenruimte inneemt. Indien er duizend gebruikers per uur komen, die elk een half uur gebruik maken van de applicatie, dient de machine alleen voor de applicatie dus ruim 500 MB aan intern geheugen te hebben. Dit is duidelijk geen wenselijke situatie.
- **Portability**
  Deze eis lijkt haast voor gedistribueerde applicaties te zijn geformuleerd. Indien veel mensen vanaf verschillende plaatsen (en mogelijk vanaf willekeurige plaatsen) een applicatie moeten kunnen gebruiken is het wenselijk dat die applicatie platformonafhankelijk is. Dit is geen bindende eis, maar het telt als een pluspunt. Dit geldt voornamelijk voor de interface, omdat die bij diverse clients uitgevoerd moet worden en daardoor mogelijk meerdere platforms moet ondersteunen. In mindere mate is dit van toepassing op de Application Logic en de database, omdat voor deze gedeelten eenmaal een platform gekozen wordt en daarna niet vaak verandert. Het is natuurlijk een pluspunt als ook database en Application Logic op meerdere platforms kunnen draaien. Portability behelst meer dan alleen het Operating System. Indien een applicatie namelijk een browser-interface biedt (de interface draait op de client in een browser), dient nagegaan te worden, welke browser ondersteund (moeten) worden. Ook geldt dat eventuele additionele programmatuur op de server ondersteund moet worden door de gebruikte webserver. Er zijn kortom eisen wat betreft ondersteunde Operating Systems aan zowel client als server kant, en, indien van toepassing, ondersteuning van browser aan de client kant en ondersteuning van webservers aan de server kant.

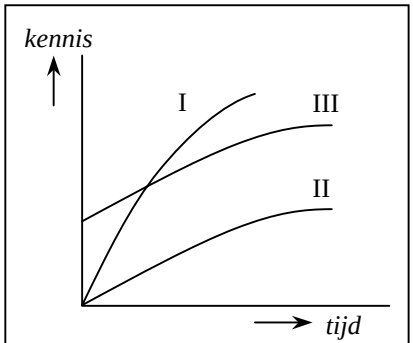
5.2 Eisen m.b.t. ontwikkeling

Aspecten die van belang zijn bij het ontwikkelen van een gedistribueerde applicatie:

- **Gebruiksgemak IDE**
  Hoe makkelijk is het om met een bepaalde ontwikkeltechniek een applicatie te bouwen? Dit wordt voor een heel groot deel bepaald

door de ontwikkelomgevingen die beschikbaar zijn voor de desbetreffende techniek. Een IDE (Integrated Development Environment) maakt het mogelijk om vanuit één ontwikkelomgeving zowel de lay-out als de code te ontwikkelen, en om de code te 'debuggen'. Hierbij sluit aan of de techniek gebonden is aan een bepaalde programmeertaal, of dat er verschillende programmeertalen voor gebruikt kunnen worden, zodat de ontwikkelaar mogelijk niet gedwongen is een nieuwe taal te leren.

- **Snelheid van ontwikkeling**
  Hoe snel kan een applicatie ontwikkeld worden met de desbetreffende techniek?
- **Prijs van ontwikkeling**
  Hoe duur is het om een applicatie te bouwen met die techniek?
- **Leercurve**
  De leercurve is een grafische representatie van de geleerde kennis, uitgezet tegen de benodigde tijd. In de figuur hiernaast zijn drie leercurven aangegeven. Deze geven twee situaties weer: een situatie waarin het vrij makkelijk is om redelijk complexe applicaties te maken met een bepaalde techniek (steile leercurve, curve I), en een situatie waarin de ontwikkelaar een hoop tijd aan een techniek moet besteden voordat er iets behoorlijks uit zijn handen komt (vlakke leercurve, curve II). Hiermee houdt verband welke kennis de ontwikkelaar al met een bepaalde techniek heeft. Als de ontwikkelaar al vaker met de techniek heeft gewerkt valt de leercurve voor hem gunstiger uit. Vanwege de voorkennis begint de curve namelijk hoger op de verticale as (curve III). Dit is echter geen eigenschap van de techniek zelf, maar eerder een omgevingsvariabele. Een techniek kan hier dan ook niet op beoordeeld worden.



Figuur 5.1: Drie leercurven

- **Ondersteuning van de techniek door derden**
  In hoeverre is er ondersteuning te verkrijgen voor het gebruik van deze techniek? Zijn er veel boeken te verkrijgen? Is er goede ondersteuning door de fabrikant? Gebruiken veel ontwikkelaars de techniek, zodat ze elkaar kunnen helpen? Indien een techniek veel gebruikt wordt, is er in het algemeen veel informatie te vinden over probleemgebieden.

5.3 Conclusie

Met dit hoofdstuk is een einde gekomen aan de beschrijving van de verschillende aspecten van gedistribueerde applicaties. Na de functionaliteitaspecten, architectuuraspecten en beveiligingsaspecten uit de voorgaande hoofdstukken, zijn in dit hoofdstuk Software Engineering aspecten en ontwikkelingsaspecten behandeld.

6 CRITERIA

Nu de belangrijkste aspecten van gedistribueerde applicaties en de ontwikkeling daarvan bekend zijn, kunnen we criteria afleiden. Op basis van deze criteria kunnen ontwikkeltechnieken beoordeeld worden op geschiktheid voor de ontwikkeling van gedistribueerde applicaties.

6.1 Afleiding

De criteria die in deze paragraaf geformuleerd worden, worden in categorieën gerangschikt op basis van dezelfde indeling als waarin ze zijn geïntroduceerd in de voorgaande hoofdstukken. Voor alle criteria wordt tevens aangegeven hoe de beoordeling van een techniek plaatsvindt. Dit kan inhouden dat een techniek wel of niet aan een criterium voldoet. Een andere mogelijkheid is dat aangegeven wordt in welke mate aan een criterium wordt voldaan. Een dergelijke beoordeling is echter vaak een subjectieve kwestie, waarbij persoonlijke voorkeuren en mogelijk configuraties van testomgevingen een belangrijke rol kunnen spelen. Omdat een te precieze beoordeling kan leiden tot een vals gevoel van objectiviteit, wordt gekozen voor een vrij grove beoordeling. Er zal een 'rapportcijfer' gegeven worden met behulp van de volgende cijfers:

Cijfer	Betekenis
1	Zeer slecht
2	Slecht
3	Redelijk
4	Goed
5	Zeer goed

6.1.1 H2 – Functionaliteiten van gedistribueerde applicaties

In hoofdstuk 2 zijn karakteristieken en voordelen van gedistribueerde applicaties naar voren gebracht. Criteria op basis van deze aspecten worden in deze paragraaf gedefinieerd.

Database communicatie
Veelal zullen gedistribueerde applicaties te maken krijgen met gegevens die in een database zijn opgeslagen (gegevens en gegevensverschaffing). Het is daarom belangrijk voor ontwikkeltechnieken om de mogelijkheid te verschaffen om connectie te maken met een database. Hoe meer verschillende soorten databases ondersteund worden, des te beter.

Beoordeling: 1-5

Distributie
Onder deze categorie vallen de mogelijke voordelen die gedistribueerde applicaties kunnen bieden boven stand-alone applicaties, zoals genoemd in paragraaf 2.2. Deze mogelijkheden zijn:

- Resource sharing
- Parallel processing
- Availability

Voor elk van deze aspecten zal afzonderlijk een beoordeling gegeven worden.

Beoordeling: 1-5

Interface
Deze categorie betreft de mogelijkheden van een techniek om de interface van de applicatie naar eigen wens aan te passen. In deze categorie vallen twee criteria:

- Biedt de techniek de mogelijkheid om de applicatie weer te geven in een browser?
- Is het met de techniek mogelijk om de applicatie een GUI (Graphical User Interface) zoals die van Windows te geven?

Het voordeel van een 'browser-interface' is dat gebruikers niet apart een programma hoeven te downloaden om de applicatie te gebruiken. De browser regelt, aan de hand van de webpagina, alles wat moet gebeuren zonder dat de gebruiker hier extra handelingen voor hoeft te doen. Een voordeel van een applicatie die niet in een browser wordt weergegeven, is dat de browser niet opgestart hoeft te worden, wat een positieve invloed heeft op de laadtijd van de applicatie (zie paragraaf 2.4). Ook is men niet afhankelijk van een bepaald type of versie browser. Voor beide criteria geldt dezelfde beoordeling:

Beoordeling: ja/nee

Prestaties:
Als laatste categorie van hoofdstuk 2 werd het prestatieaspect genoemd. Dit kwam neer op de snelheid van de applicatie, en viel uiteen in drie criteria:

- Laadtijd
- Communicatietijd
- Totale executietijd

Hierbij moet opgemerkt worden dat laadtijd en totale executietijd afhankelijk zijn van de gekozen techniek, maar dat de communicatietijd voornamelijk afhangt van de gekozen communicatiearchitectuur (zie paragraaf 3.3.3 en paragraaf 6.1.2). Communicatietijd wordt daarom niet expliciet als criterium opgenomen. Voor laadtijd en totale executietijd

geldt de volgende beoordeling, waarbij 1 een hoge tijd aangeeft en 5 een lage:

Beoordeling: 1-5

6.1.2 H3 – Architecturen

Uit hoofdstuk 3 blijkt dat bij gedistribueerde applicaties de mogelijkheid moeten bestaan om de applicatie over meerdere machines te verdelen. Een belangrijke keuze hierbij is hoe de communicatie tussen de componenten plaats moet vinden. Hieruit volgt de onderstaande categorie:

Communicatie architectuur
De communicatie kan plaatsvinden volgens een aantal verschillende protocollen, zoals in hoofdstuk 3 reeds werd vermeld. Elk van die protocollen heeft zijn eigen voor- en nadelen. De mogelijkheden zijn:

- TCP/IP
- HTTP
- Middleware

Een techniek kan meerdere mogelijkheden ondersteunen.

Beoordeling: ja/nee

6.1.3 H4 – Beveiliging

Beveiliging is een aspect dat door de ontwikkelaar zelf moet worden geregeld. In hoofdstuk 4 werden de verschillende onderdelen van beveiliging uitgebreid behandeld, evenals de verschillende methoden om een applicatie te beveiligen. De volgende categorie bevat criteria die daarop gebaseerd zijn:

Beveiligingsaspecten
De vier categorieën die in hoofdstuk 4 zijn behandeld kunnen in principe los van elkaar geïmplementeerd worden. Maar voor een optimale oplossing dienen ze alle vier toegepast te worden. Elke techniek wordt beoordeeld op de mogelijkheid voor ondersteuning van:

- Beschikbaarheid voor geautoriseerde personen
- Privacy en integriteit
- Authenticatie en non-repudiation
- Controleerbaarheid

Voor elk criterium geldt dat een techniek het wel of niet ondersteunt:

Beoordeling: ja/nee

6.1.4 H5 – Ontwikkeleisen

In hoofdstuk 5 werden de overige aspecten ingedeeld in 3 categorieën. Dezelfde indeling wordt ook hier aangehouden. De criteria worden zodoende onderverdeeld in de categorieën Software Engineering criteria, ontwikkelingscriteria en prestatiecriteria.

Software Engineering
In hoofdstuk 5 werd een aantal aspecten van Software Engineering genoemd. Ook werd daarbij aangegeven welke betekenis deze aspecten hebben voor gedistribueerde applicatie ontwikkeling. Deze Software Engineering aspecten zijn:

- Reusability
- Extensibility
- Compatibility
- Robustness
- Efficiency
- Portability

Reusability en extensibility liggen redelijk dicht bij elkaar. Beide aspecten zijn gebaat bij de mogelijkheid om een applicatie modulair op te zetten. Dit houdt in dat een applicatie in delen kan worden opgebouwd, waarbij elk deel een bepaalde (deel-)taak uit kan voeren. Reusability en extensibility kunnen daarom samengenomen worden en weergegeven worden door het criterium *modulariteit*.

Portability is een ruim begrip. Het is daarom duidelijker om de gebieden waarop een applicatie 'portable' moet kunnen zijn, af te bakenen. Hieronder staan de belangrijkste aspecten waarop de portability mogelijkheden van een techniek getoetst kunnen worden:

Ondersteuning Operating System server kant
Welke platforms worden door de techniek op de server ondersteund. Opties hierbij zijn:

- Windows 95/98
- Windows NT
- UNIX

Ondersteuning Operating System client kant
Idem, maar dan op de client. Hiervoor bestaan dezelfde opties.

Indien de applicatie gebruik maakt van het World Wide Web en een browser interface zijn ook de volgende criteria van toepassing:

Ondersteuning webserver

Op welke webservers kan de applicatie aan de server kant draaien. Opties hierbij zijn:

- Microsoft Internet Information Server
- Apache
- Netscape
- JavaSoft

Ondersteuning browser
Welke browsers worden door de techniek ondersteund. Mogelijkheden hierbij zijn:

- Microsoft Internet Explorer
- Netscape Navigator

De beoordeling van de bovengenoemde aspecten gebeurt op twee manieren. Voor de aspecten *modulariteit, compatibility, robustness* en *efficiency* geldt dat een score tussen 1 en 5 gegeven zal worden. Voor de verschillende onderdelen van *portability* geldt dat een techniek elk onderdeel wel of niet ondersteunt.

Modulariteit, compatibility, robustness, efficiency:
Beoordeling: 1-5
Portability onderdelen:
Beoordeling: ja/nee

Ontwikkeling
Naast de Software Engineering aspecten werden ook ontwikkelingsaspecten genoemd in hoofdstuk 5. Deze aspecten zijn:

- Gebruiksgemak IDE
- Snelheid van ontwikkeling
- Prijs van ontwikkeling
- Leercurve
- Ondersteuning van de techniek door derden

Bij dit laatste aspect kan nog een opmerking geplaatst worden. Hoe meer de techniek gebruikt wordt, hoe meer mensen er veel vanaf weten, dus hoe meer punten er zijn om eventuele vragen op te richten. Ook zullen er meer boeken over te vinden zijn indien meer mensen de techniek gebruiken. Dit aspect kan dus beter **algemene ondersteuning** genoemd worden.

Voor alle ontwikkelingaspecten geldt de volgende beoordelingswijze:

Beoordeling: 1-5

6.1.5 Scorekaart

De hierboven afgeleide criteria kunnen (in hun respectievelijke categorieën) weergegeven worden in de vorm van een scorekaart. Deze is hieronder afgebeeld. Voor elk criterium staat tevens aangegeven op welke wijze de beoordeling van een techniek plaats zal vinden.

Hoofdstuk	Categorie	Criterium		Beoordeling
2	*Database communicatie*	Database communicatie		1-5
2	*Distributie*	Resource sharing		1-5
		Parallel processing		1-5
		Availability		1-5
2	*Interface*	Browser interface		Ja/nee
		Graphical User Interface		Ja/nee
2	*Prestaties*	Laadtijd		1-5
		Totale executietijd		1-5
3	*Communicatie architectuur*	TCP/IP		Ja/nee
		HTTP		Ja/nee
		Middleware		Ja/nee
4	*Beveiligingsaspecten*	Beschikbaarheid geautoriseerde personen		Ja/nee
		Privacy en integriteit		Ja/nee
		Authenticatie en non-repudiation		Ja/nee
		Controleerbaarheid		Ja/nee
5	*Software Engineering*	Modulariteit		1-5
		Compatibility		1-5
		Robustness		1-5
		Efficiency		1-5
		Portability:		
		OS Server:	Windows 95/98	Ja/nee
			Windows NT	Ja/nee
			UNIX	Ja/nee
		OS Client:	Windows 95/98	Ja/nee
			Windows NT	Ja/nee
			UNIX	Ja/nee
		Webserver:	Internet Information Server	Ja/nee
			Apache	Ja/nee
			Netscape	Ja/nee
			JavaSoft	Ja/nee
		Browser	MS Internet Explorer	Ja/nee
			Netscape Navigator	Ja/nee
5	*Ontwikkeling*	Gebruiksgemak IDE		1-5
		Snelheid van ontwikkeling		1-5
		Prijs van ontwikkeling		1-5
		Leercurve		1-5
		Algemene ondersteuning		1-5

6.2 Conclusie

In dit hoofdstuk is bovenstaande scorekaart afgeleid uit de verschillende aspecten van gedistribueerde applicaties die in voorgaande hoofdstukken zijn beschreven. Hiermee is de eerste deelvraag van de probleemstelling behandeld. Deze scorekaart wordt in het volgende hoofdstuk gebruikt voor een evaluatie van een aantal bestaande technieken. Daarbij wordt ingegaan op de tweede deelvraag van de probleemstelling. Een vervelend aspect van de afgeleide criteria, is dat de categorieën niet onafhankelijk van elkaar zijn. Een beslissing in de ene categorie beïnvloedt de keuzemogelijkheden in een andere categorie. Hierop wordt in hoofdstuk 8 teruggekomen.

7 BEOORDELING

In dit hoofdstuk wordt ingegaan op de tweede deelvraag van de probleemstelling door een aantal ontwikkeltechnieken daadwerkelijk te beoordelen. Dit gebeurt aan de hand van de criteria die in het vorige hoofdstuk zijn afgeleid en gedefinieerd. In paragraaf 7.1 worden twee categorieën van technieken behandeld, namelijk systeemtalen en scripting-technieken, en wordt het begrip *mobiliteit* geïntroduceerd. In paragraaf 7.2 worden de technieken daadwerkelijk beoordeeld op basis van de criteria.

7.1 Technieken

De technieken ten behoeve van gedistribueerde softwareontwikkeling die behandeld worden, vallen uiteen in twee categorieën: *systeemtalen* en *scripting-technieken*. Deze twee categorieën worden hieronder toegelicht, waarna voor elke categorie de bijbehorende technieken genoemd worden. Daarna wordt onderscheid gemaakt worden tussen mobiele en statische code. Dit onderscheid is nuttig om de werking van de technieken in de twee categorieën te begrijpen.

7.1.1 Systeemtalen

Systeemtalen werden geïntroduceerd als een alternatief voor assembly talen. In assembly talen diende elke stap die de machine uit moest voeren expliciet geprogrammeerd te worden. Allocatie van geheugen en het vastleggen van de volgorde van stappen waren daarbij essentieel. Hierdoor was het moeilijk om grote stukken code te schrijven en te onderhouden. Tegen het einde van de jaren ‘50 kwamen er programmeertalen die op een hoger niveau werkten, zoals Lisp, Fortran en Algol. In deze talen komt een instructie in de code niet rechtstreeks meer overeen met een instructie die de machine uitvoerde. Een zogenaamde *compiler* vertaalt elke instructie in de code naar een reeks van binaire instructies voor de computer. Continu worden talen verder ontwikkeld en nieuwe talen ontworpen. Onder de hedendaagse systeemtalen bevinden zich bijvoorbeeld Pascal, C++ en Java. Deze talen zijn in theorie minder efficiënt dan assembly talen, maar veel gemakkelijker om applicaties mee te ontwikkelen en te onderhouden.

Er moet nog even stilgestaan worden bij de speciale plaats die Java inneemt in deze categorie: Java is namelijk platform-onafhankelijk. Dit komt door een tussenstap: Java wordt niet meteen gecompileerd naar een reeks binaire instructies, maar eerst naar bytecode, dat door elke computer begrepen kan worden (mits daarvoor een programma is geïnstalleerd: de Java Virtual Machine). Vervolgens wordt deze bytecode door de Java Virtual Machine (die in alle moderne browsers aanwezig is) geïnterpreteerd tot een reeks binaire instructies, waarna de machine die uit kan voeren. Door deze tussenstap is de executiesnelheid van Java wel

significant lager dan voor de andere systeemtalen. Indien echter gebruik wordt gemaakt van een Just-In-Time (JIT) compiler, kan de executiesnelheid van Java sterk verbeterd worden. Een JIT-compiler 'vertaalt' stukken van de bytecode die veel keren uitgevoerd moeten worden naar platform-specifieke binaire instructies. Dit gebeurt vlak voordat het programma uitgevoerd wordt. Er zijn ook compilers in ontwikkeling die de bytecode van Java op voorhand compileren naar binaire instructies voor specifieke platforms. Hierdoor gaat echter de platform-onafhankelijkheid van Java verloren. Dit is daarom alleen bruikbaar indien het platform waarop de code uitgevoerd moet worden van tevoren bekend is, zoals meestal voor bijvoorbeeld de Application Logic het geval is.

