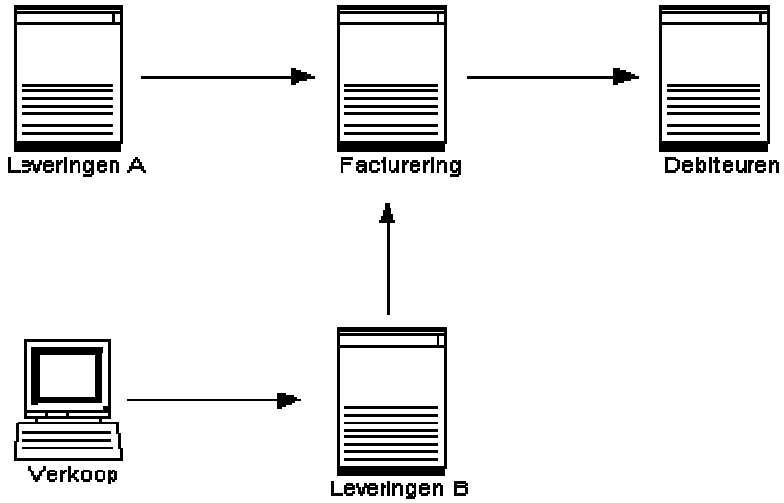
Multi-tier client/server revolutie

Op weg naar een flexibele, onderhoudbare informatievoorziening

Vandaag de dag hebben organisaties nog altijd te kampen met hoge ontwikkel- en onderhoudskosten bij het gebruik van informatietechnologie (IT). Het beschikbaar stellen van de juiste gegevens op het juiste tijdstip en de juiste plaats vergt een grote inspanning. De bestaande informatiesystemen belemmeren organisaties in het snel invoeren van nieuwe produkten en diensten. Ter illustratie beschouwen we de situaties bij de bedrijven X en Y.

Bij bedrijf X slaagt men er waarschijnlijk niet in om in korte tijd een nieuwe vorm van facturering met allerlei kortingsregelingen door te voeren. Omdat een belangrijke concurrent van X deze nieuwe vorm van facturering al aanbiedt, bestaat de kans dat klanten zullen overstappen naar deze concurrent.

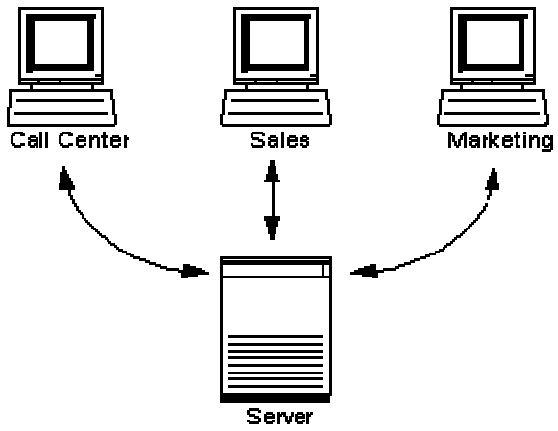


Figuur 1 Informatie-architectuur bij bedrijf X

In bovenstaand figuur is de informatie-architectuur van bedrijf X schematisch weergegeven. De twee typen produkten die X levert worden in twee aparte mainframetoepassingen (leveringen A en B) geadministreerd. Beide systemen leveren periodiek gegevens aan voor het factureringssysteem dat op zijn beurt weer gegevens aanlevert voor de debiteurenapplicatie. Daarnaast is een aparte applicatie ontwikkeld om de verkoop van produkttype B te vereenvoudigen. Deze applicatie levert periodiek gegevens aan systeem "leveringen B".

