

Netwerkarchitecturen - architectuurconcepten en hun toepassing

Marten van Sinderen

Luís Ferreira Pires

1. Inleiding
2. Het service concept
3. Het protocol concept
4. Service definitie
5. Protocol definitie
6. Ontwerpen met het service en protocol concept
7. Literatuur

1 Inleiding

Een *netwerkarchitectuur* beschrijft de samenhang tussen de verschillende delen van een computernetwerk en bovendien de functie van het computernetwerk als geheel voor zijn gebruikersomgeving. Een netwerkarchitectuur speelt een belangrijke rol bij het gebruiken, implementeren en onderhouden van een computernetwerk.

Netwerkarchitecturen kunnen fabrikantspecifiek zijn, zoals bijvoorbeeld SNA en DECnet, of fabrikantonafhankelijk, zoals het OSI referentiemodel en de Internet protocol structuur. Al deze architecturen worden beschreven met behulp van een beperkt aantal *architectuurconcepten*. Deze concepten werden reeds intuïtief toegepast in de jaren '70 toen de eerste belangrijke netwerkexperimenten, zoals de ontwikkeling van het ARPAnet, plaatsvonden. De definitie van het OSI referentiemodel heeft echter pas voor een eenduidige naamgeving en precieze betekenis van deze concepten gezorgd. Naast een cruciale rol voor het beschrijven van netwerkarchitecturen, spelen architectuurconcepten ook een fundamentele rol in *ontwerpmethoden*. Een ontwerpmethode wordt gevolgd wanneer architectuurconcepten systematisch, bijvoorbeeld in een vaste volgorde, worden toegepast, waardoor de aard van ontwerpproblemen voorspelbaar en hun omvang beheersbaar wordt.

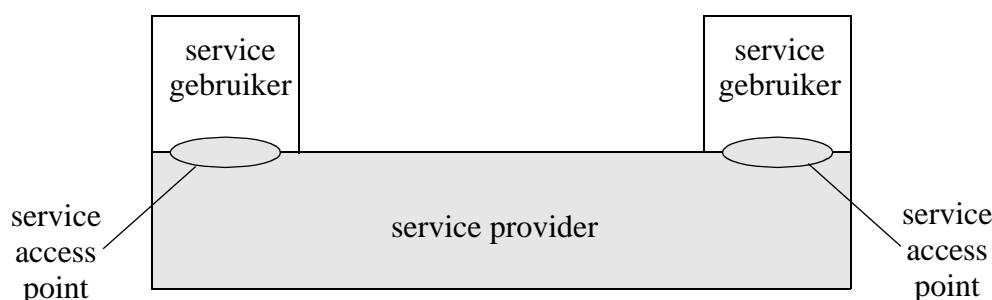
Belangrijke hoogniveau architectuurconcepten zijn het service concept en het protocol concept. Zowel een service als een protocol worden gedefinieerd in termen van een aantal basis architectuurconcepten. Voorbeelden van basis architectuurconcepten zijn service primitivum, service access point, protocol data unit, etc. In het volgende wordt het belang van hoogniveau architectuurconcepten en hun definitie in termen van basis architectuurconcepten beschreven. Tevens wordt ingegaan op de toepassing van hoogniveau architectuurconcepten in top-down ontwerpmethoden voor protocollen.

2 Het service concept

Het service concept wordt gebruikt om de diensten, of de *service*, van een gedistribueerd systeem (de service provider, bijvoorbeeld een netwerk) te beschrijven, zonder de interne structuur en werking van het gedistribueerde systeem zichtbaar te maken. Het beschouwde

gedistribueerde systeem, dat bijvoorbeeld een netwerk kan zijn, wordt een *service provider* genoemd.

Een service provider ‘levert’ een service aan zijn gebruikersomgeving, die bestaat uit *service gebruikers*. Service gebruikers kunnen zowel menselijke gebruikers als artificiële gebruikers (bijvoorbeeld computerprogramma’s) voorstellen. De service wordt beschikbaar gesteld aan een service gebruiker op een *Service Access Point* (SAP). Een SAP is een logische lokatie met een unieke naam waardoor het mogelijk wordt de service gebruiker die met de SAP verbonden is te adresseren. Figuur 1 toont een grafische weergave van het service concept¹.



Figuur 1: Het service concept

Het belang van het service concept schuilt in het feit dat het zowel voor de gebruiker als voor de ontwerper van een gedistribueerd systeem een adequate abstractie biedt. Gebruikers van een gedistribueerd systeem zijn uiteindelijk alleen geïnteresseerd in de geïntegreerde functionaliteit van het systeem, d.w.z. de functionaliteit die direct aangeeft hoe het systeem te gebruiken is en wat van het systeem verwacht kan worden. Deze functionaliteit wordt beschreven door een service.