Een voorbeeld van een stuk code waarbij een schermpje getoond wordt met als titel "My first active document" en de tekst "Hello, world!" staat hieronder. Het voorbeeld is gemaakt in Delphi, een ontwikkelomgeving die gebruik maakt van de taal Pascal. In Delphi zijn veel voorgedefinieerde onderdelen aanwezig, zodat die niet opnieuw geprogrammeerd hoeven te worden. Zo kan er in een applicatie een knop (zeg dat die "BtnHello" heet) gedefinieerd worden, waarvoor de volgende code wordt uitgevoerd als erop wordt geklikt:

```
procedure BtnHelloClick(Sender: TObject);
begin
  ShowMessage('Hello, World', 'My first active document', 0);
end;
```

Hiernaast dient elders voor de knop andere eigenschappen gedefinieerd te worden, zoals positie op het scherm, grootte en tekst die erop staat.

De technieken die onder de systeemtalen vallen zijn:

- Object Pascal (Delphi)
- Java
- C++
- Visual Basic

Bij de beoordeling hiervan (zie paragraaf 7.2.1) wordt voor elke techniek kort een toelichting gegeven.

7.1.2 Scripting-technieken

Een script is een stuk code (tekst) dat niet direct uitgevoerd wordt door de hoofdprocessor van de PC (zoals bij systeemtalen het geval is), maar door een ander programma, bijvoorbeeld een browser. Voor het schrijven van zo'n script zijn er verschillende talen (scripting talen) beschikbaar.

Voorbeelden zijn JavaScript, VBScript en Perl. Scripting talen gaan er van uit dat er al een verzameling bruikbare, direct executeerbare onderdelen bestaan die zijn geschreven in andere programmeertalen. Scripting talen zijn niet bedoeld om nieuwe applicaties van de grond af aan op te bouwen; ze zijn in eerste instantie bedoeld voor het samen lijmen van onderdelen, niet voor ingewikkelde algoritmes. Die ingewikkelde algoritmes zitten meestal bevat in de onderdelen.

Scripting talen zijn bedoeld om makkelijk en snel stukken code te schrijven. Zo hoeft bijvoorbeeld niet expliciet gedefinieerd te worden van welk type een variabele is (een variabele is een symbolische naam voor een stukje geheugenruimte waarin een waarde opgeslagen kan worden). Voor een variabele kan opgegeven worden welk type het is: een getal, een letter, etc. In een systeemtaal dient voor elke variabele expliciet gedefinieerd te worden welk type het is voordat er een waarde aan toegekend kan worden. De variabele kan dan ook niet gebruikt worden voor andere types. Bij scripting talen hoeft dit niet. Zo kan een variabele het ene moment een rentepercentage bevatten, en het andere moment de naam van een klant. Indien het type expliciet opgegeven moet worden heet dit *strongly typed*, indien dit niet nodig is hebben we het over *typeless* variabelen. Doordat scripting typeless variabelen gebruikt, is de ontwikkelingssnelheid van scripting talen groter dan van een systeemtaal. Maar omdat scripts uitgevoerd worden met behulp van een ander programma en niet direct door de hoofdprocessor, worden er extra instructies uitgevoerd waardoor de executiesnelheid van scripting talen lager ligt dan bij systeemtalen. Dit wordt nog eens versterkt door het feit dat variabelen typeless zijn, waardoor at runtime checks moeten worden uitgevoerd om te bepalen welk type een variabele is. Hierdoor wordt een taak sneller uitgevoerd door een applicatie geschreven in een systeemtaal dan in een scripting taal. Dit is ook de reden waarom scripts niet gebruikt worden voor complexe algoritmes.

Een voorbeeld van een script is hieronder te vinden. Het voorbeeld maakt gebruik van HTML (HyperText Markup Language; de taal waarmee webpagina's opgemaakt worden) en VBScript omdat dat makkelijk te gebruiken is. Allereerst wordt er met HTML aangegeven dat er een knop op het scherm getoond moet worden:

```
<INPUT TYPE=BUTTON VALUE="Click me" NAME="BtnHello">
```

`INPUT TYPE=BUTTON` geeft aan dat wat er staat een knop (een button) is. `VALUE="Click me"` geeft aan dat op de knop de tekst "Click Me" staat. `NAME="BtnHello"` geeft aan dat de knop de naam "BtnHello" heeft.

Vervolgens kan met VBScript een functie gemaakt worden die wordt uitgevoerd als iemand op de knop klikt:

```
<SCRIPT LANGUAGE="VBScript">
<!--
 Sub BtnHello_OnClick
  MsgBox "Hello, world!", 0, "My first active document"
 End Sub
-->
</SCRIPT>
```

Eerst wordt aangegeven dat de code VBScript is (`<SCRIPT LANGUAGE="VBScript">`). Dan wordt de functie (in VBScript de 'Sub') BtnHello_OnClick gedefinieerd. Als er op de knop wordt geklikt, wordt de code `MsgBox "Hello, world!", 0, "My first active document"` uitgevoerd, waardoor er een schermpje in beeld komt met als titel de tekst "My first active document" en als tekst "Hello, world!". Om dit script uit te proberen in een webbrowser dient een tekstbestand gemaakt te worden (dat bijvoorbeeld "hello.html" heet) met daarin de vorige twee stukken tekst onder elkaar.

Scripting komt in twee smaken: client-side scripting en server-side scripting. Het verschil berust op waar het script wordt uitgevoerd: op de client of op de server. Het voorbeeld hierboven wordt op de client uitgevoerd, en is dus een voorbeeld van client-side scripting. Veel scripting talen kunnen zowel op de client als op de server uitgevoerd worden.

Scripting-technieken maken gebruik van scripting-talen die op de server uitgevoerd worden, om zodoende resultaten van een bewerking naar de client op te sturen. De resultaten zijn in de vorm van HTML (HyperText Markup Language), zodat alle browsers dit kunnen begrijpen. De client krijgt alleen de resultaten van een bewerking toegestuurd, dus er wordt geen informatie vrijgegeven over hoe de bewerking precies in zijn werk gaat.

De scripting-technieken die in paragraaf 7.2.2 behandeld worden zijn:

- CGI (Common Gateway Interface)
- ASP (Active Server Pages)
- JSP (Java Server Pages)
- ColdFusion

Voorafgaand aan de daadwerkelijke beoordeling wordt voor elke techniek een korte toelichting gegeven.

7.1.3 Mobiliteit

De mobiliteit van code betreft de plaats waar de code uitgevoerd wordt. Bij statische code wordt de code uitgevoerd op de plaats waar hij is opgeslagen. Mobiele code wordt eerst naar een andere machine getransporteerd voordat de code aldaar wordt uitgevoerd. Meestal is dit de machine van de gebruiker. Het verband tussen de eerdergenoemde technieken en respectievelijk statische en mobiele code wordt hieronder kort toegelicht.

7.1.3.1 Statische code

Statische code is bij systeemtalen mogelijk indien de gebruiker de Interface kan downloaden en daarna op zijn computer kan laten staan. De gebruiker kan dan direct de Interface opstarten zonder tussenkomst van een browser. Applicaties die gemaakt zijn met een scripting-techniek bestaan voor het grootste deel uit statische code. De stukken scripts die op de server uitgevoerd worden, blijven op de server en zijn zodoende statische code (server-side scripting).

7.1.3.2 Mobiele code

Mobiele code is voor systeemtalen de enige mogelijkheid om gebruik te maken van een browser interface. Echter alleen Java biedt hiervoor direct ondersteuning in de vorm van ‘applets’. De andere systeemtalen moeten hiervoor gebruik maken van ActiveX. Beide principes worden verderop toegelicht. Voor scripting-technieken is het mogelijk om stukken scripts mee te sturen die op de machine van de gebruiker, binnen de browser, uitgevoerd worden (client-side scripting). Mobiele code biedt de mogelijkheid om de Interface wat meer geavanceerd te maken en staat de gebruiker een zekere interactie toe met de Interface. Bij scripting-technieken is deze interactie vrij beperkt, bij systeemtalen kan deze echter alle functionaliteit aannemen van een ‘gewone’ Interface, zoals *drag-and-drop*.

ActiveX

ActiveX is een Microsoft technologie die het gebruik van mobiele code mogelijk maakt voor andere systeemtalen dan Java. Het wordt voornamelijk voor gedistribueerde applicaties gebruikt. De technologie gaat uit van een modulaire opbouw van de zogenaamde ‘ActiveX componenten’, die een zelfde functionaliteit bieden als de JavaBeans die in bijlage B worden beschreven. Door deze modulaire opbouw kunnen de componenten hun taken uitvoeren binnen elke ‘container’ (programma dat een ActiveX component kan uitvoeren) die dat ondersteunt. Microsoft’s Internet Explorer is zo’n container, net zoals Microsoft Word. Hierdoor kan een

ActiveX component via Internet naar de client getransporteerd worden, waarna het binnen de browser zijn taken uitvoert. De ActiveX technologie biedt de mogelijkheid om met Middleware samen te werken. Ook kunnen ActiveX componenten gecreëerd en gebruikt worden door één van de vele talen zoals C++ of Visual Basic, of door scripting tools als VBScript.

Het uitvoeren van een ActiveX component binnen een browser biedt het voordeel dat niets geïnstalleerd hoeft te worden op de client. De browser controleert automatisch of er een nieuwe versie beschikbaar is, en als dat zo is wordt die nieuwere versie gebruikt. Dit maakt versiebeheer een stuk minder bewerkelijk. ActiveX biedt echter geen browser-interface, maar een Graphical User Interface in de stijl van Windows.

In combinatie met een browser is ActiveX op dit moment alleen nog maar te gebruiken op het Windows Operating System en alleen met Microsoft Internet Explorer. Er zijn echter zogenaamde ‘plug-ins’ beschikbaar voor Netscape Navigator. Dit zijn kleine programma’s waarmee ActiveX componenten ook binnen Navigator uitgevoerd kunnen worden. Dit werkt helaas nog niet zo goed als bij Internet Explorer. Microsoft heeft echter de ontwikkeling van ActiveX overgedragen, waardoor het als open standaard ontwikkeld kan worden voor meerdere Operating Systems. ActiveX wordt al ontwikkeld voor UNIX en Netscape is van plan ondersteuning voor ActiveX in te bouwen in toekomstige browsers.

Een probleem van het gebruik van ActiveX in een browser is de veiligheid. Programma’s die op deze manier uitgevoerd worden, hebben namelijk dezelfde bevoegdheden als normale programma’s. Dit betekent dat ze gegevens van de harde schijf kunnen lezen, gegevens uit het register halen, bestanden verwijderen en zelfs harde schijven formatteren. Het is daarom aan te raden om voorzichtig te zijn met ActiveX programma’s. ActiveX componenten kunnen geregistreerd worden bij instanties, waardoor de componenten ‘signed’ worden. In de browser kunnen veiligheidsinstellingen zó geconfigureerd worden, dat alleen ‘gesignde’ ActiveX componenten uitgevoerd mogen worden. Dit neemt echter niet weg dat de componenten nog steeds de bovengenoemde bevoegdheden hebben. Er zal dus vertrouwd moeten worden op de makers van het programma.

Java applets

Java biedt de mogelijkheid om zogenaamde applets te maken. Deze applets zijn (stukken) applicaties die binnen een browser uitgevoerd kunnen worden. In elke ‘Java-enabled’ browser is een Java Virtual Machine (zie paragraaf 7.1.1) ingebouwd die de applet vertaalt naar

binaire instructies voor het platform waarop de browser uitgevoerd wordt. Deze applets zijn dan (op dezelfde wijze als ActiveX componenten) zichtbaar binnen de browser. De platform-onafhankelijkheid van Java is (uiteraard) ook van toepassing op applets, evenals de vertraging die optreedt als de Java Virtual Machine de bytecode interpreteert.

Aan de beveiliging van Java applets is veel aandacht besteed. Het risico dat een applet bewerkingen zou uitvoeren die de computer of gegevens van de gebruiker zouden beschadigen (zoals bij ActiveX mogelijk is), is ondervangen door in de browsers een zogenaamde *Sandbox* in te bouwen. Zo beperkt de Sandbox de mogelijkheden van de applet met betrekking tot het inlezen en wegschrijven van gegevens van en naar de harde schijf. Deze beperkingen tasten wel de mogelijke functionaliteit van een applet aan. Het is echter mogelijk voor de gebruiker om een applet meer rechten te geven dan de applet standaard van de Sandbox krijgt, waardoor dit mogelijke probleem omzeild kan worden. Het gebruik van applets wordt dan wel meer een vertrouwenskwestie, zoals het geval is bij ActiveX componenten. Zie voor meer informatie [KILS, 1999].

7.2 Beoordeling

In deze paragraaf worden de technieken uit de vorige paragraaf beoordeeld op basis van de criteria die in hoofdstuk 6 zijn opgesteld. Dit gebeurt aan de hand van de scorekaart die in hetzelfde hoofdstuk werd gepresenteerd. Om de wijze van beoordeling te verduidelijken wordt de beoordeling van één techniek beschreven, namelijk van Object Pascal. De beoordeling van de overige technieken vindt op analoge wijze plaats.

7.2.1 Systeemtalen

De systeemtalen worden in deze paragraaf eerst toegelicht alvorens beoordeeld te worden met de scorekaart uit hoofdstuk 6.

7.2.1.1 Object Pascal (Delphi)

Deze taal biedt alle mogelijkheden van een geavanceerde programmeertaal, zoals ‘Object Oriented’ programmeren (zie paragraaf 3.1.1). Object Pascal is nog steeds in ontwikkeling, en veroudert daarom niet snel. Object Pascal wordt voornamelijk gebruikt in de ontwikkelomgeving Delphi en wordt daarom ook geëvalueerd in combinatie met deze ontwikkelomgeving. Delphi is een uiterst gebruiksvriendelijke IDE en biedt uitstekende ondersteuning voor ActiveX en middleware. Object Pascal wordt volledig gecompileerd en is dus niet platformonafhankelijk.

Beoordeling:

Object Pascal kan met een grote verzameling verschillende databases communiceren. De fabrikant van Object Pascal (Delphi), Inprise, levert zijn eigen protocol om met databases te praten, de Borland Database Engine (BDE). Deze zorgt voor een universele manier om met databases van verschillende fabrikanten te communiceren. De prestaties hiervan zijn goed. Object Pascal ondersteunt het gebruik van ODBC-compliant databases. Het ODBC-protocol (Open DataBase Connectivity) is een standaard manier om met veel verschillende databases te communiceren. Met Object Pascal kunnen heel gemakkelijk verschillende bewerkingen op gegevens uit databases gedaan worden, zoals wijzigen en bewerken van gegevens. De score van Object Pascal op het criterium database communicatie is zodoende 'zeer goed'.

Met de verschillende aspecten in de categorie *Distributie* heeft de auteur geen praktische ervaring. Daarom is het moeilijk een goede beoordeling te geven. De ondersteuning van middleware is echter belangrijk voor de beoordeling in de categorie *Distributie*, omdat middleware aspecten als *parallel processing* sterk vergemakkelijkt. De beoordeling van deze criteria is dan ook afhankelijk van de kwaliteit van de beschikbare vormen van middleware. Gezien het huidige niveau van middleware scoort Object Pascal een voorzichtige 'goed' in de categorie *Distributie*. Zonder middleware is de implementatie van aspecten als parallel processing erg moeilijk. Daarom wordt sterk aangeraden middleware te gebruiken, en de score is daar dan ook aan gerelateerd.

Object Pascal biedt een grafische interface (GUI). Met behulp van ActiveX kan Object Pascal ook gebruikt worden binnen een browser, en biedt zodoende ook een 'browser-interface'.

De laadtijd van Object Pascal is afhankelijk van de grootte van de Interface. De Interface moet van de server naar de client getransporteerd worden, en de snelheid waarmee dit gebeurt is, gezien de huidige snelheid van het Internet, zeker een beperkende factor. Een voordeel is echter dat, zolang er geen nieuwe versie op de server aanwezig is, de Interface slechts eenmalig gedownload hoeft te worden. De executiesnelheid van Object Pascal is goed, zij het niet zo goed als die van C++.

Object Pascal kan via het TCP/IP protocol en via middleware met andere componenten communiceren.

De verschillende aspecten van beveiliging zijn met Object Pascal goed te verzorgen. Object Pascal ondersteunt het gebruik van SSL en heeft

daardoor mogelijkheden voor authenticatie, autorisatie en encryptie. Naast SSL kan Object Pascal ook andere vormen van beveiliging gebruiken. Het maken van log-files is met Object Pascal geen probleem.

Object Pascal biedt, zoals de naam al aangeeft, mogelijkheden om 'Object Oriented' te programmeren. Daarnaast kan men zogenaamde dll's maken met Object Pascal. Dll's (Dynamic Link Libraries) zijn kleine programma's die door andere programma's aangeroepen kunnen worden (een praktische uitwerking van componenten zoals die in hoofdstuk 3 zijn beschreven). De functionaliteit van een dll kan door meerdere applicaties tegelijk gebruikt worden. Met Object Pascal kan men zelf dll's maken of dll's die in andere talen zijn geschreven, zoals C++, aanroepen. De robuustheid van Object Pascal is vrij goed, mede omdat ontwikkelaars zelden expliciet gebruik hoeven te maken van 'pointers' (directe verwijzingen naar geheugenlocaties). Tevens zijn gebruikte systeembronnen makkelijk te beheren met Object Pascal. Deze overwegingen leiden tot een 4 op de punten 'modulariteit', 'compatibility', 'robustness' en 'efficiency'.

Op dit moment kunnen applicaties die met Object Pascal zijn gemaakt alleen nog maar gecompileerd worden voor Windows platforms (Windows 95/98 en Windows NT). Delphi versies voor UNIX zijn (nog) niet beschikbaar. De ontwikkelaars van Delphi, Inprise, hebben wel aangekondigd Delphi voor Linux (een Operating System dat afgeleid is van UNIX) te gaan maken, maar op het moment van schrijven is dat nog niet beschikbaar. Voor meer informatie hierover wordt verwezen naar [BORL, 2000]. Indien Delphi beschikbaar wordt voor meerdere platforms, moet wel rekening worden gehouden met specifieke code die op het ene platform wel werkt en op het andere niet.

Applicaties die met Object Pascal zijn gemaakt maken geen gebruik van een webserver. Indien een Object Pascal applicatie gebruik maakt van ActiveX is de ondersteuning van Netscape Navigator zo slecht dat deze geen optie is. Zodoende wordt ActiveX alleen ondersteund door Microsoft Internet Explorer.

Wat betreft de ontwikkeling is Delphi de meest gebruiksvriendelijke IDE waarmee de auteur heeft gewerkt. De snelheid van ontwikkeling ligt op een goed niveau, en de prijs van ontwikkeling is afhankelijk van de benodigde 'componenten' (zie opmerking 7 hieronder). Een hoop componenten zijn gratis te gebruiken, en dit is goed voor de kosten. Met Delphi is na een kort leertraject al een vrij geavanceerde applicatie te maken. Tenslotte zijn Object Pascal en Delphi ruim vertegenwoordigd in een hoop boeken en op het Internet.

De scores die hierboven zijn beargumenteerd zijn hieronder weergegeven in de scorekaart.

Object Pascal (Delphi)				
Categorie	**Criterium**		**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie		5	
Distributie	Resource sharing		4	
	Parallel processing		4	
	Availability		4	
Interface	Browser interface		Ja	1)
	Graphical User Interface		Ja	
Prestaties	Laadtijd		3	2)
	Totale executietijd		4	3)
Communicatie architectuur	TCP/IP		Ja	
	HTTP		Nee	
	Middleware		Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen		Ja	4)
	Privacy en integriteit		Ja	
	Authenticatie en non-repudiation		Ja	
	Controleerbaarheid		Ja	
Software Engineering	Modulariteit		4	
	Compatibility		4	
	Robustness		4	
	Efficiency		4	
	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Nee	5)
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Nee	5)
	Webserver:	Internet Information Server	NVT	
		Apache	NVT	
		Netscape	NVT	
		JavaSoft	NVT	
	Browser	MS Internet Explorer	Ja	6)
		Netscape Navigator	Nee	6)
Ontwikkeling	Gebruiksgemak IDE		5	
	Snelheid van ontwikkeling		4	7)
	Prijs van ontwikkeling		4	7)
	Leercurve		4	
	Algemene ondersteuning		4	8)

Opmerkingen:

1) ActiveX biedt Object Pascal de mogelijkheid om binnen een browser uitgevoerd te worden.