Het aanpassen van het bestaande factureringssysteem brengt een aantal problemen met zich mee. Het grootste probleem betreft de koppelingen. Voor de nieuwe vorm van facturering zijn meer gegevens van de leveringssystemen nodig. Het debiteurensysteem moet echter van dezelfde gegevens blijven voorzien. Een aanzienlijk deel van de inspanning om de nieuwe vorm van facturering in te voeren is nodig om de koppelingen te onderhouden. Daarnaast geldt dat de aan te passen systemen oude systemen zijn. Veel documentatie loopt achter bij het onderhoud dat gepleegd is. Ook is het aantal medewerkers dat diepgaande kennis van het systeem heeft beperkt.

Een tweede voorbeeld betreft bedrijf Y. Uit een door een extern adviesbureau uitgevoerd onderzoek is gebleken dat de ondersteuning van klanten zeer te wensen overlaat. Het inrichten van een Call Center leidde tot serieuze problemen in de informatiehuishouding van bedrijf Y.

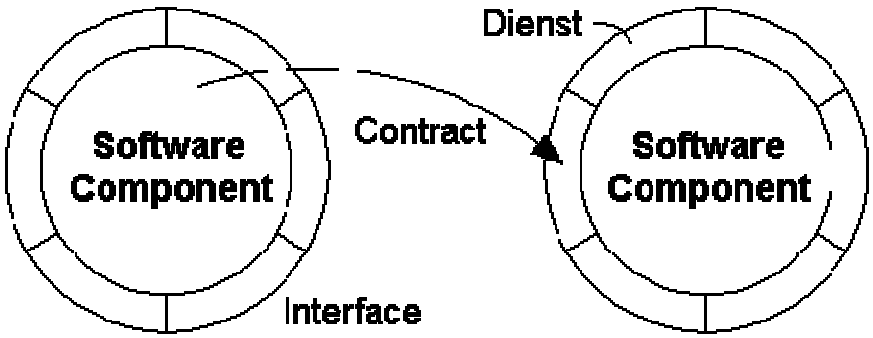


Figuur 2 Informatie-architectuur bij bedrijf Y

In bovenstaand figuur is de situatie bij bedrijf Y schematisch weergegeven. In het verleden is gekozen voor een client/server architectuur. Een database server die de bedrijfsgegevens bevat en de bijbehorende proceslogica voor zijn rekening neemt, staat centraal. Voor de verkoop- en marketingafdeling zijn aparte clients ontwikkeld. Nu voor het Call Center ook aparte clients zijn gebouwd en uitgerold, loopt men tegen de grenzen van de verwerkingscapaciteit van de database server aan. De responsietijden zijn te hoog. Klanten hangen op. In winkels raakt men geïrriteerd.

Problemen als hierboven zijn gemeengoed. Wijzigingen in systemen vergen grote inspanningen en kosten veel geld. Organisaties worden in hun slagvaardigheid en flexibiliteit bedreigd. In dit artikel gaan we in op de rol die *multi-tier client/server architecturen* kunnen spelen bij het oplossen van de problematiek. Achtereenvolgens gaan we in op de materie rond architecturen, de problematiek van client/server oplossingen, het potentieel van multi-tier architecturen, de implementatie- en migratie-aspecten en ten slotte de ondersteuning van IT-leveranciers.

Architecturen
Het fundament van een goede informatie-architectuur is *ontkoppeling*. Ontkoppeling houdt in dat de informatievoorziening wordt gerealiseerd middels een stelsel van onafhankelijke softwarecomponenten. Iedere component biedt middels een *interface* een aantal *diensten* aan. Communicatie tussen componenten verloopt altijd via de interfaces. Wanneer een component gebruik maakt van de diensten van een andere component, wordt er tussen de componenten een *contract* afgesloten. Dit contract bevat afspraken over de beschikbaarheid van de dienst, de gegarandeerde responsietijd en andere operationele afspraken.