Voor de ontwerper die bovenop een service provider een gedistribueerde functionaliteit wil plaatsen is de service het juiste uitgangspunt om deze gedistribueerde functionaliteit te ontwikkelen. Hetzelfde geldt voor de ontwerper die verantwoordelijk is voor de interne structuur en werking van de service provider. In beide gevallen wordt de ontwerper maximale vrijheid geboden (d.w.z. minimale beperkingen opgelegd): de functionaliteit bovenop de service provider moet ‘steunen’ op de service, en de interne werking van de service provider moet zodanig zijn dat de service ‘geleverd’ wordt.

Daarnaast vervult het service concept een strategische rol bij de evolutie van gedistribueerde systemen: de gedistribueerde functionaliteit die steunt op een service kan gewijzigd of vervangen worden zonder dat dit consequenties heeft voor de gedistribueerde functionaliteit die de service levert. Omgekeerd geldt natuurlijk hetzelfde.

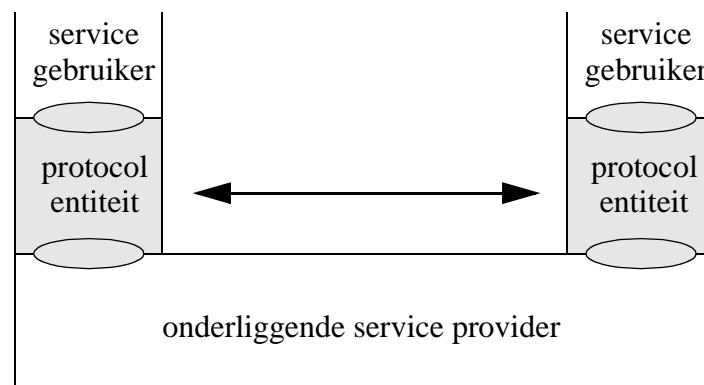
3 Het protocol concept

Het protocol concept wordt gebruikt om de procedure voor samenwerking, of het *protocol*, tussen delen van een gedistribueerd systeem te beschrijven, zonder de implementatiestructuur

1. Voor de eenvoud worden slechts twee service gebruikers getoond.

van deze delen zichtbaar te maken. De beschouwde delen worden *protocol entiteiten* genoemd. Protocol entiteiten zijn niet rechtstreeks met elkaar verbonden, maar moeten gebruik maken van een intermediair deelsysteem om informatie ten behoeve van de samenwerking uit te kunnen wisselen. Dit systeem, dat noodzakelijkwijs gedistribueerd is, wordt de *onderliggende service provider* genoemd.

Het protocol concept en het service concept zijn gekoppeld: een protocol zorgt ervoor dat een service wordt geleverd, d.w.z. het resultaat van de samenwerking tussen de protocol entiteiten moet gelijk zijn aan de door de gebruikers gewenste geïntegreerde functionaliteit. Dit houdt in dat er voor elke SAP een protocol entiteit moet zijn die hiermee direct verbonden is. Anders dan het service concept is het protocol gericht op de interne structuur en werking van een gedistribueerd systeem: de interne structuur wordt gedefinieerd door de verzameling protocol entiteiten en de werking wordt gedefinieerd door het protocol. De interne structuur volgt uit de beschouwing van de SAPs in de gewenste service en van de onderliggende service provider, die een service levert die algemener en eenvoudiger is dan de gewenste service. Figuur 2 toont een grafische weergave van het protocol concept².



Figuur 2: Het protocol concept

Het protocol concept wordt toegepast in protocol standaarden om de samenwerking tussen protocol entiteiten precies te definiëren, terwijl maximale vrijheid wordt geboden om de protocol entiteiten te implementeren. Een protocol is dus bedoeld voor fabrikanten van protocol standaarden. Door zich te houden aan het protocolvoorschrift kan de door een fabrikant geïmplementeerde protocol entiteit samenwerken met andere protocol entiteiten die zich aan dit voorschrift houden, ook wanneer deze protocol entiteiten door een andere fabrikant zijn geïmplementeerd.

Ook bij de ontwikkeling van fabrikantspecifieke protocollen is de toepassing van het protocol concept nuttig. De definitie van een protocol als tussenstap in het ontwerpproces dat moet leiden tot een protocolimplementatie maakt het proces overzichtelijker en daardoor gemakkelijker te beheersen.