2) Dit is voornamelijk van toepassing indien gebruik wordt gemaakt van een ActiveX component. Het duurt enige tijd om een dergelijk component van de server naar de client te vervoeren (afhankelijk van de grootte), maar als dat eenmaal gebeurd is, hoeft dit niet meer gedaan te worden totdat een nieuwe versie op de server geplaatst wordt.
3) De executiesnelheid is goed. Object Pascal is niet de snelste systeemtaal, maar de snelheid is goed genoeg voor vrijwel alle doeleinden.
4) Object Pascal kan in samenwerking met SSL gebruikt worden, maar kan ook een andere methode van encryptie gebruiken.
5) Er zijn (nog) geen compilers die Object Pascal naar UNIX-code kunnen compileren.
6) Met ActiveX kan een Object Pascal programma binnen Internet Explorer uitgevoerd worden. De ondersteuning van Netscape Navigator is nog te slecht om hier 'ja' te kunnen zeggen.
7) Standaard zitten bij Delphi veel kant-en-klare 'componenten'. Hier wordt niet het begrip 'component' uit hoofdstuk 3 mee bedoeld. Het betreft hier 'stukjes programma' die heel snel aan elkaar gekoppeld kunnen worden om zo een werkend geheel te krijgen. Andere van zulke 'componenten' zijn vaak voor weinig geld te koop.
8) Er zijn veel boeken voor Object Pascal en Delphi, veel gebruikersgroepen, en ook een hoop websites bieden antwoorden op vragen van gebruikers.

7.2.1.2 Java

Bij de ontwikkeling van de programmeertaal Java is rekening gehouden met het eventuele gebruik ervan in de gedistribueerde omgeving van het Internet. Het moest lijken op C++ maar minder ingewikkeld zijn. Java ondersteunt een volledig object georiënteerde programmeer aanpak. Het kan gebruikt worden om volledige applicaties te maken die op een enkele computer draaien of verspreid zijn over meerdere servers en client in een netwerk. Ook kan Java gebruikt worden om kleine programmaatjes, applets, te schrijven, die gebruikt kunnen worden op een web pagina.

Java programma's zijn 'portable'. Dat houdt in dat het programma wordt gecompileerd tot Java bytecode, dat overal op het netwerk kan worden uitgevoerd op een server of een client die een Java Virtual Machine (JVM) heeft. Een JVM interpreteert de bytecode tot code die kan draaien op de hardware van de desbetreffende computer. Dit betekent dat een programma niet voor verschillende platforms geschreven hoeft te worden. Sun Microsystems, de ontwikkelaar van Java, hebben dit principe Write Once, Run Anywhere (WORA) gedoopt.

Daarnaast is de code ‘robuust’. Dit houdt in dat, in tegenstelling tot programma’s die bijvoorbeeld in C++ geschreven zijn, Java objecten geen verwijzingen naar gegevens buiten zichzelf kunnen hebben (de pointers van C++, zie verderop). Hierdoor is een applicatie geschreven in Java veel minder ‘crash-gevoelig’, dus een applicatie loopt minder kans om ‘vast te lopen’. Naast deze verschillen lijkt Java veel op C++. Het is wel wat makkelijker om te leren, maar het is niet de makkelijkste programmeertaal.

Java biedt de mogelijkheid om zogeheten JavaBeans te schrijven. Dit zijn stukken code die zelf een taak kunnen uitvoeren (dit is Java-terminology voor de componenten zoals die in paragraaf 3.1.2 zijn beschreven).

Beoordeling:

Java				
Categorie	**Criterium**		**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie		4	
Distributie	Resource sharing		4	1)
	Parallel processing		4	1) 2)
	Availability		4	1)
Interface	Browser interface		Ja	
	Graphical User Interface		Ja	
Prestaties	Laadtijd		2	3)
	Totale executietijd		3	4)
Communicatie architectuur	TCP/IP		Ja	
	HTTP		Nee	
	Middleware		Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen		Ja	
	Privacy en integriteit		Ja	
	Authenticatie en non-repudiation		Ja	
	Controleerbaarheid		Ja	5)
Software Engineering	Modulariteit		5	6)
	Compatibility		3	7)
	Robustness		4	4)
	Efficiency		4	4)
	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	

	Webserver:	Internet Information Server	NVT	
		Apache	NVT	
		Netscape	NVT	
		JavaSoft	NVT	
	Browser	MS Internet Explorer	Ja	
		Netscape Navigator	Ja	
Ontwikkeling	Gebruiksgemak IDE		3	
	Snelheid van ontwikkeling		3	
	Prijs van ontwikkeling		4	
	Leercurve		4	
	Algemene ondersteuning		5	8)

Opmerkingen:

1) Indien Java gebruikt wordt in combinatie met middleware (RMI) en speciale programma's (Java Message Queue) komt dit zaken als resource sharing en parallel processing sterk ten goede. Zonder middleware is de score minder gunstig.
2) De scalability van Java-applicaties was een groot probleem. Zeker vergeleken met andere technieken was dit ontoereikend. Gelukkig zijn er technieken op komst (zoals SCO's 'PerkUp' technologie) waardoor de scalability van Java op een acceptabel niveau komt.
3) Als men een Java applet in een browser wil uitvoeren, moet de applet eerst vanaf de server naar de client getransporteerd worden waarna de bytecode geïnterpreteerd wordt tot machinetaal. Elke keer dat men de pagina waarbinnen de applet wordt aangeroepen opvraagt, moeten bovenstaande handelingen uitgevoerd worden. Dit heeft een nadelig effect op de laadtijd van een Java applicatie.
4) De robustness en de executiesnelheid van Java applicaties zijn sterk afhankelijk van de Java Virtual Machine die gebruikt wordt. De ene JVM zorgt voor robuustheid, de ander voor snelheid. Het feit dat Java geen gebruik maakt van pointers komt de robuustheid ten goede, omdat onzorgvuldig gebruik van pointers kan leiden tot fouten tijdens de uitvoering van een applicatie. Java zorgt zelf voor het opruimen van de geheugenruimte die gebruikt wordt, waardoor de efficiency van Java vrij goed is te noemen.
5) Het probleem van ActiveX is het risico dat een ActiveX component kwaadaardige handelingen uit kan voeren. Dit risico is bij Java ondervangen door applets in een zogenaamde 'Sandbox' uit te laten voeren. Dit houdt in dat het programma dat de Java applet uitvoert (meestal een browser) beperkingen oplegt aan wat de applet wel en niet mag doen. Fouten in de implementatie van de Sandbox in een browser heeft in het verleden echter wel eens geleid tot onverwachte gebeurtenissen, als kwaadaardige applets de beveiliging wisten te omzeilen. Hierover wordt meer verteld in het artikel van Mark Kilsdonk in het maandblad informatie [KILS, 1999].

Deze Sandbox beperkt echter ook de functionaliteit van de applet. Zo kan een applet geen log-file op de client machine zetten. Alle log-files dienen dus op de server geplaatst te worden.

6) JavaBeans zijn op zichzelf staande stukken code, die hun functionaliteit aanbieden aan andere JavaBeans of programma's. Door een applicatie op te delen in JavaBeans is een zeer grote modulariteit mogelijk.
7) Java kan C/C++ code gebruiken. Indien code in een andere taal gebruikt moet worden, kan het beste C/C++ als 'wrapper' gebruikt worden. Dit houdt in dat een stukje code in C of C++ wordt geschreven, dat als 'vertaler' dient tussen Java en de andere taal. Delphi Pascal kan direct met Java communiceren, maar dit is lastig te bereiken. Ook voor Delphi Pascal is het beter om een C/C++-wrapper te gebruiken. Het is tevens mogelijk om via RMI met componenten te communiceren die gebruik maken van CORBA.
8) Op het Internet zijn veel 'communities' te vinden, virtuele gemeenschappen die zich richten op bepaalde aspecten van Java, zoals JavaBeans of Java Server Pages. In deze communities worden vragen beantwoord, nieuwe ontwikkelingen besproken en algemene informatie aangeboden.

7.2.1.3 C++

C++ is de meest gebruikte object georiënteerde taal. De vele mogelijkheden maken van C++ een krachtige maar lastige taal, waarbij veel fout kan gaan, vooral bij het gebruik van pointers (directe verwijzingen naar fysieke geheugenruimtes).

Ondanks het feit dat C++ niet platform-onafhankelijk is, is het evenals Java geschikt voor het schrijven van gedistribueerde applicaties. C++ kan gebruikt worden om ActiveX componenten te schrijven die in een ActiveX 'container' gedraaid kunnen worden (zoals Microsoft Internet Explorer). C++ wordt volledig gecompileerd tot machinecode (dus niet zoals Java tot bytecode) en is dus niet platform onafhankelijk.

Beoordeling:

C++			
Categorie	**Criterium**	**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie	5	
Distributie	Resource sharing	4	1)
	Parallel processing	4	1)
	Availability	4	1)
Interface	Browser interface	Ja	2)
	Graphical User Interface	Ja	
Prestaties	Laadtijd	3	
	Totale executietijd	5	3)

Communicatie architectuur	TCP/IP		Ja	
	HTTP		Nee	
	Middleware		Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen		Ja	
	Privacy en integriteit		Ja	4)
	Authenticatie en non-repudiation		Ja	
	Controleerbaarheid		Ja	
Software Engineering	Modulariteit		4	5)
	Compatibility		4	
	Robustness		4	6)
	Efficiency		5	7)
	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	Webserver:	Internet Information Server	NVT	
		Apache	NVT	
		Netscape	NVT	
		JavaSoft	NVT	
	Browser	MS Internet Explorer	Ja	2)
		Netscape Navigator	Nee	2)
Ontwikkeling	Gebruiksgemak IDE		3	8)
	Snelheid van ontwikkeling		3	
	Prijs van ontwikkeling		4	
	Leercurve		3	
	Algemene ondersteuning		4	9)

Opmerkingen:

1) Zaken als resource sharing en load balancing kunnen handmatig geprogrammeerd worden met C++, maar indien gebruik gemaakt wordt van middleware is dit overbodig. Zonder middleware scoort C++ een stuk minder goed op deze punten.
2) Met behulp van ActiveX kan een C++-applicatie binnen een webpagina geplaatst worden. Hierdoor kan de applicatie met Internet Explorer uitgevoerd worden. Ondersteuning van ActiveX door Netscape Navigator is slecht.
3) C++ is de snelste systeemtaal.
4) Ontwikkelaars kunnen met C++ gebruik maken van SSL of zelf een vorm van encryptie implementeren.
5) Aangezien C++ bekend staat om zijn goede mogelijkheden betreffende Object Oriented programmeren (zie ook paragraaf 3.1.1), is de beoordeling van modulariteit voor C++ goed.

6) De robuustheid van C++ is op zich goed, maar er moet wel zorgvuldig geprogrammeerd worden. Onzorgvuldig gebruik van pointers kan leiden tot instabiele programma's
7) Geheugenruimte die door het programma wordt gebruikt kan handmatig vrijgegeven worden, ook tijdens de uitvoering van het programma. Hierdoor kan de efficiency in vrij grote mate zelf vergroot worden.
8) C++ is een zeer complexe taal die daardoor vrij lastig is om te leren en te gebruiken.
9) Er zijn zeer veel mensen die C++ gebruiken. Ondersteuning is dus over het algemeen geen probleem.

7.2.1.4 Visual Basic

Visual Basic is een door Microsoft ontwikkelde programmeertaal. Het doel was een taal te ontwikkelen die makkelijk in gebruik was, en dat doel is bereikt. Een iets aangepaste versie, Visual Basic for Applications (VBA), wordt gebruikt in de populaire Microsoft Office producten als Word en Excel. Visual Basic is bij uitstek geschikt voor gebruik in combinatie met ActiveX en scripting-talen. Visual Basic wordt evenals Object Pascal en C++ gecompileerd tot machinecode.

Beoordeling:

Visual Basic			
Categorie	**Criterium**	**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie	5	
Distributie	Resource sharing	4	1)
	Parallel processing	4	1)
	Availability	4	1)
Interface	Browser interface	Ja	2)
	Graphical User Interface	Ja	
Prestaties	Laadtijd	3	3)
	Totale executietijd	4	
Communicatie architectuur	TCP/IP	Ja	
	HTTP	Nee	
	Middleware	Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen	Ja	4)
	Privacy en integriteit	Ja	
	Authenticatie en non-repudiation	Ja	
	Controleerbaarheid	Ja	
Software Engineering	Modulariteit	3	
	Compatibility	3	
	Robustness	3	
	Efficiency	3	

	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Nee	
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Nee	
	Webserver:	Internet Information Server	NVT	
		Apache	NVT	
		Netscape	NVT	
		JavaSoft	NVT	
	Browser	MS Internet Explorer	Ja	2)
		Netscape Navigator	Nee	2)
Ontwikkeling	Gebruiksgemak IDE		5	5)
	Snelheid van ontwikkeling		4	
	Prijs van ontwikkeling		5	6)
	Leercurve		4	
	Algemene ondersteuning		4	

Opmerkingen:

1) Ook aan Visual Basic kan het gebruik van middleware veel voordelen bieden met betrekking tot resource sharing, load balancing e.d. Zonder middleware valt de score aanzienlijk lager uit.
2) Met behulp van ActiveX kan een Visual Basic applicatie in Microsoft Internet Explorer weergegeven worden.
3) Visual Basic heeft en goede executiesnelheid, al is het niet zo snel als C++.
4) Visual Basic biedt ondersteuning voor SSL, maar kan ook andere methoden toepassen voor beveiliging.
5) Visual Basic (de naam zegt het al) is een makkelijke taal om te leren en te gebruiken.
6) Visual Basic is redelijk gemiddeld geprijsd. Het is als onderdeel van Visual Studio te koop. Visual Studio omvat onder andere Visual C++ en Visual J++ (Java). Toevoegingen voor Visual Basic, zoals ondersteuning voor CORBA, zijn ook te koop.

7.2.2 Scripting-technieken

De scripting-technieken die hier genoemd en beoordeeld worden, maken allemaal gebruik van scripts die op de server uitgevoerd worden, het zogenaamde ‘server-side scripting’. De resultaten van de bewerkingen worden in HTML pagina’s teruggestuurd naar de gebruiker, waarna die in een browser weergegeven worden. Voor al deze technieken geldt dat ze gebruik maken van een browser interface. Hierdoor zijn ze onafhankelijk van de gebruikte browsers, en ook van het Operating System op de client.

7.2.2.1 CGI (Common Gateway Interface)

Bij CGI staat er op de webserver een programma dat, via de browser, informatie ontvangt van de bezoeker van een webpagina. Deze informatie wordt dan verwerkt door het programma, waarna het HTML terugstuurt naar de browser. CGI werkt nauw samen met het HTTP protocol, doordat het de standaard commando's van HTTP gebruikt (zie ook paragraaf 3.3.3.1). Voor elke bezoeker van de website wordt het programma apart opgestart. Dit kost tijd en geheugenruimte op de webserver en is daarom niet goed voor prestaties en efficiency. CGI kan met speciaal daarvoor ontwikkelde scripting talen geprogrammeerd worden (zoals Perl), maar ook met bijvoorbeeld Object Pascal in Delphi.

Beoordeling:

CGI				
Categorie	**Criterium**		**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie		4	
Distributie	Resource sharing		3	1)
	Parallel processing		3	1)
	Availability		3	1)
Interface	Browser interface		Ja	
	Graphical User Interface		Nee	
Prestaties	Laadtijd		3	2)
	Totale executietijd		3	3)
Communicatie architectuur	TCP/IP		Ja	
	HTTP		Ja	
	Middleware		Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen		Ja	
	Privacy en integriteit		Ja	4)
	Authenticatie en non-repudiation		Ja	
	Controleerbaarheid		Ja	
Software Engineering	Modulariteit		3	
	Compatibility		3	
	Robustness		2	
	Efficiency		1	2)
	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	Webserver:	Internet Information Server	Ja	
		Apache	Ja	
		Netscape	Ja	
		JavaSoft	Ja	

	Browser	MS Internet Explorer Netscape Navigator	Ja Ja	
Ontwikkeling	Gebruiksgemak IDE Snelheid van ontwikkeling Prijs van ontwikkeling Leercurve Algemene ondersteuning		3 2 5 3 4	5) 6) 7) 8) 9)

Opmerkingen:

1) De mogelijkheden van CGI met betrekking tot zaken als resource sharing en parallel processing zijn afhankelijk van de taal die gebruikt wordt om CGI mee te schrijven. Zo biedt Object Pascal een betere ondersteuning dan Perl. Toch is het gebruik van middleware sterk aan te raden omdat CGI zonder middleware aanzienlijk slechter scoort op deze punten.
2) CGI is berucht om zijn matige efficiency. Voor iedere nieuwe bezoeker wordt er een apart proces op de server opgestart. Alle voorbereidingen die in een programma bij het opstarten getroffen moeten worden, worden zodoende voor elke bezoeker opnieuw uitgevoerd. Dit kost tijd en geheugenruimte.
3) De snelheid van CGI is volledig afhankelijk van de gebruikte taal. Gecompileerde talen (C++, Object Pascal) zijn sneller dan geïnterpreteerde talen (zoals scripts).
4) CGI ondersteunt het gebruik van SSL en standaarden voor authenticatie die in het HTTP protocol verwerkt zijn.
5) Het is vrij simpel om een CGI applicatie te schrijven, zeker als gebruik wordt gemaakt van talen die speciaal voor CGI zijn ontwikkeld, zoals Perl.
6) Indien gebruik wordt gemaakt van Perl of Delphi moet de presentatie van de output zelf door de programmeur verzorgd worden. Dit houdt in dat binnen het programma de HTML (waaruit de resultaat-pagina bestaat) ‘handmatig’ ingetypt moet worden voordat die naar de bezoeker teruggestuurd wordt. Dit komt de snelheid van ontwikkeling niet ten goede. Mogelijk bestaan er ontwikkelomgevingen waar de opbouw van de presentatie makkelijker is.
7) De prijs van ontwikkeling is heel goed. Perl is gratis te downloaden en er zijn zeer veel gratis voorbeelden en CGI-programma’s te krijgen.
8) De leercurve is voor een groot deel afhankelijk van de gebruikte taal. Daarnaast is de leercurve van CGI vrij vlak.
9) CGI was één van de eerste vormen van server-side scripting. Zodoende heeft het veel aanhangers verzameld. Er zijn dan ook veel gebruikersgroepen die antwoorden geven op allerlei vragen betreffende CGI.

7.2.2.2 ASP (Active Server Pages)

Active Server Pages is een door Microsoft ontwikkelde scripting-techniek. ASP is een server-side scripting principe dat opgevraagde gegevens in een bepaald formaat aan de gebruiker presenteert. ASP is ontwikkeld als een kruising tussen SQL (een 'taal' om gegevens uit een database te kunnen halen) en HTML.

Active Server Pages zijn 'gewone' webpagina's die HTML en scripts bevatten. Die scripts worden uitgevoerd op de server. Nadat de server-side code is uitgevoerd door de server, ontvangt de client het resultaat in een pagina waar alleen client-side code zoals HTML, JavaScript of VBScript op staat. (Daarom staat in een pagina nooit de server-side code).

ASP is voornamelijk gericht op de mogelijkheid om via een browser gegevens op te vragen uit een database. SQL aanroepen kunnen binnen ASP pagina's uitgevoerd worden op elke database die benaderd kan worden via het ODBC protocol.

ASP ondersteunt verschillende scripting-talen. De twee meest gebruikte talen voor ASP zijn VBScript en JScript of JavaScript (JScript is Microsoft's versie van JavaScript), maar ook andere scripting-talen kunnen gebruikt worden.