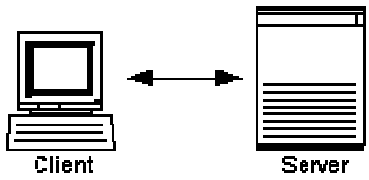


Figuur 3 Ontkoppeling

Sofwarecomponenten gaan verder dan de bekende *softwaremodules*. Bij softwaremodules blijft de ontkoppeling beperkt tot het programma of systeem waartoe de modules behoren. Bij softwarecomponenten gaat het om ontkoppeling op een hoger niveau. Systemen en programma's bestaan niet meer. Iedere softwarecomponent neemt een specifiek deel van de totale functionaliteit voor zijn rekening. Applicaties ontstaan doordat componenten op een bepaalde manier met elkaar samenwerken.

Een samenstelsel van onafhankelijke componenten is de sleutel tot een flexibele, onderhoudbare informatievoorziening. Een belangrijke vraag is op grond waarvan componenten onderscheiden moeten worden. Er is een visie nodig om tot een goede architectuur (een opdeling in componenten) te komen.

Client/server architectuur
Een veel gebruikte visie op ontkoppeling is de client/server architectuur waar twee soorten componenten worden onderscheiden: *clients* en *servers*. Een client vraagt een server om een bepaalde taak uit te voeren en het resultaat hiervan terug te geven.



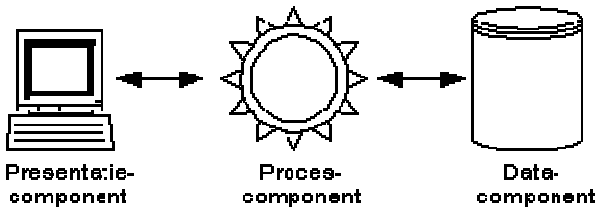
Figuur 4 Client/server architectuur

Grofweg zijn er twee varianten in het client/server model: de *fat client* variant waarbij proceslogica in de clients is ondergebracht en de *fat server* variant waarbij de proceslogica op de server draait. Een groot probleem van het fat client model is de duplicatie van proceslogica bij iedere client en de hiermee gepaard gaande distributie en consistentie problemen. Een ander probleem is de belasting van het netwerk door de grote hoeveelheid gegevens die van de server naar de clients getransporteerd moet worden.

Het fat server model kent het probleem dat de performance van de applicatie negatief wordt beïnvloed bij een toename van het aantal clients. De server krijgt dan zoveel taken uit te voeren dat deze het resultaat van de taak pas na lange tijd aan de client terug kan geven. Daarnaast geldt dat migratie naar databases van andere leveranciers lastig is omdat elke leverancier zijn eigen taal heeft waarin de proceslogica aan de server kant wordt vastgelegd.

Door deze problemen van de client/server architectuur bestaat er in toenemende mate belangstelling voor multi-tier architecturen.

Multi-tier architectuur
In een multi-tier architectuur worden ten minste drie typen componenten onderkend: presentatie-, proces- en datacomponenten.



Figuur 5 Multi-tier architectuur

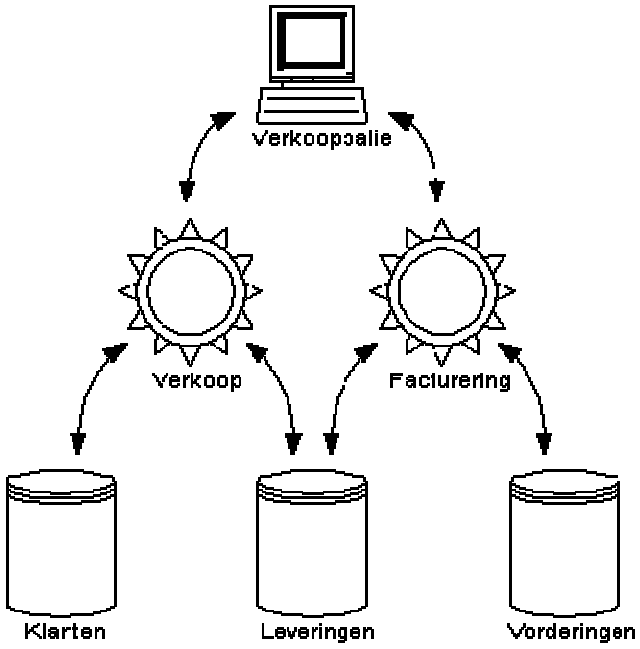
Deze indeling is gebaseerd op mate van stabiliteit. In volgorde van toenemende stabiliteit houden de componenten het volgende in:

Presentatiecomponenten - ondersteunen gebruikers zodat zij hun taak optimaal kunnen uitvoeren. Presentatiecomponenten bevatten alleen functies die presentatie en interactie met de gebruikers omvatten. Voor ieder type gebruiker (bijvoorbeeld verkopers, back-office medewerkers en systeembeheerders) is er een presentatiecomponent. Presentatiecomponenten communiceren met procescomponenten en eventueel ook met datacomponenten.

Procescomponenten - implementeren bedrijfsprocessen. Groepen van bij elkaar horende elementaire bedrijfsfuncties zijn opgenomen in een procescomponent. Voorbeelden van diensten van een procescomponent dat het bedrijfsproces van leveringen ondersteunt zijn "Toevoegen nieuwe klant", "Opnemen levering" en "Maken afspraak". Procescomponenten maken gebruik van de diensten van andere procescomponenten en datacomponenten.

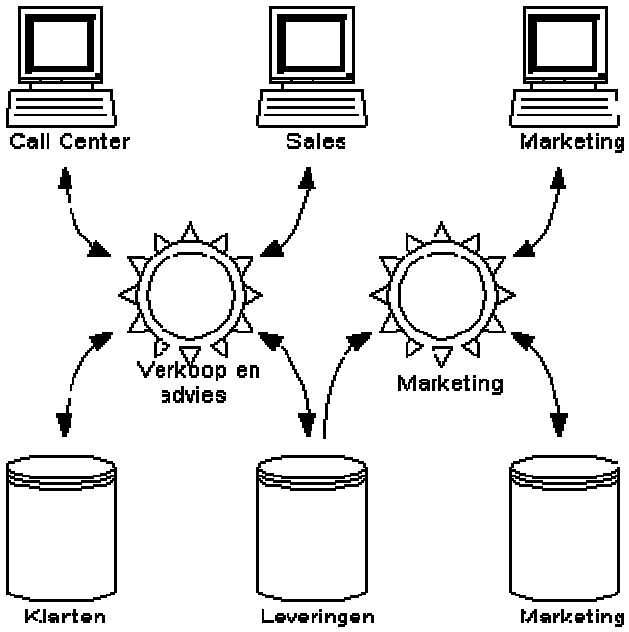
Datacomponenten - beheren bedrijfsgegevens. Dit zijn gegevens die door verschillende proces- en presentatiecomponenten gebruikt worden. De diensten van datacomponenten bestaan uit elementaire functies op gegevens en eventueel ook complexe queries. De datacomponent klant biedt bijvoorbeeld de diensten aan voor het creëren van een nieuwe klant, het verwijderen van een klant, het ophalen van de gegevens van een klant, het actualiseren van de gegevens van een klant, en het ophalen van alle klanten uit een bepaald marktsegment.

Stel dat de informatiebehoefte van bedrijf X volgens een multi-tier architectuur gerealiseerd was als in onderstaande figuur. Men zou dan kunnen volstaan met het maken van een nieuwe dienst in de factureringscomponent. Er is geen overhead met het onderhouden van koppelingen. De nieuwe vorm van facturering kan sneller en goedkoper worden gerealiseerd.



Figuur 6 Multi-tier oplossing voor bedrijf X

De ontwikkeling van een aparte Call Center presentatiecomponent bij bedrijf Y zal geen problemen opleveren wanneer het werk uitgevoerd wordt door een aantal proces- en datacomponenten in plaats van door één fat server. De componenten dragen bij aan een betere verdeling van de werklast.