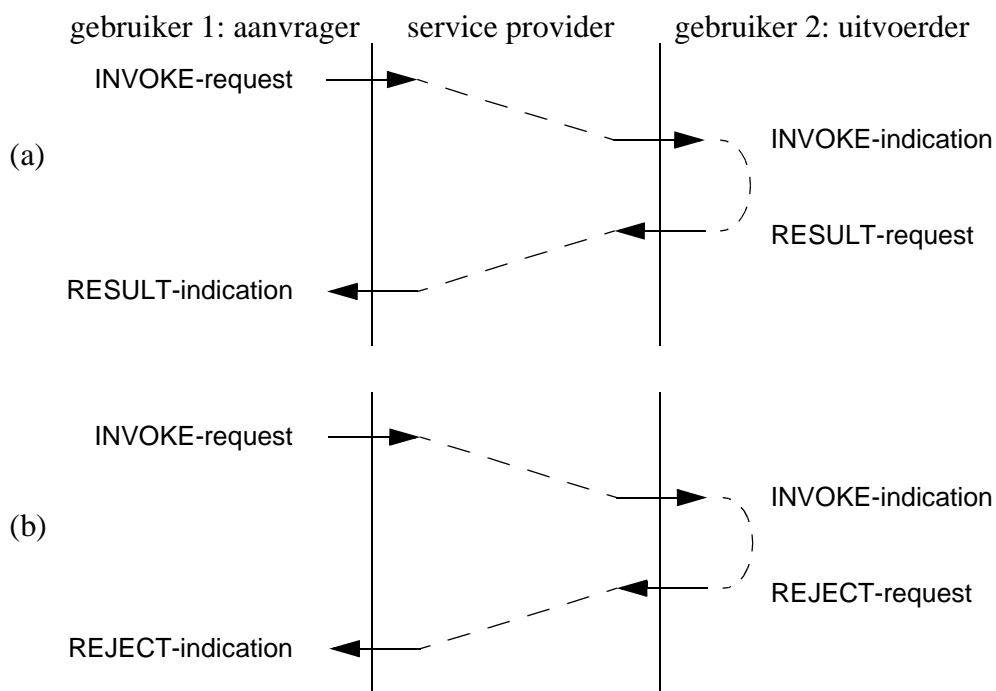
2. Voor de eenvoud worden slechts twee protocol entiteiten getoond. De samenwerking tussen protocol entiteiten hoeft niet per se paarsgewijs te zijn; het is mogelijk dat meer dan twee protocolentiteiten bij dezelfde samenwerking betrokken zijn.

4 Service definitie

Een service wordt gedefinieerd in termen van *service primitiva*. Een service primitivum is een interactie tussen een service gebruiker en de service provider.

Service primitiva zijn elementaire interacties in de zin dat het onderscheiden van deze interacties noodzakelijk en voldoende is om de service als uitgangspunt te kunnen laten dienen voor protocolontwikkeling; het onderscheiden van ‘fijnere’ interacties is hiervoor niet nodig. Elk service primitivum is bovendien een abstracte interactie omdat de verantwoordelijkheden in c.q. de bijdragen aan de interactie van respectievelijk de betrokken service gebruiker en de service provider niet worden gedefinieerd. De reden hiervoor is dat de fabrikant die een protocol implementeert zelf moet kunnen bepalen hoe het interface tussen de protocol entiteit en de service gebruiker eruit komt te zien. Dit *abstracte interface* kan bijvoorbeeld geïmplementeerd worden als een concreet interface, zoals een Application Programming Interface, maar kan ook onderdeel worden van een geïntegreerde implementatie van de service gebruiker en de protocol entiteit.

Bovengenoemde eigenschappen van service primitiva zorgen ervoor dat een service definitie op de meest eenvoudige wijze wordt gegeven. Figuur 3 illustreert de definitie van een service illustreren aan de hand van een voorbeeld.



Figuur 3: Twee mogelijke scenario's volgens de Remote Operations Service

In figuur 3a vindt een INVOKE-request primitivum plaats waarmee een service gebruiker (de aanvrager) om de uitvoering van een operatie vraagt. Dit primitivum wordt gevolgd door een INVOKE-indication primitivum dat als resultaat heeft dat de aanvraag wordt doorgegeven aan een andere service gebruiker (de uitvoerder). De uitvoerder maakt het resultaat van de succesvol verlopen operatie bekend in een RESULT-request primitivum. Dit primitivum wordt gevolgd door een RESULT-indication, waarmee het resultaat wordt doorgegeven aan de aanvrager.

Het voorbeeld laat zien dat in een service behalve de verzameling mogelijke service primitiva, ook de relatie tussen service primitiva moet worden gedefinieerd.

Elk service primitivum heeft een naam (bijvoorbeeld: INVOKE) en een type (bijvoorbeeld: request). Daarnaast bestaat een service primitivum meestal uit een of meer parameters (niet gerepresenteerd in het voorbeeld). Een belangrijke parameter is de *Data parameter*, die dient om op transparante wijze gegevens uit te wisselen tussen service gebruikers. De Data parameter wordt ook wel *Service Data Unit* (SDU) genoemd wanneer alleen gegevensuitwisseling wordt beschouwd en de naam van het service primitivum er niet toe doet.

Service primitiva kunnen onderling gerelateerd zijn wat betreft tijdsvolgorde en inhoud. Bijvoorbeeld, een INVOKE-indication kan alleen plaatsvinden na een INVOKE-request, en de operatie gespecificeerd in de INVOKE-indication moet dezelfde zijn als die in de INVOKE-request. Gerelateerde service primitiva kunnen plaatsvinden op dezelfde SAP of op verschillende SAPs. In het laatste geval is de service provider altijd volledig verantwoordelijk voor het afdwingen van de relatie: bij het protocol ontwerp zullen deze relaties geïmplementeerd moeten worden.