Beoordeling:

ASP			
Categorie	**Criterium**	**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie	3	
Distributie	Resource sharing	3	1)
	Parallel processing	3	1)
	Availability	3	1)
Interface	Browser interface	Ja	
	Graphical User Interface	Nee	
Prestaties	Laadtijd	4	
	Totale executietijd	3	
Communicatie architectuur	TCP/IP	Ja	
	HTTP	Ja	
	Middleware	Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen	Ja	2)
	Privacy en integriteit	Ja	
	Authenticatie en non-repudiation	Ja	
	Controleerbaarheid	Ja	
Software Engineering	Modulariteit	3	3)
	Compatibility	3	4)
	Robustness	3	5)
	Efficiency	3	

	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	6)
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	Webserver:	Internet Information Server	Ja	6)
		Apache	Ja	6)
		Netscape	Ja	6)
		JavaSoft	Ja	6)
	Browser	MS Internet Explorer	Ja	
		Netscape Navigator	Ja	
Ontwikkeling	Gebruiksgemak IDE		5	7)
	Snelheid van ontwikkeling		5	7)
	Prijs van ontwikkeling		4	
	Leercurve		3	
	Algemene ondersteuning		4	8)

Opmerkingen:

1) Indien gebruik wordt gemaakt van middleware is de beoordeling van ASP op deze punten goed. Zonder middleware valt de beoordeling minder positief uit.
2) SSL is de methode bij uitstek voor beveiliging van applicaties die gebruik maken van server-side scripting, en wordt ondersteund door ASP.
3) De ASP-code die gebruikt wordt kan in aparte bestanden opgeslagen worden, zogenaamde scriptlets. Deze scriptlets kunnen dan vanuit een ASP-pagina aangeroepen worden. Zo kan een applicatie redelijk modulair opgezet worden.
4) ASP kan zowel in JavaScript als in VBScript geschreven worden. ASP kan dll's (Dynamic Link Libraries) in andere talen aanroepen. Een dll is een bestand waarin stukken code staan die door andere programma's aangeroepen kunnen worden.
5) De robustness van ASP is vrij goed als de server Unix als besturingssysteem heeft. Het is wat minder goed als de server Windows als besturingssysteem heeft. Al met al is de robuustheid van JSP groter (zie verderop).
6) Standaard wordt ASP alleen ondersteund door Windows en Internet Information Server. Met behulp van speciale programma's, zoals Chili!ASP van Chili!Soft of Instant ASP van Halcyon Software, kan ASP ook gebruikt worden op andere platforms en met andere servers.
7) ASP is makkelijk te leren en te gebruiken, en de snelheid van ontwikkeling is heel goed. Grootschaliger applicaties zijn echter wat lastiger te onderhouden.

8) ASP is een populaire techniek met veel gebruikers. De algemene ondersteuning ligt daarom op een goed niveau.

7.2.2.3 JSP (Java Server Pages)

Java Server Pages (JSP) is een technologie voor het beheersen van inhoud of uiterlijk van web pagina's door het gebruik van servlets. Een servlet is een klein Java programmaatje dat op een server draait. De naamgeving is afgeleid van de Java applet, een klein programma dat als een los bestand met een web pagina meegestuurd wordt en bij de client draait. Soms is het echter nodig een programma op de server te laten draaien, bijvoorbeeld in verband met communicatie met een database. Traditioneel zijn deze geïmplementeerd als een CGI applicatie, en de laatste tijd komen principes als ASP en ColdFusion op als oplossingen hiervoor. Indien echter op de server een Java Virtual Machine draait, kunnen zulke applicaties in Java geïmplementeerd worden. Op de webpagina staat aangegeven welke servlet aangeroepen dient te worden. Het verschil tussen JSP en het gebruik van alleen servlets ligt in het feit dat JSP de inhoud en de presentatie scheidt. Indien de lay-out van een pagina, die door een servlet wordt gegenereerd, veranderd moet worden, moet de hele servlet ge-herprogrammeerd worden. Indien pagina-generatie op basis van JSP gebeurt, kan de servlet gewoon intact blijven. Dit gebeurt door vanuit pagina's (waarvan de lay-out dus is vastgelegd) servlets aan te roepen, die dan zorgen voor de inhoud.

JSP wordt door Sun Microsystems ook wel de Servlet API genoemd. JSP is vergelijkbaar met ASP. Waar een Java Server Page een Java programmaatje aanroept dat door de webserver wordt uitgevoerd, bevat een Active Server Page een script dat geïnterpreteerd wordt door een script interpreter (zoals VBScript of JScript) voordat de pagina naar de gebruiker wordt toegestuurd.

Beoordeling:

JSP			
Categorie	**Criterium**	**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie	4	1)
Distributie	Resource sharing	4	2)
	Parallel processing	4	2)
	Availability	4	2)
Interface	Browser interface	Ja	
	Graphical User Interface	Nee	
Prestaties	Laadtijd	4	
	Totale executietijd	3	3)
Communicatie architectuur	TCP/IP	Ja	
	HTTP	Ja	
	Middleware	Ja	

Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen		Ja	4)
	Privacy en integriteit		Ja	
	Authenticatie en non-repudiation		Ja	
	Controleerbaarheid		Ja	
Software Engineering	Modulariteit		5	5)
	Compatibility		3	6)
	Robustness		4	6)
	Efficiency		4	6)
	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	Webserver:	Internet Information Server	Ja	
		Apache	Ja	
		Netscape	Ja	
		JavaSoft	Ja	
	Browser	MS Internet Explorer	Ja	
		Netscape Navigator	Ja	
Ontwikkeling	Gebruiksgemak IDE		3	7)
	Snelheid van ontwikkeling		3	8)
	Prijs van ontwikkeling		4	
	Leercurve		4	
	Algemene ondersteuning		5	9)

Opmerkingen:

1) JSP maakt gebruik van JDBC (Java DataBase Connectivity) om met databases te communiceren. JDBC is vergelijkbaar met ODBC (Open DataBase Connectivity), een standaard voor database-communicatie. Via JDBC kan JSP met bijna alle databases communiceren.
2) JSP kan net zoals Java gebruik maken van RMI als middleware.
3) De executiesnelheid is groter dan bij scripting talen, omdat de Servlets al gecompileerd zijn tot bytecode. Het is echter weer trager dan andere systeemtalen (met uitzondering van Java).
4) JSP biedt de mogelijkheid om gegevens met behulp van SSL te versleutelen.
5) De Application Logic kan bij JSP onderverdeeld worden in zogenaamde servlets: dit zijn in feite JavaBeans (zie § 7.2.1.2) die vanuit een JSP-pagina aangeroepen worden.
6) De servlets zijn geschreven in Java. De compatibility, robustness en efficiency van Java zijn dus van toepassing. De compatibility van JSP met andere Java componenten is echter zeer goed te noemen.

7) Java is niet de makkelijkste taal, maar ook niet extreem moeilijk om te leren. Het lijkt wat op C++.
8) De ontwikkelingssnelheid is wat lager dan ASP. Wel is een groot project makkelijker te beheersen dan bij ASP.
9) Zoals bij Java werd genoemd bestaan er verscheidene communities. Dit is ook het geval voor JSP.

7.2.2.4 ColdFusion

ColdFusion is een voorloper van, en een alternatief voor ASP. ColdFusion is ontwikkeld door Allaire, en na ASP de meest gebruikte server-side scripting taal. ColdFusion heeft zijn eigen 'Markup Language' (lay-out definitie taal), genaamd CFML (ColdFusion Markup Language). CFML omvat HTML en het nieuwe XML (eXtensible Markup Language, zie appendix B). De werking van ColdFusion is vergelijkbaar met die van ASP. Om ColdFusion te gebruiken, dient ColdFusion Server geïnstalleerd te worden op de server. ColdFusion Server moet apart gekocht worden.

Beoordeling:

ColdFusion				
Categorie	**Criterium**		**Beoordeling**	**Opmerkingen**
Database communicatie	Database communicatie		4	
Distributie	Resource sharing		4	1)
	Parallel processing		4	1)
	Availability		4	
Interface	Browser interface		Ja	
	Graphical User Interface		Nee	
Prestaties	Laadtijd		4	
	Totale executietijd		3	
Communicatie architectuur	TCP/IP		Ja	
	HTTP		Ja	
	Middleware		Ja	
Beveiligingsaspecten	Beschikbaarheid geautoriseerde personen		Ja	
	Privacy en integriteit		Ja	4)
	Authenticatie en non-repudiation		Ja	
	Controleerbaarheid		Ja	
Software Engineering	Modulariteit		4	5)
	Compatibility		4	6)
	Robustness		2	7)
	Efficiency		4	
	Portability:			
	OS Server:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	8)

	OS Client:	Windows 95/98	Ja	
		Windows NT	Ja	
		UNIX	Ja	
	Webserver:	Internet Information Server	Ja	
		Apache	Ja	
		Netscape	Ja	
		JavaSoft	Nee	
	Browser	MS Internet Explorer	Ja	
		Netscape Navigator	Ja	
Ontwikkeling	Gebruiksgemak IDE		4	9)
	Snelheid van ontwikkeling		3	
	Prijs van ontwikkeling		2	10)
	Leercurve		3	
	Algemene ondersteuning		3	11)

Opmerkingen:

1) ColdFusion kan profiteren van de voordelen die middleware biedt.
2) De ColdFusion Server (die apart gekocht moet worden) zorgt voor een goede load balancing.
3) ColdFusion is redelijk scalable; bij extreem hoge belastingen wil het nog wel eens fout gaan.
4) ColdFusion kan gebruik maken van de faciliteiten die SSL biedt. Ook heeft ColdFusion zijn eigen methoden voor beveiliging.
5) Doordat de Application Logic in aparte ‘snippets’ opgeslagen kan worden (vergelijkbaar met scriptlets van ASP) ligt de modulariteit van ColdFusion op een goed niveau.
6) ColdFusion kan eerder geschreven code in Java en C++ gebruiken, evenals stukken script in JavaScript en VBScript. Tevens biedt ColdFusion ondersteuning voor de ActiveX technologie en CGI.
7) Voor simpele websites is ColdFusion vrij robuust te noemen. Zodra echter de belasting toeneemt neemt de betrouwbaarheid af.
8) Alleen voor de HP-UX en Solaris Unix versies. Overigens moet vermeld worden dat bij de ontwikkeling alleen het Windows besturingssysteem ondersteund wordt.
9) Om ColdFusion te gebruiken dient men wel een nieuwe taal te leren: ColdFusion Markup Language (CFML, vergelijkbaar met HTML). Deze is echter vrij makkelijk onder de knie te krijgen.
10) Zoals al eerder genoemd dient de ColdFusion Server apart gekocht te worden. Dit is geen goedkoop product.
11) ColdFusion is een populaire techniek. Er zijn dus veel gebruikers die elkaar kunnen helpen. De ondersteuning van Allaire zelf (de fabrikant van ColdFusion) laat echter soms wel wat te wensen over.

7.3 Conclusie

De beoordelingen in de vorige paragraaf zijn in zekere mate subjectief, en vaak ook afhankelijk van de beschikbare ontwikkelomgevingen. Het is daarom

niet de bedoeling dat de bevindingen als absolute waarheden gehanteerd worden, maar meer als algemene leidraad bij het nemen van beslissingen.

Om een overzicht van de resultaten te kunnen krijgen, zijn de beoordelingen naast elkaar gezet in de volgende matrix:

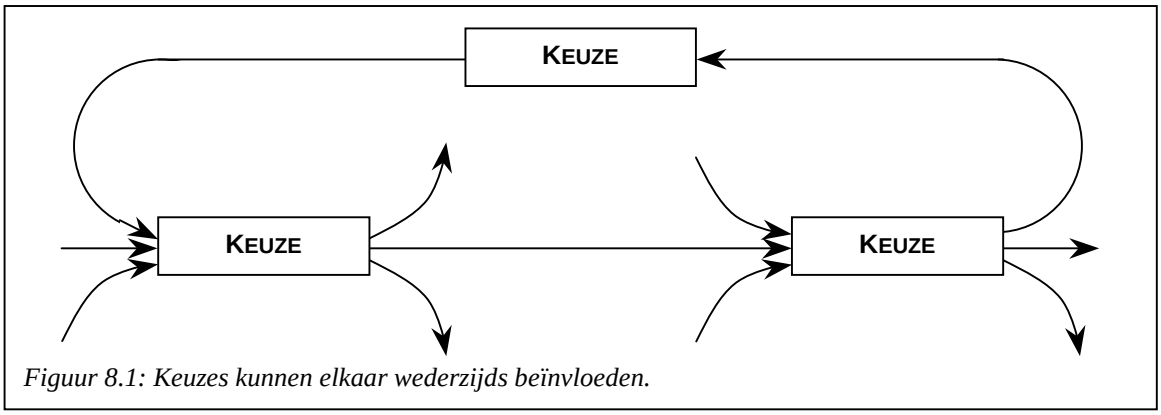
CRITERIUM		**TECHNIEK**							
		OBJECT PASCAL	JAVA	C++	VISUAL BASIC	CGI	ASP	JSP	COLDFUSION
Database communicatie		5	4	5	5	4	3	4	4
Resource sharing		4	4	4	4	3	3	4	4
Parallel processing		4	4	4	4	3	3	4	4
Availability		4	4	4	4	3	3	4	4
Browser interface		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Graphical User Interface		Ja	Ja	Ja	Ja	Nee	Nee	Nee	Nee
Laadtijd		3	2	3	3	3	4	4	4
Totale executietijd		4	3	5	4	3	3	3	3
TCP/IP		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
HTTP		Nee	Nee	Nee	Nee	Ja	Ja	Ja	Ja
Middleware		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Beschikb. geautoriseerde pers.		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Privacy en integriteit		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Authenticatie en non-repudiation		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Controleerbaarheid		Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
Modulariteit		4	5	4	3	3	3	5	4
Compatibility		4	3	4	3	3	3	3	4
Robustness		4	4	4	3	2	3	4	2
Efficiency		4	4	5	3	1	3	4	4
Portability:									
OS Server	Windows 95/98	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
	Windows NT	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
	UNIX	Nee	Ja	Ja	Nee	Ja	Ja	Ja	Ja
OS Client	Windows 95/98	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
	Windows NT	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
	UNIX	Nee	Ja	Ja	Nee	Ja	Ja	Ja	Ja
Webserver	IIS	NVT	NVT	NVT	NVT	Ja	Ja	Ja	Ja
	Apache	NVT	NVT	NVT	NVT	Ja	Ja	Ja	Ja
	Netscape	NVT	NVT	NVT	NVT	Ja	Ja	Ja	Ja
	JavaSoft	NVT	NVT	NVT	NVT	Ja	Ja	Ja	Nee
Browser	Internet Explorer	Ja	Ja	Ja	Ja	Ja	Ja	Ja	Ja
	Navigator	Nee	Ja	Nee	Nee	Ja	Ja	Ja	Ja
Gebruiksgemak IDE		5	3	3	5	3	5	3	4
Snelheid van ontwikkeling		4	3	3	4	2	5	3	3
Prijs van ontwikkeling		4	4	4	5	5	4	4	2
Leercurve		4	4	3	4	3	3	4	3
Algemene ondersteuning		4	5	4	4	4	4	5	3

Zoals uit de beoordelingen blijkt, heeft elke techniek zijn eigen voor- en nadelen. Voor elke situatie dient moet men daarom nagaan, welke techniek het meest overeenkomt met de eisen die gesteld worden door die specifieke situatie. Dit wordt in het volgende hoofdstuk toegelicht.

8 METHODIEK

In dit hoofdstuk wordt ingegaan op het beslissingstraject voor de ontwikkeling van gedistribueerde applicaties. Dit beslissingstraject betreft de keuzes die gemaakt moeten worden teneinde een gedistribueerde applicatie te ontwikkelen die voldoet aan de functionele specificaties. Dit moet resulteren in voorschriften zoals de voorschriften die in de traditionele Software Engineering voor stand-alone applicaties worden beschreven. In paragraaf 8.1 wordt een bestaande applicatie beschreven (met enige toegevoegde fictieve aspecten om de bespreking compleet te maken). Op basis van die specificaties wordt in paragraaf 8.2 een realistisch beslissingstraject beschreven. Vervolgens wordt in paragraaf 8.3 geabstraheerd van de beschreven applicatie en van de situatie van de beslissingsnemers, om zodoende een generieke methodiek te kunnen afleiden die kan dienen als leidraad voor de besluitvorming omtrent gedistribueerde applicatie ontwikkeling.

De keuzes die bij de evaluatie van de applicatie gemaakt worden, kunnen leiden tot een algemene methodiek ten aanzien van de ontwikkeling van gedistribueerde applicaties. Het is echter belangrijk om te realiseren dat een beslissingsproces vaak een iteratief denkproces is, waarbij verschillende beslissingen elkaar wederzijds kunnen beïnvloeden. De gevolgen van een bepaalde keuze kunnen de mogelijkheden voor andere keuzes bijvoorbeeld beperken. Hierdoor kan het voorkomen dat voor een beslissing geen acceptabele opties bestaan, waardoor men moet '*back-tracken*' naar een voorgaande keuze, die de mogelijkheden heeft beperkt voor de beslissing waar de knelpunten optreden. Deze wederzijdse invloeden kunnen grafisch als volgt worden weergegeven, waarbij de blokjes keuzes voorstellen, binnenkomende pijlen zijn gegevens die de keuze beïnvloeden, en uitgaande pijlen zijn invloeden die van een bepaalde keuze uitgaan:



Figuur 8.1: Keuzes kunnen elkaar wederzijds beïnvloeden.

Deze eigenschap heeft als gevolg, dat er veel manieren zijn om dit beslissingsproces vorm te geven. Elke vorm heeft zijn eigen volgorde van beslissingsmomenten. Een beslissingsmoment is een onderdeel van het beslissingstraject, waarbinnen een keuze gemaakt moet worden voor een bepaalde structuur, techniek of methode. De invulling van een specifiek

beslissingsproces, zoals de volgorde van de keuzes en de mate waarin men '*back-trackt*', is afhankelijk van een aantal zaken. Het betreft hierbij niet alleen de applicatie zelf, maar ook bijvoorbeeld persoonlijke eigenschappen van de personen die de beslissingen nemen. De keuzes die gemaakt worden in de onderstaande evaluatie, worden daarom als volgt beschreven:

- Voor elk beslissingsmoment worden de verschillende mogelijkheden voor het maken van de keuze gegeven.
- Voor zover mogelijk wordt beschreven welke invloeden eerdere keuzes op latere keuzes hebben.
- Gegeven de beschreven situatie wordt een keuze gemaakt uit de verschillende mogelijkheden.

De volgorde van de beslissingen ligt niet vast en is in de onderstaande evaluatie dus slechts een voorbeeld.

Voor het maken van een keuze wordt zoveel mogelijk gebruik gemaakt van de informatie die in deze scriptie is gegeven. Men mag echter niet vergeten dat in bij elke keuze een zekere mate van subjectiviteit is vertegenwoordigd. Deze subjectiviteit is afkomstig van de voorkeuren van de beslissingnemers en factoren die binnen de betrokken organisatie spelen, zoals al aanwezige ontwikkelomgevingen.

8.1 Beschrijving van de applicatie

De applicatie is een *Crew Duty Rostering* programma voor een vliegmaatschappij. Dit houdt in dat met deze applicatie vluchten worden gepland en bemanningen voor die vluchten worden ingeroosterd. Daarnaast biedt het programma *real-time* operationele ondersteuning. Dat wil zeggen dat, indien zich ongeregeldheden voordoen rond de vluchten, het programma mogelijkheden moet bieden om deze ongeregeldheden te verhelpen. Een voorbeeld is het inroosteren van een andere piloot als een piloot ziek is, maar ook het treffen van maatregelen bij vertragingen behoort tot de mogelijkheden.