Figuur 7 Multi-tier oplossing voor bedrijf Y

Migratie
Een belangrijk vraagstuk betreft de overgang van de bestaande informatie-architectuur naar een multi-tier client/server oplossing. Het simpelweg in één keer implementeren en in produktie nemen van alle nieuwe componenten brengt grote risico's met zich mee en kost veel geld. Beter is het om in kleine stapjes naar de gewenste architectuur toe te migreren. Globaal gezien kunnen daarvoor twee strategieën worden gevolgd: *database first* en *database last*. Voor beide strategieën is het belangrijk om vooraf de gewenste eindarchitectuur te ontwerpen. Vervolgens wordt gekeken in hoeverre de

bestaande systemen in deze blauwdruk passen. Per systeem wordt bepaald of deze de rol van een softwarecomponent gaat vervullen of dat deze wordt afgebouwd.

De database first strategie houdt in dat begonnen wordt met de realisatie van de ontworpen datacomponenten. Met behulp van *reverse gateways* worden de bestaande systemen gekoppeld aan de nieuwe datacomponenten. Reverse gateways zijn complex, het aantal tools hiervoor beperkt. Voor de database last aanpak is meer ondersteuning. Bij database last wordt begonnen met de bouw van presentatiecomponenten die met behulp van *forward gateways* aan de bestaande systemen worden gekoppeld. Een voorbeeld van een forward gateway is een *screen scraper*. Na de presentatiecomponenten worden de procescomponenten gerealiseerd en ten slotte de databases.

Bestaande systemen die de rol van een softwarecomponent gaan vervullen, worden met behulp van *adaptors* in de architectuur gehangen. Een adaptor is een stuk software dat het interface van de softwarecomponent implementeert en aanroepen van diensten vertaalt naar aanroepen van routines in het bestaande systeem.

Infrastructuur
Om softwarecomponenten daadwerkelijk met elkaar te laten communiceren is er een transport medium nodig, een infrastructuur. Dit type software wordt ook wel aangeduid met de term *middleware*. De belangrijkste infrastructuurstandaards zijn CORBA van de Object Management Group (OMG), DCOM van Microsoft en DCE van de Open Software Foundation (OSF). Een goede infrastructuur biedt voorzieningen voor onder andere gedistribueerde transacties, synchrone en asynchrone communicatie tussen componenten, unieke naamgeving van componenten binnen een domein en mechanismen voor autorisatie en authenticatie.

Middleware kan op verschillende paradigma's zijn gebaseerd. De meeste varianten zijn gebaseerd op het aanroepen van functies op servers. Communicatie vindt plaats op het niveau van componenten. Dit betekent dat componenten van elkaar moeten weten dat ze bestaan. Bij geavanceerdere middleware kunnen de diensten van instanties van objecten op servers worden aangeroepen. De communicatie vindt hier op een hoger abstractieniveau plaats, dat van objecten.

Ter illustratie van het laatste paradigma kijken we kort naar CORBA. CORBA bestaat uit een *Object Request Broker* (ORB), *Object Services*, *Application Objects* en *Common Facilities*. Via de ORB kunnen CORBA objecten tijdens de uitvoering vragen welke diensten beschikbaar zijn en daar vervolgens gebruik van maken. Een ORB beschikt over voorzieningen voor beveiliging van het gebruik van diensten en dataverkeer. De Object Services zijn verzamelingen elementaire diensten voor naamgeving, persistentie, concurrency etcetera. De Application Objects zijn de objecten die de organisatie modelleren en die de gebruikers bij hun werkzaamheden ondersteunen. Deze objecten worden ook wel aangeduid met de term *business objects*.

Om de ontwikkeling van application objects te vereenvoudigen kent CORBA Common Facilities. Dit zijn voorgedefinieerde raamwerken van samenwerkende objecten die kant-en-klaar ingezet kunnen worden in specifieke situaties. Hierbij kan onderscheid gemaakt worden naar twee categorieën: horizontale en verticale. De horizontale bieden algemene, domeinonafhankelijke ondersteuning (gebruikersinterface-objecten, informatiemanagementobjecten en objecten ter ondersteuning van werkstroombesturing, e-mail en dergelijke), de verticale omvatten objecten voor de ondersteuning van specifieke bedrijfstakken (banken, telecommunicatie, gezondheidszorg, procesindustrie, etcetera).