Service primitiva die altijd volgens een vast volgorde patroon zijn gerelateerd vormen een *service element*; de service primitiva van hetzelfde service element hebben dezelfde naam, maar verschillende types. De naam van het service element is dezelfde als de naam van de service primitiva. Het INVOKE service element heeft bijvoorbeeld betrekking op een INVOKE-request en een corresponderende INVOKE-indication. Een service definitie omvat in het algemeen meerdere service elementen die op verschillende manieren (vervlochten of in verschillende volgordes) gecombineerd kunnen worden.

Er zijn verschillende notationele hulpmiddelen voor service definities. In figuur 3 is een *tijdsvolgorde diagram* gebruikt om een deel van de Remote Operations Service te representeren. Een nadeel van deze notatie is dat per diagram slechts één mogelijke volgorde van service primitiva kan worden getoond. Een alternatieve volgorde vereist een ander diagram. Zo kan in plaats van een RESULT service element ook een REJECT service element plaatsvinden na een INVOKE service element, namelijk wanneer de operatie niet uitgevoerd kon worden door de uitvoerder. Dit alternatief is getoond in figuur 3b. Een andere notatie is de *toestandstabel*, waarmee, voor elk SAP, een aantal toestanden worden gedefinieerd, met in elke toestand een aantal mogelijke ‘gebeurtenissen’ die leiden tot een nieuwe toestand. Een gebeurtenis komt hier overeen met het plaatsvinden van een service primitivum. Tabel 1 toont de representatie van de Remote Operations Service met een toestandstabel. Een nadeel van de toestandstabel notatie is dat relaties tussen service primitiva die plaatsvinden op verschillende SAPs niet worden gerepresenteerd.

toestand gebeurtenis	idle_ initiator (0A)	await_ indication (1A)	idle_ responder (0B)	await_ request (1B)	...
INVOKE-request	1A				
INVOKE-indication			1B		
RESULT-request				0B	
RESULT-indication		0A			

Tabel 1: Toestandstabel voor de Remote Operations Service

toestand gebeurtenis	idle_ initiator (0A)	await_ indication (1A)	idle_ responder (0B)	await_ request (1B)	...
REJECT-request				0B	
REJECT-indication		0A			
...					

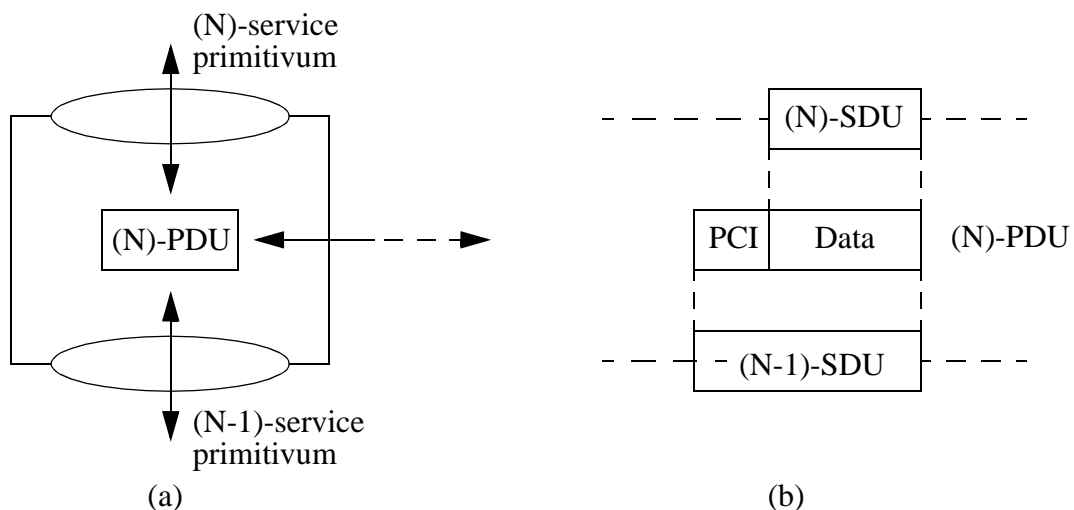
Tabel 1: Toestandstabel voor de Remote Operations Service

5 Protocol definitie

Een protocol definieert de samenwerking tussen de protocol entiteiten in termen van de functies (gedragingen) van de protocol entiteiten. Protocol entiteiten kunnen niet rechtstreeks met elkaar communiceren, maar moeten altijd gebruik maken van de onderliggende service. Dit betekent dat protocol entiteiten informatie-eenheden moeten uitwisselen, waarmee hun samenwerking tot stand gebracht kan worden. Deze informatie-eenheden heten *Protocol Data Units (PDUs)*.