Indien we op een meer technisch vlak naar de applicatie kijken, komen we de volgende eigenschappen tegen:

De (vertrouwelijke) gegevens betreffende vluchten en personeel staan in een database. Deze gegevens worden gebruikt door een planmodule die personeel toewijst aan vluchten. Dit schema wordt voor een periode van drie weken vastgelegd. Ook is er een operationele module die van dezelfde gegevens gebruik maakt om de eerdergenoemde *real-time* ondersteuning te bieden. Hierbij kan gedacht worden aan het vervangen van een personeelslid vanwege ziekte. Het programma moet een geavanceerde interface hebben, die het mogelijk maakt om bijvoorbeeld personeel grafisch van de ene vlucht naar de andere te 'slepen' (*drag-and-drop*). Daarnaast moet de applicatie uitgebreide rapportage-mogelijkheden bieden.

De planmodule en operationele module moeten door meerdere gebruikers vanaf verschillende locaties gebruikt kunnen worden. Sommige onderdelen van de applicatie moet berekend zijn op een groot aantal gebruikers. Daarnaast moeten andere applicaties ook gebruik kunnen maken van de gegevens van deze applicatie. Ook moet het programma een zogenaamde *browser-interface* bieden. Er moet rekening gehouden worden dat de Application Logic (bevat in de planmodule en operationele module) ook door andere applicaties gebruikt moet kunnen worden. Gezien het karakter van de applicatie, is het belangrijk dat de applicatie altijd beschikbaar is. Tenslotte is de kans groot dat de komende tijd regelmatig uitbreidingen aan de applicatie aangebracht zullen worden.

8.2 Evaluatie

In deze paragraaf worden eerst wat algemene aspecten besproken. Daarna worden de keuzes voor de applicatie als geheel behandeld, waarna we 'inzoomen' op de applicatie en de onderdelen *Interface* en *Application Logic* afzonderlijk behandelen.

8.2.1 Algemeen

De ontwikkelingsaspecten die zijn besproken in hoofdstuk 5 (en de criteria die in hoofdstuk 6 daarvan zijn afgeleid) hebben invloed op elke keuze binnen het beslissingsproces. Het is wat bewerkelijk om deze voor elk beslissingsmoment te noemen, daarom worden ze hier apart genoemd. Deze aspecten zijn:

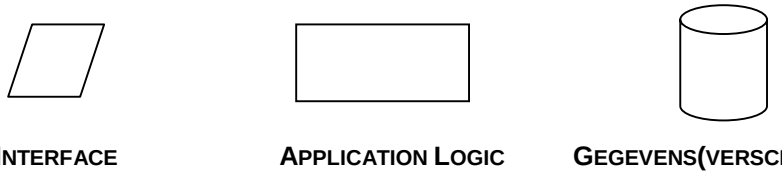
- Het ontwikkelgemak
- Het tijdsvenster waarbinnen de ontwikkeling moet worden afgerond (*Time-to-market*)
- De toegestane kosten
- Het gemak waarmee, en de tijd waarbinnen, een ontwikkelaar met de techniek kan leren werken. Dit hangt samen met de al aanwezige kennis van een techniek binnen een organisatie
- Ondersteuning door anderen

Afhankelijk van de betrokken keuze, gelden deze aspecten voor de ontwikkeling van de gehele applicatie (paragraaf 8.2.2), of een onderdeel daarvan (paragrafen 8.2.3 en 8.2.4).

8.2.2 Applicatie als geheel

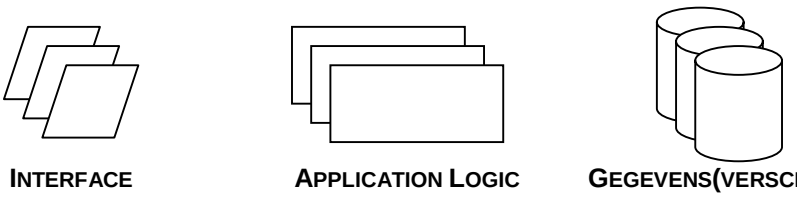
Als we op een vrij hoog niveau naar de applicatie kijken, kunnen we een aantal beslissingen nemen die invloed hebben op de gehele applicatie. Dit betreft in eerste instantie de structuur. In hoofdstuk 3 is gesproken over de

component-architectuur en de communicatie-architectuur. In hetzelfde hoofdstuk werd beargumenteerd dat er met betrekking tot de component architectuur gekozen moet worden voor meerdere componenten. De wens dat de Application Logic ook door andere applicaties gebruikt moet kunnen worden, is een argument voor deze indeling. Dit resulteert in de volgende situatie:



Figuur 8.2: Structuur applicatie.

Het bedrijfskritische karakter van de applicatie vraagt om maatregelen met betrekking tot de beschikbaarheid van de Application Logic en de gegevens, zeker gezien het feit dat veel gebruikers gelijktijdig de applicatie moeten kunnen gebruiken. Dit geeft aanleiding om de Application Logic en gegevens te repliceren op andere machines. Een andere aanleiding voor replicatie van de Application Logic is het *resource*-intensieve karakter hiervan. Replicatie verschaft dan de mogelijkheid om *parallel processing* toe te passen en te zorgen voor een hoge *availability*. Om deze adviezen op te kunnen volgen, is het noodzakelijk om een multi-tier structuur te gebruiken. Dit resulteert bij deze applicatie in meerdere Interface gedeeltes, meerdere Application Logic gedeeltes en een gerepliceerde database. Dit leidt tot de volgende figuur:



Figuur 8.3: Structuur applicatie.

Bij de implementatie van bovengenoemde functionaliteiten (load balancing, parallel processing e.d.) kan middleware een grote toegevoegde waarde hebben, zoals reeds werd genoemd in hoofdstuk 3. De drie genoemde vormen van middleware bieden ruwweg dezelfde functionaliteit. Bij de keuze voor een vorm van middleware moet er echter wel rekening gehouden worden met het platform waarop de applicatie uitgevoerd zal worden. Indien voor COM+ gekozen wordt, is men beperkt tot het Windows Operating System. Gelukkig bestaan er methoden om communicatie tussen COM+, CORBA en RMI mogelijk te maken, dus zijn niet alle componenten gebonden aan Windows. CORBA en RMI

hebben geen beperkingen met betrekking tot het platform. Omdat de applicatie (in eerste instantie) alleen gericht is op het Windows platform, en vanwege eerdere ervaring met COM+, wordt gekozen voor COM+ als middleware.

Het opdelen van de applicatie in componenten en het plaatsen op meerdere tiers kan leiden tot een meer complexe applicatie en een langere ontwikkeltijd, ook als van middleware gebruik gemaakt wordt. Het gebruik van middleware kan tot hogere kosten leiden indien aparte aankopen gedaan moeten worden. Onderhoud en uitbreidingen zijn echter makkelijker te realiseren indien de applicatie in componenten is opgesplitst. Omdat er een geavanceerde Interface nodig is die direct communiceert met de Application Logic, en omdat we kiezen voor een 3-tier structuur, is het gebruik van middleware noodzakelijk.

Gegeven het karakter van de applicatie zal de communicatie tussen de componenten vrij gegevensintensief zijn. In dit licht is het aan te raden om een 'licht' communicatie protocol te gebruiken. TCP/IP is het snelste protocol dat in deze scriptie behandeld wordt. Indien echter gebruik wordt gemaakt van middleware, zal een protocol 'bovenop' TCP/IP gebruikt worden, wat enige *overhead* met zich mee zal brengen. Dit gaat ten koste van de snelheid.

Wat ook in consideratie moet worden genomen, is het feit dat de gegevens vertrouwelijk zijn. Er moet dus een vorm van beveiliging toegepast worden. Ook dit betekent een bepaalde *overhead* voor de communicatie. Er moeten namelijk extra gegevens heen en weer gestuurd worden. Het grote probleem bij beveiliging is sleutelbeheer. SSL lost dit op door middel van 'certificates' (zie ook hoofdstuk 4), die door alle betrokken partijen moet worden aangevraagd, betaald en geïnstalleerd. Dit is vrij bewerkelijk en kostbaar, maar op dit moment wel de meest betrouwbare manier om sleutels te beheren. Indien de beveiliging wel belangrijk wordt gevonden, maar niet van levensbelang, kan worden gekozen voor een vorm van *private key* encryptie waarbij het sleutelbeheer zelf geregeld wordt. Er bestaan simpelere algoritmes die voor weinig geld te gebruiken zijn en die een voldoende niveau van beveiliging bieden. Voor de beschreven applicatie wordt voor de laatste variant gekozen, waarbij de private key gebaseerd wordt op de gegevens die de klant gebruikt om zich te authenticeren.

8.2.3 Interface

Vrij vroeg in het beslissingstraject moet er een beslissing genomen worden betreffende de te gebruiken interface: een 'stand-alone interface' met een graphical user interface (GUI) of een interface die binnen een browser uitgevoerd wordt. Indien voor een stand-alone interface gekozen wordt, vallen de scripting technieken buiten beschouwing, omdat ze die

mogelijkheid niet bieden. Als er gekozen wordt voor een ‘browser-interface’ zijn de scripting technieken wel mogelijk, evenals de systeemtaal Java. Andere systeemtalen zijn alleen mogelijk indien gebruik wordt gemaakt van ActiveX. De voor- en nadelen van de verschillende interfaces zijn in hoofdstuk 6 behandeld.

De keuze hangt onder meer af van de gewenste functionaliteit van de interface. Indien het een geavanceerde interface betreft, met mogelijkheden als *drag-and-drop* en geavanceerde rapportage mogelijkheden, zal men voor een systeemtaal kiezen. Als de interface snel zichtbaar moet zijn, genieten scripting technieken de voorkeur vanwege de betere laadtijd. De executiesnelheid van systeemtalen is echter weer beter dan die van scripting technieken. Dit is mogelijk te combineren door voor de interface een scripting techniek te kiezen, en voor de Application Logic een systeemtaal. De verschillende technieken moeten dan wel goed ‘samen kunnen werken’.

Ook kan het platform waarop de interface uitgevoerd moet worden is een factor in het keuzeproces. Indien de interface op alle platforms ondersteund moet worden, valt ActiveX af. ActiveX is namelijk alleen geschikt voor het Windows platform, alhoewel er ontwikkelingen gaande zijn die ActiveX geschikt moeten maken voor meerdere platforms. De andere technieken zijn geschikt voor alle platforms. Hierbij moet wel vermeld worden, dat voor de systeemtalen, met uitzondering van Java, voor elk platform een aparte versie gemaakt moet worden. Java daarentegen is direct voor elk platform geschikt.

Indien gekozen wordt voor een browser-interface, heeft ActiveX ook enkele beperkingen. Op dit moment wordt ActiveX alleen ondersteund door Microsoft Internet Explorer. Ontwikkelingen zijn gaande om dit ook voor Netscape Navigator te bewerkstelligen, maar die zijn nog niet ver genoeg om van de combinatie ActiveX en Netscape Navigator een optie te maken. De andere technieken worden door beide browsers even goed ondersteund.

Zoals in de beschrijving werd vermeld, zal gebruik worden gemaakt van een browser-interface. In het geval van de Crew Duty Rostering applicatie moet de interface zeer geavanceerd zijn, met vergaande functionaliteit op het gebied van *drag-and-drop* en visualisatie van gegevens en statistieken. Daarom wordt gekozen voor een systeemtaal. Aangezien de verschillende systeemtalen goed met elkaar kunnen samenwerken, is wordt de beslissing betreffende de techniek voor de interface, niet beïnvloed door de keuze voor de techniek waarin de Application Logic ontwikkeld zal worden. Omdat de Interface op Windows machines zal ‘draaien’, en vanwege de ervaring van de ontwikkelaars met Object Pascal, wordt voor Delphi gekozen als ontwikkelomgeving. Hierdoor is het gebruik van ActiveX nodig om een browser-interface mogelijk te maken. Het feit dat

Internet Explorer voorlopig de enige bruikbare browser is, wordt niet als een probleem gezien.

8.2.4 Application Logic

De Application Logic is in veel gevallen een heel belangrijk onderdeel van de totale applicatie. Ook in dit voorbeeld is dat het geval, temeer daar de Application Logic ook door andere applicaties gebruikt moet kunnen worden. Er zullen dus hoge eisen worden gesteld aan de Application Logic.

In de beschrijving wordt gesteld dat de applicatie altijd beschikbaar moet zijn. Deze eis geldt voornamelijk voor het operationele deel, aangezien men met het operationele deel direct wijzigingen door moet kunnen voeren. Dit houdt in dat de Application Logic erg robuust moet zijn. Deze robuustheid kan op een aantal manier bewerkstelligd worden. Allereerst door de Application Logic zorgvuldig te programmeren. Onzorgvuldigheden tijdens het ontwikkelproces kunnen danig afbreuk doen aan de robuustheid van een applicatie. Daarnaast kan overwogen worden om de Application Logic op meerdere machines te plaatsen, zodat de ene machine het over kan nemen als de andere er mee op houdt. Dit argument geldt overigens ook voor de database.

Er moeten meerdere gebruikers tegelijkertijd gebruik maken van de Application Logic, en de problemen die aan de Application Logic voorgelegd worden, kunnen vrij omvangrijk zijn. Daarom moet de Application Logic snel zijn en efficiënt omspringen met de beschikbare *resources*. Zoals eerder genoemd kan *parallel processing* in combinatie met *load balancing* gebruikt worden als extra maatregel om te zorgen dat de Application Logic de belasting aan kan. Ook is niet van tevoren bekend hoeveel gebruikers tegelijkertijd van de applicatie gebruik zullen maken. Het aantal simultane gebruikers kan sterk fluctueren. De *scalability* van de applicatie is daarom belangrijk. Bij deze aspecten kan middleware een grote hulp zijn.

Wat betreft het platform waarop de Application Logic uitgevoerd moet worden, geldt hetzelfde als bij de interface werd gezegd. Systeemtalen (behalve Java) dienen voor elk platform apart gecompileerd te worden. Aangezien voor de Application Logic meestal op voorhand bekend is welk(e) platform(s) gebruikt zullen worden, is dit van minder belang dan voor de interface.

De Application Logic moet communiceren met de interface en waarschijnlijk met andere applicaties. Daarom zijn de mogelijkheden om tussen die onderdelen te communiceren belangrijk. Ook moet rekening gehouden worden met een goede (modulaire) opzet van de code, omdat onderhoud en uitbreidingen aan de orde van de dag zullen zijn.

Voornamelijk gezien de nadruk die gelegd wordt op prestaties en robuustheid wordt voor de voorbeeld-applicatie gekozen voor een systeemtaal. Hierbij is Java wat minder aantrekkelijk vanwege de lagere executiesnelheid. Daarom, en aangezien de Application Logic uitgevoerd zal worden op het Windows Operating System, wordt ook hier gekozen voor Object Pascal.

8.2.5 Overzicht gemaakte keuzes

In de bovenstaande paragrafen zijn een aantal beslissingen genomen met betrekking tot de ontwikkeling van de in paragraaf 8.1 beschreven situatie. Allereerst werd besloten tot een opdeling in verschillende componenten en een multi-tier structuur met gerepliceerde Application Logic en gerepliceerde gegevens. Daarnaast werd besloten COM+ te gebruiken als middleware. Als communicatie-protocol werd gekozen voor TCP/IP met daar 'bovenop' het protocol dat door COM+ gebruikt wordt. De beveiliging zal gerealiseerd worden door een bestaand algoritme te implementeren. Voor de Interface is gekozen voor Object Pascal in combinatie met ActiveX. Ook voor de implementatie van de Application Logic zal Object Pascal gebruikt worden.

8.3 Afleiding van de methodiek

Mede op basis van hetgeen in paragraaf 8.2 is besproken, wordt in deze paragraaf een algemene methodiek afgeleid voor de besluitvorming omtrent de ontwikkeling van een gedistribueerde applicatie. De volgorde waarin de verschillende stappen van de besluitvorming doorlopen worden volgt het voorbeeld in de vorige paragraaf, maar is geenszins bedoeld om de indruk te wekken dat het de enig mogelijke volgorde betreft. De volgorde is, zoals ook al eerder is gesteld, vrij te kiezen, zolang er maar rekening wordt gehouden met de invloed tussen keuzes onderling.

Deze invloed wordt als volgt beschreven: de indeling in categorieën die in paragraaf 8.2 gehanteerd werd (algemeen, applicatie als geheel, Interface en Application Logic), wordt ook in deze paragraaf gebruikt. Voor elke categorie wordt aangegeven welke keuzes van toepassing zijn. Vervolgens wordt aangegeven welke van de in hoofdstuk 6 afgeleide criteria met deze keuzes gemoeid zijn. Tenslotte wordt beschreven hoe de verschillende criteria die bij de keuzes horen, elkaar beïnvloeden. Het betreft daarbij *positieve invloeden* en *negatieve invloeden*. Een positieve invloed geeft aan dat keuzes elkaar niet wederzijds beperken, terwijl een negatieve invloed aangeeft dat een bepaalde beslissing voor keuze A de mogelijkheden voor keuze B beperkt. Voor vrijwel alle combinaties van criteria zijn situaties voorstelbaar waarin de criteria een negatieve invloed ten opzichte van elkaar hebben. Daarom worden alleen de belangrijkste en algemeen geldige invloeden beschreven. Tenslotte wordt een algemene methodiek afgeleid op basis van deze stappen.

8.3.1 Algemeen

In paragraaf 8.2.1 werd al vermeld dat sommige aspecten betrekking hebben op alle stappen in het beslissingsproces. Daarom worden deze aspecten apart genoemd in deze paragraaf, in plaats van ze elke keer te vermelden. De keuzes die in deze paragraaf vallen hebben betrekking op:

1. Ontwikkelgemak
2. Time-to-market
3. Toegestane kosten
4. Gemak waarmee geleerd kan worden met de techniek te werken
5. Ondersteuning door anderen

De criteria waarop deze keuzes betrekking hebben, zijn:

Ad 1: Gebruiksgemak IDE,
Ad 2: Snelheid van ontwikkeling, leercurve
Ad 3: Prijs van ontwikkeling
Ad 4: Leercurve
Ad 5: Algemene ondersteuning

De volgende positieve en negatieve invloeden bestaan tussen de bovengenoemde criteria:

Criterium	Positieve invloed op…	Negatieve invloed op…
Snelheid van ontwikkeling	▪ Prijs van ontwikkeling	▪
Steile leercurve	▪ Snelheid van ontwikkeling	▪
Gebruiksgemak IDE	▪ Snelheid van ontwikkeling	▪
Algemene ondersteuning	▪ Snelheid van ontwikkeling	▪

8.3.2 Applicatie als geheel

Sommige beslissingen hebben invloed op de applicatie als geheel. Aspecten die met deze beslissingen te maken hebben worden hieronder beschreven.

1. Meerdere simultane gebruikers
2. Grote beschikbaarheid van resource-intensieve applicatie
3. Gegevensintensieve communicatie
4. Vertrouwelijkheid gegevens

Deze aspecten hebben betrekking op de volgende criteria:

Ad 1: Scalability
Ad 2: Load balancing, parallel processing
Ad 3: Communicatiesnelheid (TCP/IP, HTTP of middleware)

Ad 4: Beveiliging (authenticatie, autorisatie en encryptie)

De onderlinge invloeden tussen bovenstaande criteria zijn in onderstaande tabel beschreven.

Criterium	Positieve invloed op…	Negatieve invloed op…
Middleware	▪ Scalability ▪ Load balancing ▪ Parallel processing	▪ Communicatiesnelheid
TCP/IP	▪ Communicatiesnelheid	▪
Beveiliging	▪	▪ Communicatiesnelheid

8.3.3 Interface

Als we de afzonderlijke componenten van een gedistribueerde applicatie beschouwen, kunnen we ons richten op de Interface, de Application Logic en de gegevens en gegevensverschaffing. In deze paragraaf wordt de Interface behandeld, terwijl in de volgende twee paragrafen de Application Logic en gegevens(verschaffing) aan bod komen. De keuzes die betrekking hebben op de Interface zijn:

1. Keuze type interface
2. Performance van de Interface
3. Interface/Application Logic zijn verschillende technieken
4. Ondersteuning Platform/Webserver/Browser

Criteria die hiermee gemoeid zijn:

Ad 1: 'Browser interface' of een Graphical User Interface
Ad 2: Laadtijd, executiesnelheid
Ad 3: Compatibility
Ad 4: Portability (Operating System op de Client en de Server, Webserver en de Browser)

Een korte toelichting op een aantal van de bovenstaande punten:

- Als men kiest voor een 'stand-alone' Graphical User Interface, zijn scripting-technieken geen optie meer, omdat deze alleen binnen een browser uitgevoerd kunnen worden.
- Als de applicatie een geavanceerde interface moet hebben die functionaliteit biedt zoals *drag-and-drop*, moet men een systeemtaal gebruiken. Als dit in combinatie is met een browser-interface, en er niet voor Java gekozen wordt, moet men de systeemtaal in combinatie met ActiveX gebruiken. ActiveX kan (op dit moment) alleen in combinatie met het Windows Operating System en Internet Explorer gebruikt worden. Systeemtalen hebben over het algemeen een langere laadtijd dan scripting-technieken.