Ontwikkeltools
Op de markt is er een aantal tools die invulling geven aan zowel de multi-tier architectuur als de infrastructuur. Deze tools bieden ondersteuning voor het bouwen van software componenten en hun *deployment* (installatie in een run-time omgeving). De belangrijke spelers op de markt zijn:

- Composer van Texas Instruments
- Dynasty van Dynasty
- Forté van Forté
- NatStar van NatSystems
- Prolifics van Prolifics
- Sedona van Oracle
- SuperNova van Four Seasons
- Usoft van Usoft
- VisualGen van IBM

Vragen die bij de keuze van een ontwikkeltool gesteld moeten worden, zijn:

- Welke visie op ontkoppeling hanteert het tool?
- Biedt het tool ondersteuning voor de ontwerpfase?
- Hoe wordt de overgang van ontwerp naar implementatie ondersteunt?
- Hoe wordt het ontwikkelproces ondersteunt?
- Wordt versiebeheer ondersteund?
- In hoeverre wordt het maken van operationele afspraken (contracten) tussen componenten ondersteund?
- Is het mogelijk om componenten vanaf één punt te distribueren?

- Is het mogelijk om de componenten centraal te beheren?
- Welke middleware wordt er door het ontwikkeltool gebruikt om de componenten met elkaar te laten communiceren?
- Voorziet het tool in zogenaamde *load balancing* waarmee componenten afhankelijk van de belasting verdeeld kunnen worden over de beschikbare servers?
- Welke mogelijkheden zijn er om bestaande systemen te koppelen?
- Welke ondersteuning is er voor de verschillende databasesystemen?
- Welke ondersteuning is er voor de verschillende TP monitors?
- In hoeverre is het tool platformonafhankelijk?

Evaluatieproject
Om een antwoord te vinden op bovenstaande vragen organiseren Computable en SERC van 7 tot en met 11 april 1997 het multi-tier client/server evaluatieproject. Het doel van dit project is naast het onderzoeken van de mogelijkheden van multi-tier client/server ontwikkeltools het opdoen van ervaring met het ontwikkelen van multi-tier architecturen.

Het vijf dagen durend evaluatieproject begint met een introductie van multi-tier architecturen. Tijdens deze introductie zullen zowel de theoretische als praktische aspecten aan bod komen. Vervolgens zal men in een team aan de slag gaan met het uitwerken van een casus tot een multi-tier ontwerp, om deze vervolgens met één van de ontwikkeltools te implementeren. De teams zullen ondersteund worden door experts van de verschillende ontwikkeltools. Ter afsluiting zal elk team de resultaten presenteren en vindt er een discussie plaats.

Ontwikkelaars die aangelopen zijn tegen de grenzen van client/server en op zoek zijn naar alternatieven en andere belangstellenden, kunnen zich schriftelijk, telefonisch, per fax of per e-mail opgeven bij:

Software Engineering Research Centre
Postbus 424
3500 AK Utrecht
Telefoon: 030 254 5412
Telefax: 030 254 5948
E-mail: info@serc.nl

De kosten voor deelname aan het evaluatieproject bedragen *f* 3.975,- per persoon. De tweede en derde deelnemer van één bedrijf krijgen respectievelijk *f* 500,- en *f* 1.000,- korting.

Auteurs

- *Drs. P.J. Koning* is Consultant bij het Software Engineering Research Centre.
- *Ir. Guido van der Harst* is Project Manager bij het Software Engineering Research Centre.
- *Ing. G.F. Florijn* is universitair docent bij de Vakgroep Informatica van de Universiteit Utrecht en Project Manager bij het Software Engineering Research Centre