Het uitwisselen van een PDU vindt plaats naar aanleiding van een bepaalde gebeurtenis: het optreden van een service primitivum, het ontvangen van een PDU, of het optreden van een interne actie van de protocol entiteit (zoals het verstrijken van een wachttijd). Het ontvangen van een PDU kan op zijn beurt weer nieuwe gebeurtenissen op gang brengen: het optreden van een service primitivum, het versturen van een PDU, of het optreden van een interne actie van de protocol entiteit (zoals het starten van een klok die een wachttijd meet).

Service primitiva, zowel van de gewenste als van de onderliggende service, zijn altijd gerelateerd met een PDU. Als conventie gebruiken wij de termen (N)-service primitivum, (N)-PDU, (N-1)-service primitivum, etc. om de elementen van een bepaald niveau in de protocol structuur aan te duiden. Figuur 4a illustreert de relatie tussen (N)-service primitiva, (N)-PDUs en (N-1)-service primitiva.



Figuur 4: Relatie tussen service primitiva en PDUs in een protocol

Figuur 4b laat zien dat een (N)-SDU, d.w.z. de Data parameter van een (N)-service primitivum, het *Data veld* vormt van een (N)-PDU. Een (N)-PDU bevat behalve een Data veld tevens een of meer velden met informatie die het gedrag ‘sturen’ van de protocol entiteit die de PDU ontvangt. Deze informatie wordt de ‘header’ of *Protocol Control Information (PCI)* genoemd. Een gecodeerde (N)-PDU wordt transparant (‘verpakt als data’) vervoerd van een zendende naar een ontvangende protocol entiteit, doordat er een afbeelding bestaat tussen de (N)-PDU en een (N-1)-SDU.

Door alle mogelijk (N)- en (N-1)-service primitiva, (N)-PDUs en hun relaties te definiëren wordt het gedrag van een protocol entiteit gedefinieerd.

PDUs vervullen een centrale rol in een protocol. Gebaseerd op deze rol zijn protocol functies te verdelen in twee typen:

- *hogere protocol functies* zijn gericht op het leveren van de gewenste service. Deze functies definiëren de relaties tussen (N)-service primitiva en (N)-PDUs, en hoe de verschillende gebeurtenissen de (N)-PDUs beïnvloeden en vice versa; en
- *lagere protocol functies* zijn gericht op het gebruik van de onderliggende service. Deze functies definiëren de relaties tussen (N)-PDUs en (N-1)-service primitiva, zoals de codering van PDUs en de manier waarop PDUs worden verstuurd.

Voorbeelden van hogere protocol functies zijn *segmentatie* en *reassemblage*. Deze twee functies worden gebruikt wanneer de maximale lengte van het Data veld van een (N)-PDU onvoldoende is om een (N)-SDU te vervoeren (de lengte van het Data veld wordt bepaald door de maximale (N-1)-SDU lengte). De segmentatie functie zorgt ervoor dat een (N)-SDU door een reeks van (N)-PDUs kan worden vervoerd door de data in stukken op te delen. De reassemblage functie zorgt ervoor dat uit de ontvangen (N)-PDUs de oorspronkelijke data wordt teruggewonnen door de stukken in de goede volgorde samen te stellen tot een (N)-SDU.

Voorbeelden van lagere protocol functies zijn *multiplexing* en *demultiplexing*. Deze functies zijn nodig om met een enkele (N-1)-verbinding meerdere (N)-verbindingen te kunnen leveren. Het delen van een (N-1)-verbinding leidt in principe tot lagere kosten per (N)-verbinding. De multiplexing functie zorgt ervoor dat de gecodeerde (N)-PDUs die horen bij de gemultiplexte (N)-verbindingen worden getransporteerd als (N-1)-SDUs over een (N-1)-verbinding. De demultiplexing functie doet het omgekeerde. De gecodeerde (N)-PDUs die als (N-1)-SDUs over een (N-1)-verbinding zijn ontvangen worden weer geassocieerd met de juiste (N)-verbindingen.

Een voorschrift voor protocol implementatie wordt in het algemeen *protocol specificatie*, in plaats van protocol definitie, genoemd. Een protocol specificatie bevat de volgende componenten:

- *PDU formaten*: een definitie van alle PDUs en de manier waarop deze PDUs gecodeerd worden als reeksen van bits en bytes; en
- *protocol functies*: de relaties tussen (N)-service primitiva, (N)-PDUs en (N-1)-service primitiva, die samen het gedrag van de protocol entiteiten definiëren

Net als voor service definities zijn er veel verschillende notationele hulpmiddelen voor protocol specificaties. De meest gebruikte notaties zijn het toestandovergangendiagram en de

toestandstabel. Tabel 2 toont de representatie van een gedeelte van de Remote Operations Protocol specificatie met een toestandstabel. De toestandstabel definieert voor elke protocol entiteit een aantal toestanden, met in elke toestand een aantal mogelijke gebeurtenissen die leiden tot een volgende gebeurtenis en een nieuwe toestand. Bijvoorbeeld het service primitivum INVOKE-request is een mogelijke gebeurtenis in de toestand idle-initiator (0A), en heeft tot gevolg dat een INVRI PDU wordt verstuurd waarna de toestand await_RI_PDU (1A) wordt aangenomen. In de toestandstabel notatie voor protocol specificatie wordt dus onderscheid gemaakt tussen ‘oorzaak’ en ‘gevolg’ gebeurtenissen.