- Systeemtalen hebben over het algemeen een hogere executiesnelheid dan scripting-technieken.
- Als voor de implementatie van de Interface en de Application Logic voor verschillende technieken wordt gekozen, moeten de technieken wel de mogelijkheid bieden om de Interface en de Application Logic met elkaar te laten communiceren. Dit betreft het criterium 'compatibility'. Zie hiervoor paragraaf 8.3.4.

Onderlinge verbanden tussen deze criteria zijn:

Criterium	Positieve invloed op...	Negatieve invloed op...
Graphical User Interface (of browser interface met systeemtaal)	▪ Executiesnelheid	▪ Laadtijd ▪ Browser (ActiveX vereist Internet Explorer en Windows bij de client)

8.3.4 Application Logic

Bij de ontwikkeling van het Application Logic gedeelte spelen de volgende zaken een rol:

1. De Application Logic wordt gedeeld met andere applicaties
2. Grote beschikbaarheid
3. Veel gebruikers tegelijkertijd
4. Sterk variërend aantal simultane gebruikers
5. Uitvoeren gespecialiseerde taken
6. Platform
7. Veel onderhoud en uitbreidingen verwacht
8. Prestaties

Deze zaken houden verband met de volgende criteria:

Ad 1: Compatibility, modulariteit
Ad 2: Availability, robustness, efficiency
Ad 3: Parallel processing, load balancing
Ad 4: Scalability
Ad 5: Resource sharing, compatibility
Ad 6: Operating System op de Server
Ad 7: Modulariteit
Ad 8: Executiesnelheid, efficiency

Voor zover nodig volgt hieronder een korte toelichting bij een aantal punten:

- De aspecten bij de punten 2, 3, 4 en 5 kunnen aanleiding zijn om middleware te gebruiken.

- Voor sommige gespecialiseerde taken zijn computers gereserveerd. Applicaties die die speciale taken nodig hebben, kunnen dan gebruik maken van die computer. Dit heeft te maken met resource sharing en compatibility.
- Systeemtalen zijn over het algemeen makkelijker te onderhouden dan scripting-technieken.

Mogelijke invloeden tussen deze criteria zijn:

Criterium	Positieve invloed op…	Negatieve invloed op…
Middleware	▪ Compatibility, parallel processing, load balancing, scalability, availability, resource sharing.	▪

8.3.5 Gegevens(verschaffing)

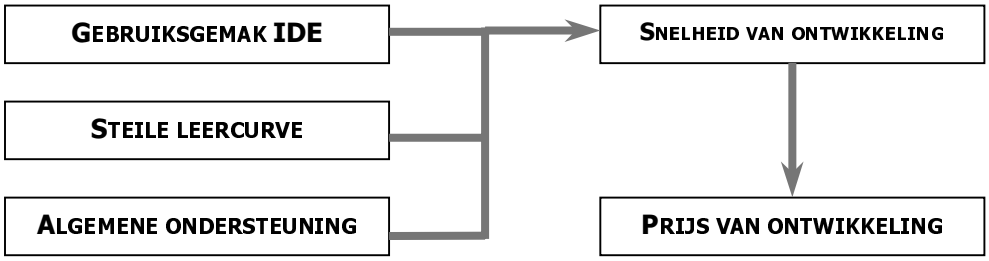
Wat betreft gegevens en gegevensverschaffing valt er niet zo heel veel te zeggen. Het belangrijkste aspect betreft de mogelijkheid om gegevens op te kunnen vragen uit een database, ze te kunnen aanpassen en te kunnen verwijderen. Dit vertaalt zich naar het criterium *database communicatie* in de scorekaart. De enige invloed die dit op het beslissingstraject kan hebben, is in hoeverre een techniek een gekozen type database ondersteunt. De moderne database-systemen bieden vergaande functionaliteit aan. Deze functionaliteit kan worden beschouwd als onderdeel van de Application Logic, en ook als zodanig behandeld worden.

8.3.6 Methodiek

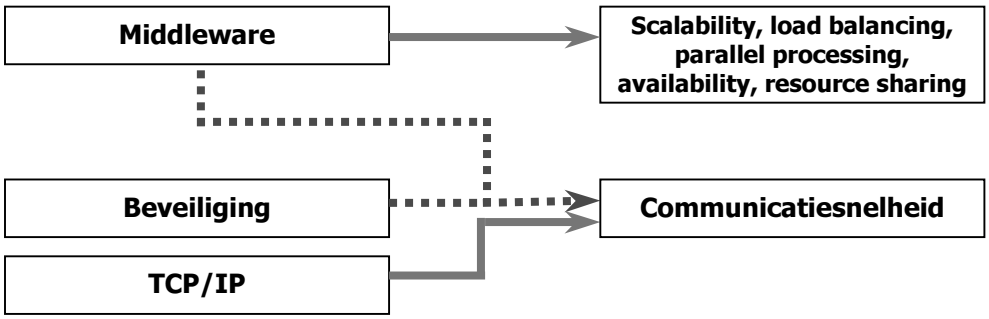
De methodiek die in deze paragraaf gepresenteerd wordt, bestaat voornamelijk uit een ‘stappenplan’. In dit stappenplan worden voor de verschillende beslissingsmomenten, zoals ze in het voorbeeld in paragraaf 8.2 de revue zijn gepasseerd, de verschillende mogelijkheden genoemd. Daarnaast wordt aangegeven welke omgevingsvariabelen invloed hebben op de keuze bij elk beslissingsmoment. Tevens wordt aangegeven welke beoordelingscriteria (zie hoofdstuk 6) betrokken zijn bij die beslissings-momenten. In de vorige paragrafen werden invloeden tussen beslissingsmomenten beschreven. Voordat het stappenplan uitgewerkt wordt, worden eerst de invloeden tussen de verschillende beslissingsmomenten behandeld.

De invloeden die in de vorige paragrafen zijn beschreven, zijn grafisch weergegeven in de onderstaande figuren. De rode stippellijnen geven negatieve invloeden weer, groene (ononderbroken) lijnen geven positieve

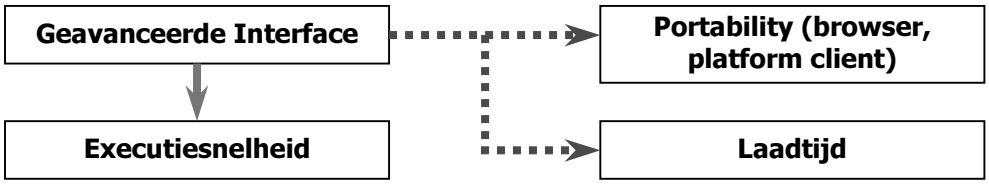
invloeden weer, de richting van de pijlen geeft aan in welke richting de invloed werkzaam is. Deze figuren zijn een belangrijk hulpmiddel bij het gebruik van het stappenplan.



Figuur 8.4: Algemeen



Figuur 8.5: Applicatie als geheel

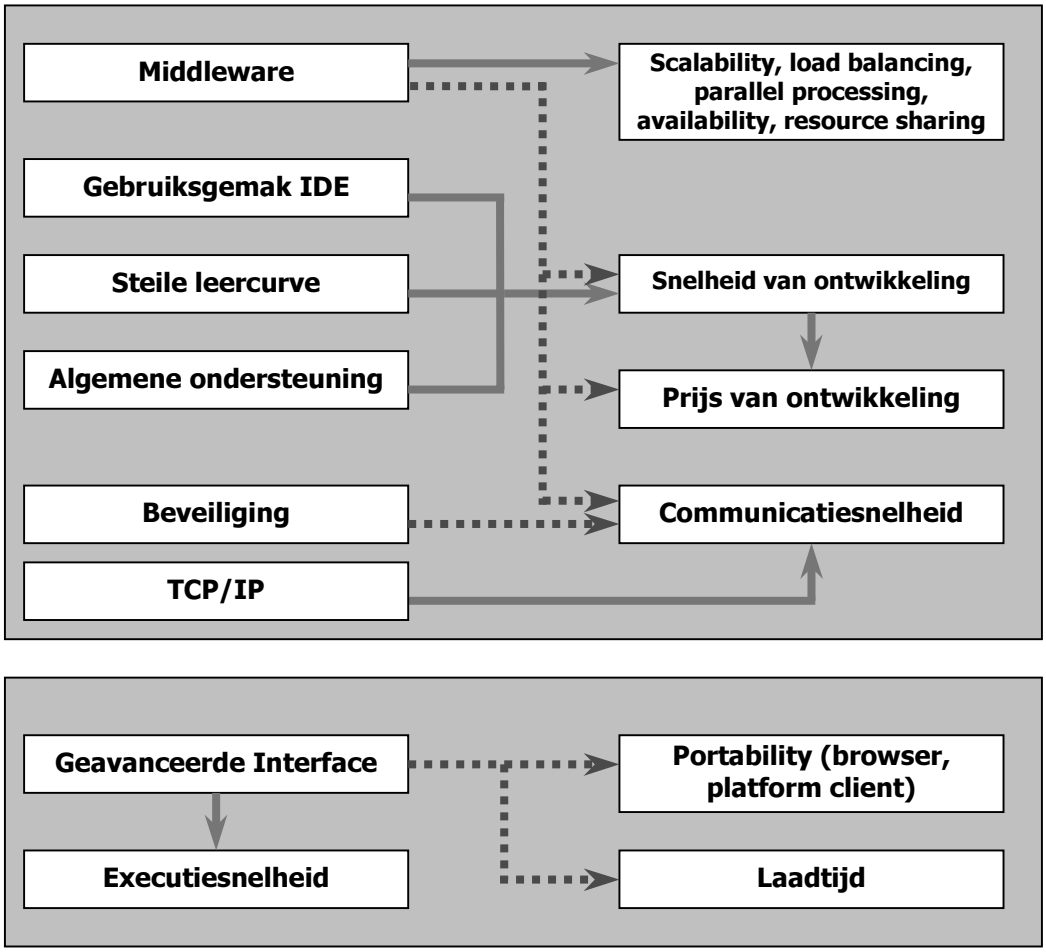


Figuur 8.6: Interface

Figuur 8.7: Application Logic

De invloeden zijn in bovenstaande figuren gerangschikt naar de categorieën waarin ze zijn behandeld. Van sommige beslissingen reikt de

invloed echter verder dan de categorie waarin die genomen worden. Dit komt tot uiting in het meermalen voorkomen van criteria in meerdere categorieën. Zo heeft het gebruik van middleware voor de Application Logic gevolgen voor de prijs van ontwikkeling voor de gehele applicatie. Om deze *trans-categoriële invloeden* in beeld te kunnen brengen, worden de bovenstaande figuren, voor zover van toepassing, samengevoegd in één figuur.



Figuur 8.8: Alle invloeden in één figuur

Bij het gebruiken van het stappenplan dient rekening gehouden te worden met bovenstaande invloeden. Enkele van deze worden expliciet in het stappenplan genoemd, maar impliciet zijn alle invloeden van toepassing. Analoog aan het voorbeeld worden de stappen voor de applicatie als geheel genoemd, voordat op de afzonderlijke onderdelen 'ingezoomd' wordt. We beginnen echter met een aantal algemene opmerkingen

8.3.6.1 Stap 1 – Algemeen

Net zoals in het voorbeeld betreft het hier enkele punten die betrekking hebben op applicatieontwikkeling in het algemeen. Indien voor de

applicatie een strakke deadline is gesteld, is *snelheid van ontwikkeling* van belang. Als voornamelijk geldt dat de ontwikkeling gemakkelijk moet zijn, moet gekeken worden naar het *gebruiksgemak van de IDE*. Een krap budget is gebaat bij een lage *prijs van ontwikkeling*. Als de beschikbare tijd voor ontwikkeling kort is, of snel een geavanceerde applicatie gebouwd moet worden, moet de *leercurve* in de gaten gehouden worden. Tenslotte kan een goede *algemene ondersteuning* bijdragen aan de snelheid en de kwaliteit van ontwikkeling.

8.3.6.2 Stap 2 – Applicatie als geheel

- In hoofdstuk 3 is beargumenteerd dat een multi-component en multi-tier structuur een goede basis is voor de bouw van een gedistribueerde applicatie. Het kan zijn dat de huidige functionele eisen de noodzaak voor een dergelijke uitgebreide structuur tegenspreken, maar de mogelijkheid van toekomstige uitbreidingen kan toch pleiten voor het gebruik van deze structuur. De indeling in componenten dient te geschieden door de Interface, Application Logic en Gegevens(verschaffing) los te koppelen.
- Parallel processing en load balancing kunnen redenen zijn om de Application Logic en gegevens te repliceren.
- Naarmate de applicatie complexer wordt en geavanceerde functionaliteit als resource sharing e.d. moet bieden, wordt het aantrekkelijker om gebruik te maken van middleware. De keuze tussen de verschillende vormen hangt voornamelijk af van de gebruikte techniek en platform. Het gebruik van COM+ beperkt zich tot het Windows Operating System, RMI is beperkt tot Java. Het is mogelijk om de verschillende vormen van middleware samen te laten werken.
- Indien de communicatie tussen de verschillende componenten van de applicatie gegevensintensief zal zijn, is het verstandig om voor een snel protocol te kiezen. TCP/IP is hierbij de beste keuze.
- Wat betreft beveiliging moet de keuze worden gemaakt tussen SSL (of vergelijkbare methoden) en een eigen implementatie. SSL is de meest geavanceerde vorm en is daarom aan te raden indien de grootst mogelijke veiligheid vereist is. Sleutelbeheer wordt door SSL zelf geregeld. Wel dient elke gebruiker een ‘certificate’ te installeren, waardoor het gebruik van SSL bewerkelijk en kostbaar wordt. Eigen implementatie biedt een wat lager niveau van veiligheid, waarbij een verkeerde wijze van sleutelbeheer en –verspreiding de genomen maatregelen volkomen teniet kan doen.

8.3.6.3 Stap 3 – Interface

- Indien voor een stand-alone interface gekozen wordt, zijn systeemtalen de enige mogelijkheid. Voor een browser-interface zijn zowel scripting-technieken als systeemtalen (eventueel in combinatie met ActiveX) mogelijk. Scripting-technieken scoren

beter op het gebied van laadtijd, terwijl de executiesnelheid van systeemtalen hoger ligt.

- Een geavanceerde browser-interface met bijvoorbeeld *drag-and-drop* functionaliteit vereist het gebruik van een systeemtaal. Hiervoor kan Java gebruikt worden, of een andere systeemtaal in combinatie met ActiveX. ActiveX beperkt de applicatie wel tot Microsoft Internet Explorer en het Windows platform. Java kent op dat vlak geen beperkingen. Wel zijn de prestaties minder.

8.3.6.4 Stap 4 – Application Logic

- De Application Logic vormt een belangrijk onderdeel van de applicatie. Het is daarom belangrijk dat de Application Logic robuust is en een hoge beschikbaarheid heeft. Dit kan aanleiding zijn om de Application Logic te repliceren en een vorm van load balancing te gebruiken. Ook parallel processing is hierbij een optie.
- Als vooraf niet bekend is hoeveel gebruikers simultaan van de Application Logic gebruik zullen maken, of als het aantal gebruikers sterk kan fluctueren, is scalability een belangrijk punt.
- Als de applicatie aspecten als parallel processing en resource sharing moet gebruiken vanwege meerdere gebruikers en meerdere locaties, is het aan te raden middleware te gebruiken. Zonder middleware zijn deze aspecten zeer lastig te implementeren.

8.3.7 Conclusie

Op basis van een praktijkvoorbeeld is in dit hoofdstuk een methodiek afgeleid waarmee de nodige beslissingen omtrent structuur en implementatie van een te ontwikkelen gedistribueerde applicatie genomen kunnen worden. Deze methodiek bestaat uit twee delen. Het eerste deel is een stappenplan waarin beschreven wordt welke beslissingen genomen moeten worden en welke factoren invloed hebben op die beslissingen. In dit stappenplan wordt de applicatie vanuit verschillende invalshoeken bekeken. De stappen (invalshoeken) zijn:

1. *Algemeen*
2. *Applicatie als geheel*
3. *Interface*
4. *Application Logic*

Het tweede deel beschrijft de invloeden die tussen die factoren onderling bestaan. Deze invloeden zijn in de figuren 8.4 tot en met 8.8 weergegeven. Deze figuren zijn een belangrijk gereedschap bij het gebruik van het stappenplan.

9 CONCLUSIES EN AANBEVELINGEN

Het doel van deze scriptie is een invulling te geven aan de discipline van Software Engineering van Gedistribueerde Applicaties. In dit hoofdstuk wordt de scriptie afgesloten door te evalueren in welke mate aan deze doelstelling is voldaan. Tenslotte worden enkele aanbevelingen gedaan met betrekking tot nader onderzoek.

9.1 Conclusies

De genoemde evaluatie vindt plaats door na te gaan in hoeverre antwoord is gegeven op de vragen die in de probleemstelling zijn geformuleerd. De probleemstelling van deze scriptie werd in hoofdstuk 1 geformuleerd en luidt:

> Wat moet worden verstaan onder Software Engineering van Gedistribueerde Applicaties? Meer specifiek gaat het om de beantwoording van de volgende vragen:
>
> - Welke criteria zijn van belang voor de beoordeling van ontwikkeltechnieken van gedistribueerde applicaties?
> - Welke procedure kan worden gevolgd om – op basis van deze criteria – een beslissing te nemen ten aanzien van de te gebruiken ontwikkeltechniek?
> - Welke methodiek kan worden afgeleid ten behoeve van de ontwikkeling van gedistribueerde applicaties?

Voor de beoordeling van de mate waarin de scriptie heeft voldaan aan zijn doelstelling wordt voor de afzonderlijke deelvragen bekeken in welke mate ze zijn beantwoord.

9.1.1 Welke criteria zijn belang voor de beoordeling van ontwikkeltechnieken?

Om criteria voor de beoordeling van ontwikkeltechnieken af te kunnen leiden is in hoofdstuk 2 gekeken naar de gevolgen die de gewenste functionaliteit van gedistribueerde applicaties heeft voor de ontwikkeling daarvan. Criteria werden afgeleid op basis van functionaliteiten en prestatie-eisen van gedistribueerde applicaties. In hoofdstuk 3 werden criteria afgeleid die betrekking hebben op de communicatie tussen de componenten van een gedistribueerde applicatie. In hoofdstuk 4 werden criteria geformuleerd gebaseerd op beveiligingsaspecten. In hoofdstuk 5 gebeurde tenslotte hetzelfde met betrekking tot algemene Software Engineering eisen en eisen ten aanzien van de ontwikkeling. Dit alles resulteerde in de *scorekaart* die in hoofdstuk 6 werd samengesteld. Deze scorekaart is het eerste belangrijke resultaat van deze scriptie. In het algemeen zal deze verzameling van criteria voldoende zijn om een gemotiveerd oordeel te kunnen geven over de verschillende

ontwikkeltechnieken. Hierbij moet echter wel rekening gehouden worden met de tijdgebondenheid van deze scriptie. Toekomstige ontwikkelingen met betrekking tot de mogelijkheden van ontwikkeltechnieken en veranderingen in de gewenste functionaliteiten van gedistribueerde applicaties, kunnen invloed hebben op de toepasbaarheid van de geformuleerde criteria. Het valt echter buiten het bereik van deze scriptie om voor elke denkbare situatie een voorschrift te geven. De afleiding van de criteria biedt voldoende basis voor de ontwikkelaar om zelf een set criteria samen te stellen voor de betreffende situatie.