toestand gebeurtenis (oorzaak)	idle_ initiator (0A)	await_ RI_PDU (1A)	idle_ responder (0B)	await_ request (1B)	...
INVOKE-request	INVRI PDU 1A				
INVRI PDU			INVOKE- indication 1B		
RESULT-request				RESRI PDU 0B	
RESRI PDU		RESULT- indication 0A			
REJECT-request				REJRI PDU 0B	
REJRI PDU		REJECT- indication 0A			
...					

Tabel 2: Toestandstabel voor het Remote Operations Protocol

De toestandstabel is bedoeld om het gedrag van protocol entiteiten te representeren; PDU formaten moeten op een andere manier gedefinieerd worden. Een nadeel van de toestandstabel notatie voor het definiëren van protocol gedrag is dat de lagere protocol functies (de relaties tussen (N)-PDUs en (N-1)-service primitiva) niet worden gerepresenteerd.

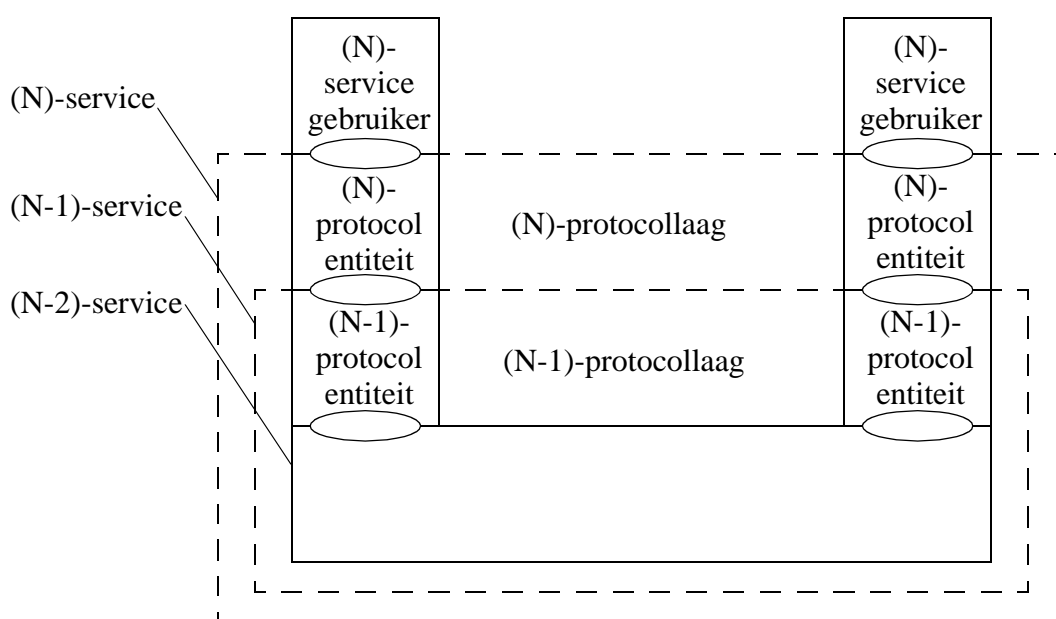
6 Ontwerpen met het service en protocol concept

Bij het ontwerpen van complexe gedistribueerde systemen is het gebruik van een ontwerpmethodede gebaseerd op het herhaaldelijk toepassen van het service en protocol concept onmisbaar. Bij deze ontwerpmethodede is het uitgangspunt een service definitie; deze service correspondeert met de geïntegreerde functionaliteit die het te ontwerpen systeem moet leveren. De eerste ontwerpstep richt zich op het decomponeren van deze service, zodanig dat een protocol resulteert die steunt op een onderliggende service. De keuze van de onderliggende service is een belangrijk aspect van de ontwerpstep: ontwerpproblemen die te maken hebben met het implementeren van deze service hoeven nu niet beschouwd te worden. Het resulterende

protocol vormt een *protocollaag* die steunt op een onderliggende service en die de vereiste service levert.

De tweede ontwerpstep richt zich op het decomponeren van de onderliggende service uit de eerste ontwerpstep. Ook tijdens deze ontwerpstep kan de complexiteit beperkt blijven door een (nieuwe) onderliggende service te kiezen. Dit proces kan zich herhalen totdat een onderliggende service wordt gevonden die correspondeert met de service geleverd door een reeds ontwikkeld protocol. Het resultaat van de ontwerpmethodode is een *hiërarchie* van protocollagen, ofwel een *gelaagde* protocolarchitectuur.

Figuur 5 toont een gelaagde protocolarchitectuur. De verschillende serviceniveaus en protocollagen in deze architectuur worden onderscheiden door middel van symbolische namen: (N), (N-1), (N-2), etc. ((N) representeert het hoogste niveau).



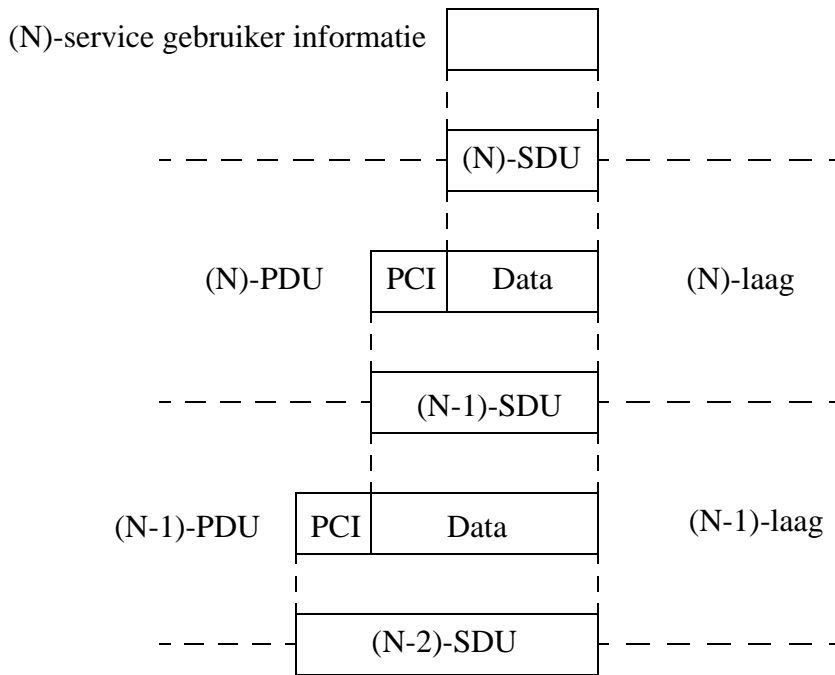
Figuur 5: Gelaagde protocolarchitectuur

In een gelaagde protocolarchitectuur bestaat er tevens een hiërarchische relatie tussen PDUs. Dit is geïllustreerd in figuur 6³. De afbeelding tussen PDUs en SDUs zorgt ervoor dat elke protocollaag zijn eigen PCI heeft, en niet op de hoogte is van de PCI die gebruikt wordt in andere lagen. Deze benadering illustreert de *scheiding van ontwerpzorgen* ('separation of concerns') die met de ontwerpmethodode wordt bereikt.

Feitelijk zijn alle netwerkarchitecturen gelaagde protocolarchitecturen; sommige netwerkarchitecturen hebben behalve een gelaagde structuur nog een structureringsmechanisme waarmee deelprotocollen kunnen worden geïsoleerd en die leidt tot een *modulaire* protocolarchitectuur.

De ontwerpmethodode zoals hierboven beschreven kan ook gebruikt worden om een modulaire protocolarchitectuur af te leiden. Het voornaamste verschil is het uitgangspunt: bij het afleiden van een modulaire protocolarchitectuur wordt begonnen met het identificeren van generieke

3. Voor de eenvoud worden hier alleen één-op-één afbeeldingen tussen SDUs en PDUs beschouwd.



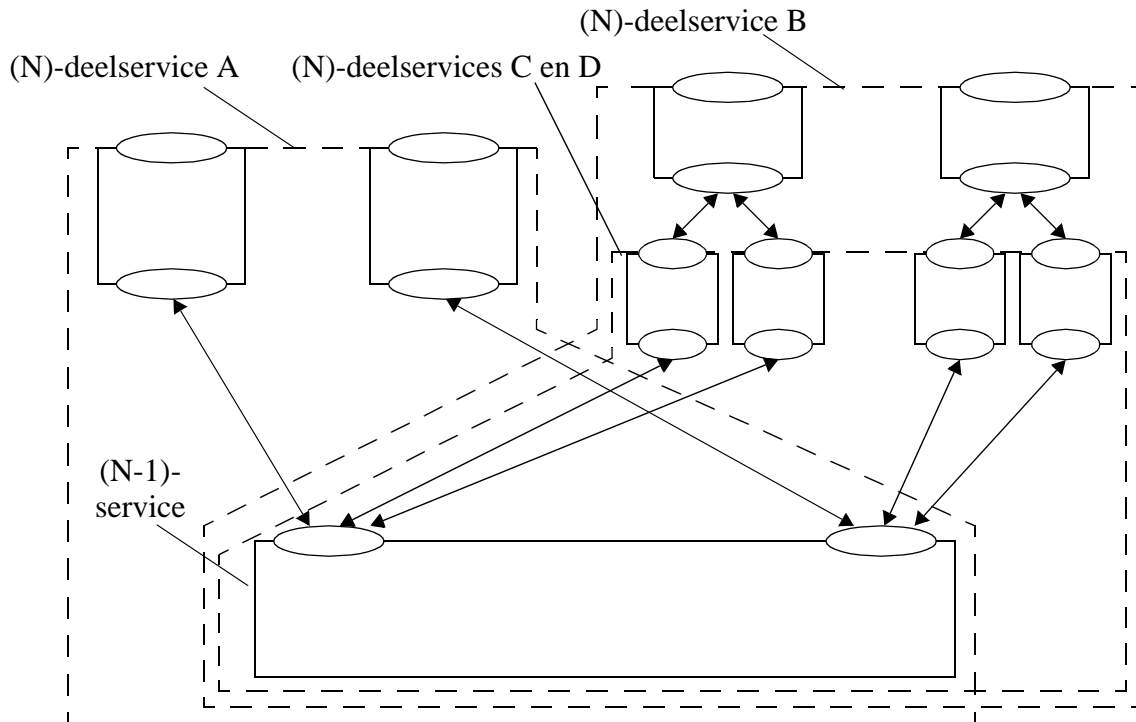
Figuur 6: Relatie tussen PDUs en SDUs in een gelaagde protocolarchitectuur