9.1.2 Welke beslissingsprocedure kan worden gevolgd?

De procedure om tot een beslissing te komen ten aanzien van de te gebruiken ontwikkeltechniek is in twee delen uiteengevallen. Het ene deel is de beoordeling van een techniek op basis van de criteria die zijn afgeleid in de hoofdstukken 2 tot en met 6. Het andere deel betreft de wijze waarop deze beoordeling invloed uitoefent op de daadwerkelijke beslissing.

Het beoordelingsproces is in hoofdstuk 7 op vrij summiere wijze behandeld door de beoordeling van één techniek te beschrijven en de resultaten van nog zeven andere technieken te geven. De reden voor de beperkte beschrijving ligt in het feit dat de beoordeling van een ontwikkeltechniek voor veel criteria vaak een subjectieve keuze betreft. Dit heeft tot gevolg dat het beoordelingsproces voor verschillende personen vaak in meer of mindere mate afwijkend kan verlopen. Deze subjectiviteit is een beperking op de algemene geldigheid van dit onderdeel van deze scriptie.

De wijze waarop de beoordeling van een ontwikkeltechniek invloed uitoefent op de uiteindelijke beslissing is onderdeel van de methodiek die beschreven is in hoofdstuk 8. Bij deze methodiek werd behandeld welke criteria gemoeid waren in welke beslissingen. Vervolgens kan met behulp van de beoordeling zoals die in hoofdstuk 7 is gegeven, de geschiktheid van verschillende technieken geëvalueerd worden.

9.1.3 Welke methodiek kan worden afgeleid?

Aan de hand van de functionele specificaties van een (deels fictieve) applicatie werd in hoofdstuk 8 een realistisch beslissingsproces voor de ontwikkeling van een gedistribueerde applicatie beschreven. De verschillende onderdelen van dit beslissingsproces werden geïdentificeerd en beschreven, waarna de invloeden die tussen de verschillende aspecten van die onderdelen bestaan werden behandeld. Daarna werden de onderdelen in een *stappenplan* ‘gegoten’, waarmee het beslissingstraject op een redelijk gestructureerde wijze doorlopen kan worden. Samen met de beschreven onderlinge invloeden, weergegeven in de figuren 8.4 tot en

met 8.8, vormt dit stappenplan een *methodiek* voor de ontwikkeling van gedistribueerde applicaties. Deze methodiek is het tweede belangrijke resultaat van deze scriptie. Het feit dat de onderdelen van het beslissingsproces elkaar wederzijds beïnvloeden heeft tot gevolg dat er geen eenduidige volgorde aangegeven kan worden waarin de stappen doorlopen moeten worden. Afhankelijk van de situatie, de functionele specificaties van de applicatie en de voorkeuren en eigenschappen van de beslissingnemers, zal namelijk een andere volgorde gekozen worden. Dit heeft ook invloed op de mate van 'back-tracking' die plaats zal vinden tijdens het beslissingsproces. Dit komt het gebruiksgemak van de methodiek niet ten goede, omdat voor elke situatie afzonderlijk bepaald moet worden wat de ideale volgorde van beslissingen is. De gekozen aanpak heeft echter wel een positieve invloed op de algemene bruikbaarheid van de methodiek.

Zoals uit bovenstaande blijkt zijn subjectiviteit van de beslissingnemer (of beoordelaar) en de invloed van omgevingsvariabelen beperkende factoren op de algemeenheid van deze scriptie. Door te abstraheren van praktische situaties is getracht deze algemene geldigheid toch zoveel mogelijk te behouden.

9.2 Aanbevelingen

Deze scriptie heeft geprobeerd een overzicht te geven van de aspecten die spelen bij het ontwikkelen van gedistribueerde applicaties. Deze aanpak is wel ten koste gegaan van de diepgang die noodzakelijk is voor een grondig begrip van elk aspect. Veel van de onderwerpen zijn zo uitgebreid dat het mogelijk is een aparte scriptie over dat onderwerp alleen te schrijven. Zo kan er dieper worden ingegaan op de voor- en nadelen van gedistribueerde applicaties, en of die in alle situaties te behalen zijn. Ook is het mogelijk te onderzoeken of het mogelijk is om de prestaties op te voeren. Daarnaast is bij de behandeling van de mogelijke structuren van gedistribueerde applicaties een vrij grove opdeling behandeld. Verder onderzoek zou zich kunnen richten op een fijnere verdeling van de applicatie (Component Based Development). Ook zou hierbij rekening gehouden kunnen worden met de invloed van de gewenste functionaliteit op verschillende structuren.

In hoofdstuk 4 werd reeds genoemd dat beveiliging een zeer diepgaand onderwerp is dat een hoge mate van expertise vereist. In deze scriptie is niet gezegd hoe beveiliging precies geïmplementeerd kan worden in een gedistribueerde applicatie. Dit zou een gebied kunnen zijn voor verder onderzoek. Wat betreft de Software Engineering aspecten die in hoofdstuk 5 zijn genoemd kan worden onderzocht wat de verschillen zijn tussen deze aspecten indien toegepast op stand-alone applicaties en op gedistribueerde applicaties. Hetzelfde geldt voor ontwikkelingsaspecten.

Ook op het gebied van de geformuleerde criteria is nog veel onderzoek mogelijk. Zo kan onderzocht worden of er een algemene methode voor de beoordeling van de verschillende technieken mogelijk is, zodat de negatieve invloed die uitgaat van de subjectiviteit van de beoordelaar, geneutraliseerd kan worden. Ook kan verder onderzoek naar de invloeden tussen de criteria verricht worden.

Zoals reeds in de inleiding werd gezegd, is deze scriptie meer een verkenning in de breedte, en niet in de diepte. Deze scriptie leent zich zodoende niet voor het ontwikkelen van een expertise op één of meer van de deelgebieden die hier zijn behandeld. Daarvoor wordt dan ook geadviseerd andere bronnen te raadplegen (zie hiervoor de referenties).

Een andere beperking van deze scriptie ligt in het theoretische karakter. De informatie over elk van de behandelde onderwerpen is dan wel beproefd in de praktijk, maar de synthese hiervan, zoals in deze scriptie is bewerkstelligd, is nog onbeproefd, voor zover het de auteur bekend is. Hoewel de verschillende aspecten wel in praktijk gebruikt worden, is het toch belangrijk om de combinatie van deze aspecten, resulterend in de methodiek van hoofdstuk 8, in de praktijk te toetsen. Op basis van deze toetsing kan de methodiek, waar nodig, bijgesteld worden.

BRONNEN EN REFERENTIES:

De literatuurverwijzingen die in de tekst van deze scriptie voorkomen, staan bij de bronnen vermeld. Bij referenties staan verwijzingen naar literatuur voor de belangrijkste onderwerpen in deze scriptie.

Bronnen

Verwijzing	Bron
[COUL, 1988]	George F. Coulouris en Jean Dollimore, Distributed Systems, 1988
[STEE, 1995]	Maarten van Steen en Henk Sips, Computer and Network Organization, 1995
[TANE, 1995]	Andrew Tanenbaum, Computer Networks, 3rd edition, 1995
[SZYP, 1998]	Clemens Szyperski, Component Software, 1998
[OMG, 2000]	Object Management Group website: http://www.omg.com
[MS, 2000]	Microsoft website: http://www.microsoft.com
[SUN, 2000]	Sun Microsystems website: http://www.sun.com
[JSOFT, 2000]	JavaSoft website: http://www.javasoft.com
[RIDD, 1997]	Ed Ridderbeekx, doctoraalscriptie, Internet, World Wide Web en Beveiliging, 1997, http://www.few.eur.nl/few/people/jvandenberg/master_theses/mark.pdf
[VLIE, 1993]	Hans van Vliet, Software Engineering, Principles and Practice, 1993
[IEEE, 1983]	IEEE Standard Glosary of Software Engineering Terminology, 1983
[KILS, 1999]	Mark Kilsdonk e.a., Beveiligingsrisico's van Java-applets, Informatie, feb. 1999, p.p. 18-29
[BORL, 2000]	Borland website: http://www.borland.com

Referenties

Onderwerp	Referentie
'Distributed processing'	George F. Coulouris en Jean Dollimore, Distributed Systems, 1988
Gedistribueerde architecturen	Robert Orfali e.a., Client/Server Survival Guide, Third Edition, 1999
Object Oriented Programming	Timothy Budd, An Introduction to Object-Oriented Programming, 1997
Communicatie architectuur	Andrew Tanenbaum, Computer Networks, 3rd edition, 1995
COM+	Rosemary Rock-Evans, DCOM Explained, 1998
CORBA	Robert Orfali e.a., Instant CORBA, 1997
RMI	Daniel J. Berg e.a., Advanced Techniques for Java Developers, 1999
Beveiliging	Ed Ridderbeekx, Internet, World Wide Web en Beveiliging, 1997
Software Engineering	Shari L. Pfleeger, Software Engineering: Theory and Practice, 1998
Object Pascal (Delphi)	Steve Teixeira e.a., Delphi 5 Developer's Guide, 1999
Java	Cay S. Horstmann e.a., Core Java 2, Volume 1: Fundamentals, 1998
C++	Gregory Satir e.a., C++: the Core Language, 1995
Visual Basic	Michael Halvorson, MS Visual Basic 6.0 Professional (Step by Step), 1998
CGI	Jacqueline D. Hamilton, CGI Programming 101, 2000
ASP	Alex Homer e.a., Professional Active Server Pages 3.0, 1999
JSP	Andrew Patzer e.a., Professional Java Server Programming, 1999
ColdFusion	Ben Forta e.a., The ColdFusion 4.0 Web Application Construction Kit, 1998

APPENDIX A: PUBLIC KEY ENCRYPTIE

Zoals in hoofdstuk 4 reeds is verteld, heeft Public Key Encryptie grote potentie op het gebied van authenticatie en het versturen van versleutelde berichten (privacy). De grootste kracht van Public Key Encryptie ligt echter op het vlak van sleutelbeheer. De Public Key Infrastructuur zorgt hiervoor. In de volgende voorbeelden wordt deze potentie duidelijk gemaakt. Maar ook de gevaren die er toch nog aan vastzitten, en mogelijke oplossingen hiervoor worden aangedragen. De notatie $\{iets\}_{key}$ betekent dat 'iets' versleuteld of ontcijferd is door middel van 'key'.

Gebruik van Public Key Encryptie voor authenticatie

Stel dat Alice zeker wil weten dat Bob is wie hij zegt dat hij is. Bob heeft een public key en een private key. Bob zorgt dat Alice zijn public key weet (hoe dat gebeurt wordt verderop behandeld). Alice genereert dan een willekeurig bericht, en stuurt dit naar Bob:

A → B	random-bericht

Bob gebruikt dan zijn private key om het bericht te versleutelen en stuurt dat terug naar Alice:

B → A	$\{random\text{-}bericht\}_{bobs\text{-}private\text{-}key}$

Alice ontvangt dit bericht en ontcijfert het met behulp van Bob's public key, die ze al wist. Ze vergelijkt het ontcijferde bericht met wat ze heeft verstuurd. Als ze overeen komen, weet ze dat ze met Bob praat, want een bedrieger zou Bob's private key niet weten, en zou daarom niet het random-bericht op de juiste wijze kunnen versleutelen.

Maar er zit meer aan vast. Want tenzij Bob precies weet wat hij versleutelt, is het nooit een goed idee om iets met zijn private key te versleutelen en het dan naar iemand anders te sturen. De versleutelde gegevens kunnen namelijk tegen hem gebruikt worden. Hij is namelijk de enige die de gegevens versleuteld kan hebben. Stel dat Alice kwaad zou willen en een schadelijk bericht als random-bericht naar Bob stuurt. Bob versleutelt het met zijn sleutel, Alice stuurt het door, en er is geen manier om te bewijzen dat Bob dat bericht niet heeft verstuurd.

Dus in plaats van het versleutelen van het originele bericht van Alice, maakt Bob een message digest van dat bericht, versleutelt dat (zie hiervoor paragraaf 4.2.1.2) en stuurt het naar Alice. Alice kan dat ontcijferen en vergelijken met de waarde die ze zelf kan berekenen. Deze techniek staat bekend als een digitale handtekening.

Maar als Bob een bericht ‘tekent’ dat door Alice gegenereerd is, doet hij iets dat net zo gevaarlijk is als het versleutelen van een random-bericht van Alice. Een uitbreiding is dus gewenst: een gedeelte (of alles) van de gegevens die verstuurd worden voor het authenticeren van Bob moeten gegenereerd zijn door Bob zelf:

A → B	Hallo, ben jij bob?
B → A	Alice, hier is Bob $\{digest[Alice, hier is Bob]\}_{bobs\text{-}private\text{-}key}$

Als Bob dit protocol gebruikt, weet Bob welk bericht hij stuurt, en het is niet erg als hij het ‘tekent’. Bob stuurt de niet-versleutelde versie van het bericht eerst, “Alice, hier is Bob” en daarna stuurt hij de digested-versleutelde versie. Alice kan makkelijk nagaan dat Bob Bob is, en Bob heeft niets getekend wat hij niet wil.

Verspreiden van Public Keys

Hoe kan Bob nou zijn public key op een vertrouwenswaardige manier afgeven aan Alice. Stel dat het authenticatie protocol er zo uitziet:

A → B	Hallo
B → A	Hoi, ik ben Bob bobs-public-key
A → B	Bewijs het maar
B → A	Alice, hier is Bob $\{digest[Alice, hier is Bob]\}_{bobs\text{-}private\text{-}key}$

Als het op deze manier gebeurt, kan iedereen zeggen dat hij Bob is. Het enige wat nodig is, zijn een public key en een private key. Een bedrieger liegt tegen Alice en zegt dat hij Bob is, en het enige wat hij hoeft te doen is zijn public key opgeven in plaats van die van Bob. Daarna bewijst de bedrieger het door iets te versleutelen met zijn private key, en Alice kan niet weten dat hij niet Bob is.

Om dit probleem op te lossen is een object uitgevonden wat een certificate wordt genoemd. Een certificate heeft de volgende inhoud:

- de naam van de instantie die het certificate verschaft
- de entiteit voor wie het certificate wordt gebruikt (m.a.w. het subject)
- de public key van het subject
- een aantal timestamps

Het certificate is getekend met behulp van de private key van de verschaffer van het certificate. Iedereen weet de public key van de verschaffer (dit houdt in, de verschaffer van het certificate heeft zelf een certificate, et cetera). Certificates zijn een standaard manier om public keys aan namen te koppelen. Door gebruik te maken van deze certificate technologie kan iedereen Bob’s

certificate bestuderen om na te gaan of het vervalst is. Indien aangenomen wordt dat Bob goed let op zijn private key en dat het werkelijk Bob is voor wie het certificate is gaat alles goed. Het protocol werkt dan op de volgende manier:

A → B	Hallo
B → A	Hoi, ik ben Bob bobs-certificate
A → B	Bewijs het maar
B → A	Alice, hier is Bob $\{digest[Alice, hier is Bob]\}_{bobs\text{-}private\text{-}key}$

Als Alice nu Bob's eerste bericht krijgt, kan ze het certificate bestuderen, de handtekening controleren (zoals hierboven beschreven via een digest en public key encryptie), en dan het subject controleren (dit houdt in Bob's naam) en zien dat het inderdaad Bob is. Ze kan er dan van uitgaan dat de public key Bob's public key is en hem vragen zich te identificeren op de hierboven omschreven manier.

Een bedrieger, zeg dat hij Mallet heet, kan dan wel het volgende doen:

A → M	Hallo
M → A	Hoi, ik ben Bob bobs-certificate
A → M	Bewijs het maar
M → A	?????

Maar Mallet kan niet aan het laatste verzoek van Alice voldoen omdat hij niet Bob's private key heeft.

Van Public naar Secret

Als Alice Bob eenmaal heeft geauthenticeerd, kan ze hem een bericht sturen dat alleen hij kan ontcijferen:

A → B	$\{geheim\}_{bobs\text{-}public\text{-}key}$

De enige manier om bovenstaan geheim te kunnen lezen is door het te ontcijferen met Bob's private key. Het geheim kan dan ook als sleutel worden gebruikt voor een symmetrisch encryptie algoritme als DES, RC4 of IDEA. Alice en Bob zijn de enigen die 'geheim' weten, dus kunnen ze dat als sleutel gebruiken, op de volgende manier:

A → B	Hallo
B → A	Hoi, ik ben Bob bobs-certificate
A → B	Bewijs het maar

B → A	Alice, hier is Bob $\{digest[Alice, hier is Bob]\}_{bobs\text{-}private\text{-}key}$
A → B	Ok, Bob, hier is een geheim $\{geheim\}_{bobs\text{-}public\text{-}key}$
B → A	$\{een\ bericht\}_{geheim\text{-}key}$

Hoe de geheim-key wordt berekend ligt aan het protocol dat afgesproken wordt, maar het kan gewoon een kopie van 'geheim' zijn.

'Pitfalls'

Maar er kan nog meer gebeuren. De bedrieger, Mallet, kan niet ontdekken welk geheim Alice en Bob uitgewisseld hebben, maar hij kan wel de communicatie verstoren. Als Mallet bijvoorbeeld tussen Alice en Bob in gaat zitten, kan hij de meeste informatie doorsturen, maar bepaalde berichten vervormen (makkelijk voor hem als hij weet welk protocol Alice en Bob gebruiken):

A → M	Hallo
M → B	Hallo
B → M	Hoi, ik ben Bob bobs-certificate
M → A	Hoi, ik ben Bob bobs-certificate
A → M	Bewijs het maar
M → B	Bewijs het maar
B → M	Alice, hier is Bob $\{digest[Alice, hier is Bob]\}_{bobs\text{-}private\text{-}key}$
M → A	Alice, hier is Bob $\{digest[Alice, hier is Bob]\}_{bobs\text{-}private\text{-}key}$
A → M	Ok, Bob, hier is een geheim $\{geheim\}_{bobs\text{-}public\text{-}key}$
M → B	Ok, Bob, hier is een geheim $\{geheim\}_{bobs\text{-}public\text{-}key}$
B → M	$\{een\ bericht\}_{geheim\text{-}key}$
M → A	Vervormd [$\{een\ bericht\}_{geheim\text{-}key}$]

Mallet stuurt alle gegevens heen en weer totdat Alice en Bob een geheim delen. Dan vervormt Mallet Bob's bericht aan Alice. Op dat moment vertrouwt Alice Bob, dus neemt ze het vervormde bericht serieus. Mallet weet niet het geheim. Het enige wat hij kan doen is de gegevens die versleuteld zijn met de geheim-key beschadigen. Afhankelijk van het protocol kan het zijn dat Mallet geen geldig bericht produceert, maar hij zou ook geluk kunnen hebben.

Om dat te voorkomen, kunnen Alice en Bob een Message Authentication Code (MAC) introduceren in hun protocol. Een MAC is een stuk data dat berekend is met behulp van een geheim en wat verzonden gegevens. Het

digest algoritme dat hierboven beschreven is heeft precies de juiste eigenschappen voor het bouwen van een MAC:

MAC = digest[een bericht, geheim]

Omdat Mallet het geheim niet weet, kan hij niet de digest berekenen. Zelfs als Mallet op random wijze berichten vervormt, is zijn kans op succes klein als de digest data groot is. Het werkt dan op de volgende manier:

A → B	Hallo
B → A	Hoi, ik ben Bob bobs-certificate
A → B	Bewijs het maar
B → A	Alice, hier is Bob $\{digest[Alice, hier\ is\ Bob]\}_{bobs\text{-}private\text{-}key}$
A → B	Ok Bob, hier is een geheim $\{geheim\}_{bobs\text{-}public\text{-}key}$
B → A	$\{een\ bericht, MAC\}_{geheim\text{-}key}$

Nu kan Mallet wel berichten vervormen, maar door de MAC kunnen Alice en Bob ontdekken dat het bericht veranderd was en ophouden met praten.