deelservices voor verschillende gebruikersomgevingen. Elk van de deelservices wordt op dezelfde manier gedeclineerd als een 'hele' service, met dien verstande dat bij de decompositie van verschillende deelservices gestreefd moet worden naar gemeenschappelijke onderliggende services.

De OSI applicatielaagstructuur is een voorbeeld van (een conceptuele basis voor) een modulaire protocolarchitectuur. De verschillende gebruikersomgevingen zijn hier de verschillende applicatiegebieden, die elk een specifieke applicatieservice vereisen. Het is mogelijk een generieke applicatieservice voor een bepaald applicatiegebied samen te stellen met behulp van een combinatie van generieke deelservices. Een deelservice wordt geleverd door een deelprotocol dat gebruik maakt van een onderliggende service, in veel gevallen de OSI presentatie service maar soms ook een andere deelservice. De protocol entiteiten in een deelprotocol worden, enigzins verwarrend, *Application Service Elements* (ASEs) genoemd volgens de OSI terminologie. ASEs zijn de bouwstenen voor generieke applicatie services; het identificeren van geschikte ASEs (dus eigenlijk van deelservices) is echter moeilijk, omdat het kennis van meerdere applicatiegebieden vereist.

Figuur 7 illustreert de toepassing van de ontwerpmethodede voor het afleiden van een modulaire protocolarchitectuur: (N)-deelservice A wordt gedeclineerd in een (N)-deelprotocol A en een onderliggende (N-1)-service; (N)-deelservice B wordt gedeclineerd in een (N)-deelprotocol B en twee onderliggende (N)-deelservices C en D (d.w.z. deelprotocol B maakt gebruik van deelservice C en van deelservice D). Decompositie van deze onderliggende deelservices levert voor elk van de deelservices een eigen deelprotocol maar een gemeenschappelijke onderliggende (N-1)-service.

De toepassing van de ontwerpmethodede zoals tot nu toe besproken is puur 'top-down'. In praktijk zal gelijktijdig een 'bottom-up' benadering gevolgd worden, bijvoorbeeld het configureren van



Figuur 7: Decompositie van deelservices

beschikbare protocollen voor het leveren van een service die als onderliggende service kan dienen in een decompositie volgens de ontwerpmethode.

7 Literatuur

- [1] ISO. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model*. International standard IS 7498, 1984 (ook ITU-T Recommendation X.200).
- [2] ISO/IEC. *Information Technology - Open Systems Interconnection - Application Layer Structure*. International standard IS 9545, 1989.
- [3] P.F. Linington. Fundamentals of the layer service definitions and protocol specifications. *Proceedings of the IEEE*, Vol. 21, No. 12, 1983, 1341-1345.
- [4] C.A. Vissers, L. Logrippo. The importance of the service concept in the design of data communication protocols. *Protocol Specification, Testing, and Verification, V* (Elsevier Science Publishers, 1986) 3-17.
- [5] H. Zimmermann. On protocol engineering. *Information Processing '83* (Elsevier Science Publishers, 1983) 283-292.

Auteurs

Marten van Sinderen is werkzaam als Universitair Docent bij de faculteit Informatica van de Universiteit Twente. Zijn onderzoek op het gebied van toepassingsgerichte protocollen is ondergebracht bij het Centrum voor Telematica en Informatie Technologie (CTIT), een onderzoeksinstituut van de Universiteit Twente, dat zich richt op multidisciplinair en interdisciplinair onderzoek op het gebied van telematica en informatietechnologie.

Luís Ferreira Pires is werkzaam als Universitair Hoofd Docent bij de faculteit Informatica van de Universiteit Twente. Zijn onderzoek op het gebied van architectuur en implementatie van gedistribueerde systemen is ondergebracht bij het CTIT (zie boven).