Tenslotte kan Mallet nog berichten opnemen. Hij kan ze dan wel niet begrijpen, maar hij kan ze herhalen, opnieuw opsturen. Zo kan hij aardig wat rottigheid uithalen als hij tussen Alice en Bob in zit. De oplossing is om random-elementen van beide kanten in de conversatie op te nemen. De kans is dan klein dat een bericht hetzelfde random-element zal hebben, dus als dat gebeurt, is het waarschijnlijk een herhaling, en kan dus weggegooid worden.

APPENDIX B

In deze appendix worden enkele technieken toegelicht die een toegevoegde waarde hebben voor gedistribueerde applicaties en de ontwikkeling daarvan. Allereerst wordt een toelichting gegeven op de verschillende vormen van middleware. Daarna worden transactie-ondersteunende technieken behandeld, waarna tenslotte nog wat mogelijkheden voor gegevens-representatie genoemd worden.

Middleware

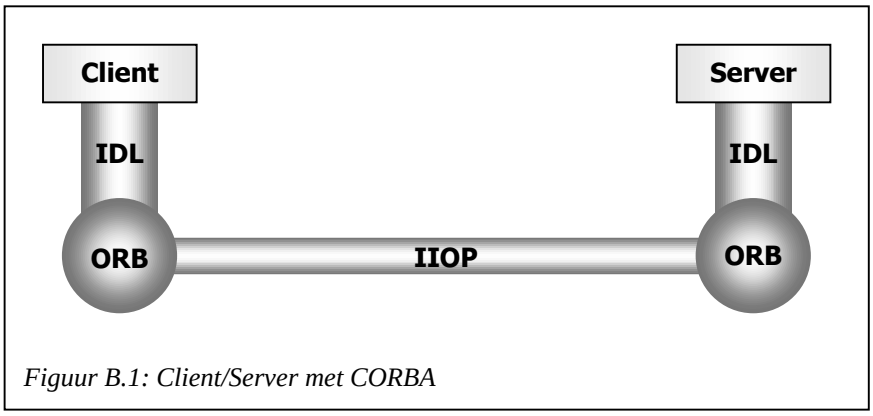
Middleware is een algemene term voor elk programma dat dient om de communicatie tussen twee aparte en meestal al bestaande programma's mogelijk te maken of te verbeteren. Een veel voorkomende vorm van middleware is om programma's die geschreven zijn om een bepaalde database te benaderen, de mogelijkheid te verschaffen toegang te krijgen tot andere databases. In het kader van het Internet wordt middleware veelal beschouwd als medium tussen client(s) en server: middleware is de "/" in client/server. Hieronder worden de meest voorkomende en ondersteunde vormen van middleware beschreven.

CORBA

CORBA (Common Object Request Broker Architecture) is een architectuur en specificatie voor het creëren, distribueren en beheersen van gedistribueerde programma-objecten in een netwerk. CORBA is ontwikkeld door de Object Management Group (OMG). Het geeft programma's op verschillende locaties en ontwikkeld door verschillende fabrikanten de mogelijkheid om met elkaar te communiceren in een netwerk door middel van een "interface broker". ISO (International Organization for Standardization) heeft gedeclareerd dat CORBA de standaard architectuur voor gedistribueerde objecten (ook wel: componenten) is.

Het essentiële concept in CORBA is de Object Request Broker (ORB). ORB ondersteuning in een netwerk van clients en servers op verschillende computers betekent dat een client programma (dat zelf een object kan zijn) diensten van een server programma of object kan aanvragen zonder dat het hoeft te weten waar de server in een gedistribueerd netwerk is of hoe de interface voor het server-programma in elkaar steekt. De ORB's communiceren met elkaar door middel van het General Inter-ORB Protocol (GIOP) en, voor het Internet, door middel van het Internet Inter-ORB Protocol (IIOP). IIOP vertaalt GIOP berichten naar TCP (Transmission Control Protocol). Een belangrijk concept binnen CORBA is de Interface Definition Language (IDL). De interface van een object wordt gedefinieerd op een taal-onafhankelijke manier, namelijk met behulp van de IDL, een definitie-taal die qua syntax veel op C lijkt. Door

het gebruik van IDL kunnen objecten geschreven in verschillende talen met elkaar communiceren.



Figuur B.1: Client/Server met CORBA

COM

COM staat voor Component Object Model, en is Microsoft's opzet voor het ontwikkelen en ondersteunen van component objecten. Het richt zich op het verschaffen van dezelfde mogelijkheden als CORBA. COM verschaft diensten voor onder andere het bemiddelen van een bepaalde interface tussen componenten en het bepalen wanneer een component van het systeem verwijderd kan worden (life cycle management). COM richt zich niet op gedistribueerde client/server interfaces, maar meer op component-interfaces voor clients en servers die op dezelfde computer draaien.

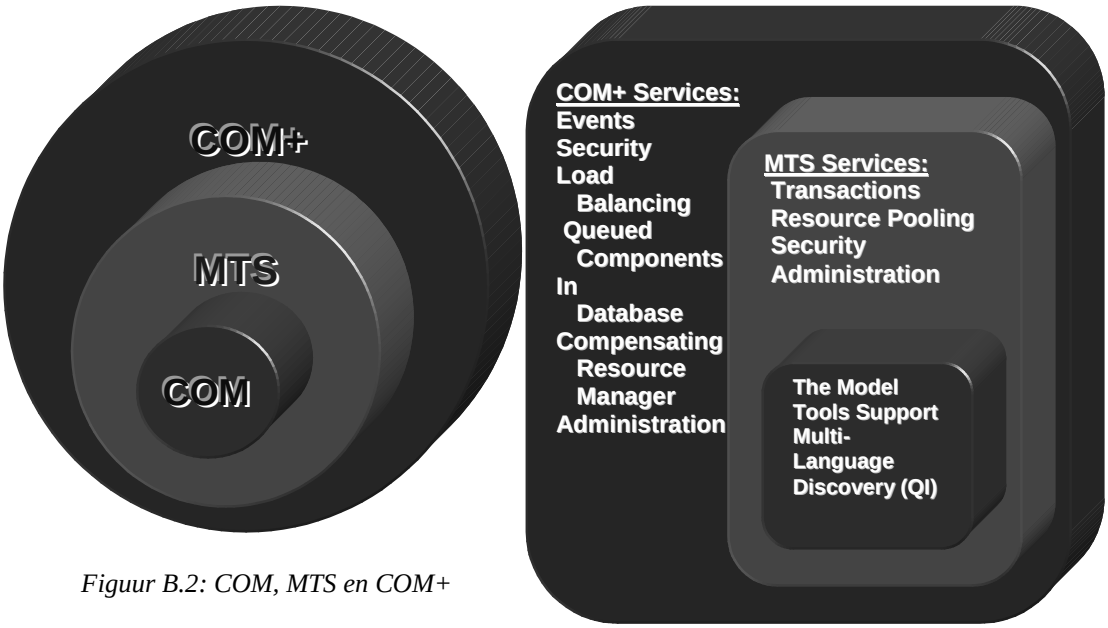
DCOM

DCOM (Distributed Component Object Model) is een extensie op het hierboven genoemde COM model van Microsoft in die zin, dat het zich wel richt op client/server interfaces. DCOM is een set concepten en programma interfaces waarmee client objecten diensten kunnen aanvragen bij server componenten op andere computers in het netwerk. De berichten die tussen objecten heen en weer gestuurd worden, worden RPC's genoemd (Remote Procedure Calls).

DCOM is niet gelimiteerd tot een bepaald netwerk: het gebruikt TCP/IP en HTTP als protocollen. DCOM en CORBA zijn over het algemeen gelijkwaardig aan elkaar wat betreft het verschaffen van gedistribueerde diensten. Microsoft onthoudt zich dus van de CORBA 'standaard'. CORBA en Microsoft hebben echter wel afspraken gemaakt over koppelingen tussen CORBA en DCOM (een zogenaamde 'gateway'), zodat een client object dat ontwikkelde is met (D)COM ook met een CORBA server kan communiceren (en omgekeerd).

COM+

COM+ is een extensie van Microsoft's Component Object Model. Het voegt aan COM een aantal diensten toe voor applicatie componenten, zoals het informeren van belangrijke gebeurtenissen. COM+ is bedoeld om een model te verschaffen dat het relatief makkelijk moet maken om bedrijfsapplicaties te ontwikkelen die goed samenwerken met Microsoft Transaction Server (MTS, zie verderop), een antwoord op het initiatief van Sun, IBM en Oracle dat Enterprise JavaBeans wordt genoemd (EJB, zie eveneens verderop).



Figuur B.2: COM, MTS en COM+

Remote Method Invocation

Remote Method Invocation, kortweg RMI, is een manier om in Java geschreven objecten op verschillende computers met elkaar kunnen communiceren en samenwerken. RMI is de Java versie van wat algemeen bekend staat als een Remote Procedure Call (RPC), maar met de mogelijkheid om één of meerdere objecten met het verzoek (of het antwoord) mee te sturen. Het object bevat informatie dat de uitvoering van de service aan kan passen. Zo kan op een willekeurige computer een object bijgehouden worden met eigenschappen en procedures die bepalen hoe een bepaalde taak uitgevoerd wordt. Als de taak uitgevoerd moet worden, wordt het object opgevraagd, en zodoende de taak op de meest up-to-date wijze uitgevoerd. RMI is bedoeld om eigenschappen en procedures van objecten, die gedistribueerd zijn over een netwerk, op dezelfde wijze te kunnen benaderen alsof ze op een lokale machine staan.

RMI is geïmplementeerd in drie lagen tussen de applicatie en de Java Virtual Machine (het interpreterende gedeelte dat boven het Operating System geplaatst is). Deze zijn:

1. Een stub programma aan de client zijde en een skeleton programma aan de server zijde. Voor het uitgevoerde programma lijkt de stub (Sun noemt dit *proxy*) het programma te zijn dat aangeroepen wordt voor een bepaalde dienst.
2. Een Remote Reference Layer. Deze laag bepaalt, op basis van wat de uitgevoerde applicatie vereist, hoe de procedures uitgevoerd dienen te worden.
3. Een Transport Connection Layer, die een verzoek afhandelt.

Een verzoek gaat in de ene computer omlaag door de lagen heen, en bij de andere computer van omhoog.

Transacties

Een transactionele bewerking heeft een 'alles-of-niets' karakter. Hiermee wordt bedoeld dat de gehele bewerking atomair dient te zijn: er mag niet slechts een deel van uitgevoerd worden. Een klassiek voorbeeld van een transactie is het overschrijven van geld van de ene rekening naar de andere. De deelhandelingen zijn: het afschrijven van het geld van de ene rekening en het bijschrijven op de andere rekening. Als slechts één van deze twee handelingen wordt uitgevoerd, zal zeker één van de betrokken partijen niet blij zijn. De transactie moet dus ofwel als geheel uitgevoerd worden, ofwel helemaal niet. De onderstaande technieken bieden mogelijkheden die speciaal gericht zijn op transactionele bewerkingen van applicaties.

Enterprise JavaBeans

JavaBeans is een platform-onafhankelijk component model dat geschreven is in Java. Het stelt ontwikkelaars in staat om één maal hergebruikbare componenten te schrijven en die overal te runnen (WORA – Write Once, Run Anywhere™). Net zoals Java applets, verschaffen JavaBeans componenten (of "Beans") de mogelijkheid om web pagina's (of andere applicaties) interactieve aspecten te geven zoals het berekenen van rente. De componenten kunnen in willekeurige combinaties samenwerken. Applicaties kunnen nu ook binnen willekeurige applicaties ("containers") draaien, zoals Word. Over het web kunnen Beans gebruikt worden door ze te koppelen met applets.

Enterprise JavaBeans (EJB) is een API (Application Program Interface) specificatie voor het bouwen van gedistribueerde multi-tier applicaties. EJB is een uitbreiding voor het JavaBeans componenten model gericht op transactionele bedrijfstoepassingen. EJB is een extensie voor JavaBeans richting zakelijke middle-tier/server-side applicaties. Enterprise

JavaBeans richt zich op applicaties die te maken hebben met transacties. EJB is geen product, maar een API voor de Java technologie.

Microsoft Transaction Server

Microsoft Transaction Server (MTS) is een alternatief voor Enterprise JavaBeans. In tegenstelling tot EJB is MTS wel een product. De Microsoft Transaction Server is een programma dat op een Internet server of een andere server draait en applicatie- en databasetransactie verzoeken afhandelt voor een client computer gebruiker. Ook worden beveiliging, verbinding met andere servers en transactie integriteit verzorgd. MTS verhoogt de scalability van al bestaande applicaties die op ActiveX (zie ook paragraaf 7.1.3.2) gebaseerd zijn.

MTS is bedoeld om de creatie van gedistribueerde applicaties over een netwerk makkelijker te maken. MTS (en ook EJB, of beter gezegd, applicaties gebaseerd op de EJB API) vallen in de categorie middleware voor de zakelijke markt.

Gegevens-representatie

Web based ontwikkeling, hoe uiteenlopend de uiteindelijke toepassingen ook mogen zijn, heeft altijd één ding gemeen: het draait om communicatie. Altijd worden gegevens tussen een bron en een doel heen en weer gestuurd. Natuurlijk kunnen aan de opmaak van die gegevens ook eisen gesteld worden, en zijn er verschillende alternatieven. Afhankelijk van het doel van de uiteindelijke applicatie, zullen verschillende alternatieven van gegevensrepresentatie meer of mindere interessant zijn.

HTML

HTML staat voor HyperText Markup Language. HTML bestaat uit een aantal codes (of “markup” symbolen) die aangeven hoe de tekst in het document getoond moet worden in een web browser. HTML is in staat om tekst op te maken, afbeeldingen en tabellen weer te geven en dingen als knoppen en radiobuttons te tonen. Officieel wordt HTML (en andere talen zoals XML) gedefinieerd door het World Wide Web Consortium (W3C). Een voorbeeld van HTML is het volgende:

De HTML code

```
Deze tekst kan <B>vet</B> en <I>schuin</I> afgedrukt worden.
```

Resulteert in de volgende opmaak:

Deze tekst kan **vet** en *schuin* afgedrukt worden.

DHTML

DHTML (Dynamic HTML) is eigenlijk gewoon HTML 4, de vierde revisie van de HyperText Markup Language, met een paar extra mogelijkheden. HTML 4 wordt 'aanbevolen' door het W3C als een officiële versie. De extra mogelijkheden van DHTML boven HTML 4 zijn per browser (Navigator, Internet Explorer) anders ingevuld. Wat DHTML en HTML 4 boven 'gewoon' HTML bieden is een rijker aanbod van mogelijkheden om de lay-out van een web pagina te manipuleren en maakt meer interactiviteit met bezoekers mogelijk. Een paar voorbeelden zijn meertalige versies van web pagina's en ondersteuning van frames.

Daarnaast is er ondersteuning voor Cascading Style Sheets (CSS). Een style sheet geeft aan hoe een document er uit moet zien. Het bepaalt lettertype eigenschappen, lay-out van paragrafen, et cetera. Een cascading style sheet is een style sheet die er van uit gaat dat andere style sheets gedeeltes van de 'hoofd style sheet' zullen herdefiniëren. Het voordeel hiervan is dat een ontwikkelaar altijd uit kan gaan van de hoofd style sheet, en, waar nodig, gedeeltes kan aanpassen. W3C geeft ook hiervoor een aanbeveling die gedefinieerd is als Cascading Style Sheets, level 1 (CSS1).

XHTML

XHTML staat voor eXtensible HyperText Markup Language. Volgens W3C is het "een herformulering van HTML 4 in de vorm van een toepassing van de eXtensible Markup Language" (XML, zie hieronder). XHTML is een derivaat van XML, speciaal voor het opmaken van web pagina's. XHTML is in feite de opvolger van HTML 4 (XHTML zou dus HTML 5 genoemd kunnen worden). Het verschil met HTML is dat iedereen XHTML kan uitbreiden (vandaar 'extensible') met nieuwe tags (sleutelwoordjes die de opmaak aangeven). Zie voor meer informatie het gedeelte over XML.

XML

Zoals hierboven genoemd staat XML voor eXtensible Markup Language. Het is een meta-markup language die een formattering verschaft voor het beschrijven van gestructureerde gegevens. Dit vergemakkelijkt het maken van meer precieze inhoudelijke declaraties, zorgt voor meer betekenisvolle zoekresultaten over meerdere platforms en een uniforme manier van gegevens beschrijving over het World Wide Web, intranetten en andere netwerken. Zodra afspraken gemaakt worden over de manier van beschrijven van een bepaald object kan deze structuur gebruikt worden voor geavanceerde mogelijkheden. Bijvoorbeeld: computer fabrikanten bereiken overeenstemming over een standaard wijze voor het

beschrijven van computer producten (processor snelheid, geheugen, enz.) en implementeren die met behulp van XML. Dan kunnen gebruikers een “intelligent agent” (een programmaatje) naar de website van elke computer fabrikant sturen om gegevens te verzamelen en een geldige vergelijking te maken.

XML en HTML komen overeen met betrekking tot het feit dat beide markup symbolen gebruiken om de inhoud van een document te omschrijven. Bij HTML betreft dat echter alleen de wijze hoe de inhoud van een web pagina wordt weergegeven. XML omschrijft de inhoud op basis van de eigenschappen van de gegevens die worden omschreven. Dit houdt in dat een XML bestand puur als gegevens door een programma verwerkt kan worden, samen met gelijksoortige gegevens op een andere computer opgeslagen kan worden, of, zoals een HTML bestand, weergegeven kan worden.

XML is uitbreidbaar (“extensible”) in die zin dat, in tegenstelling tot HTML, de opmaak symbolen ongelimiteerd zijn en zichzelf definiëren. XML is eigenlijk een makkelijkere sub-set van de Standard Generalized Markup Language (SGML), de standaard voor het opzetten van een document structuur. XML en HTML kunnen goed samen gebruikt worden in veel web applicaties. Zie ook het gedeelte over XHTML.

XML kan naast het opmaken van tekst ook gebruikt worden voor het systematisch beschrijven en opmaken van de volgende dingen:

- Een gestructureerd ‘record’, zoals een afspraak of een bestelling
- Een object met gegevens en procedures, zoals een ActiveX control
- Een gegevens ‘record’, zoals het resultaat van een query
- Grafische presentatie, zoals de user interface van een applicatie

Zodra de gegevens locaal bij de client zijn, kunnen ze gemanipuleerd, aangepast en op verschillende manieren gepresenteerd worden, zonder dat de informatie weer van de server gehaald hoeft te worden. Ook kunnen gegevens van verschillende bronnen makkelijk samengevoegd worden.

XML scheidt de gegevens van de presentatie. HTML specificeert hoe gegevens in een browser weergegeven dienen te worden, XML definieert de inhoud. De presentatie in een browser van gegevens in een XML bestand gebeurt door middel van Cascading Style Sheets (CSS, zie boven) en eXtensible Style Language (XSL). Door deze scheiding kunnen gegevens van bronnen naadloos geïntegreerd worden. Een XML document bestaat uit drie delen:

- **Schema:**
  document met definities van de gebruikte ‘element typen’ (wat is een klant, een bestelling, enz.). De eigen tags van de gebruiker worden hier gedefinieerd. Dit bestand heeft .xml als extensie.
- **Style Sheet:**
  verzorgt de interface, zorgt voor de ordening, kan scripting bevatten (JavaScript, VBScript, …) maar ook bijvoorbeeld C++. XSL file, of een CSS (Cascading Style Sheet). Voordeel van XSL boven CSS is dat, indien de lay-out aangepast moet worden, bij gebruik van CSS de XML file aangepast moet worden. Bij het gebruik van XSL dient alleen de XSL file aangepast te worden. Extensie: .xsl.
- **XML document:**
  bevat de gegevens in geordende manier, met behulp van de ‘datatypen’ zoals die in het schema zijn gedefinieerd. Een XML document heeft een .xml extensie.

Omdat, net zoals bij HTML, HTTP (HyperText Transfer Protocol) gebruikt wordt om XML van de server naar de client te transporteren, hoeft de bestaande infrastructuur dus niet aangepast te worden om XML te gebruiken.