Het web is een ideaal medium voor de verspreiding van informatie. Daarnaast kan het dienen als infrastructuur voor applicaties. Dit laatste biedt bedrijven een aantal belangrijke voordelen zoals bijvoorbeeld een goede beschikbaarheid en toegankelijkheid richting klanten en medewerkers. Ook binnen ABN-AMRO wordt de kracht van het web onderkent en zijn er reeds verschillende 'webapplicaties' ontwikkeld. Daarnaast wordt er binnen ABN-AMRO nagedacht over de precieze structuur (architectuur) van dergelijke applicaties. Dit artikel beschrijft het web als infrastructuur voor het realiseren van applicaties en de voor- en nadelen die daarmee samenhangen. Daarnaast wordt ingegaan op de aanbevolen applicatie-architectuur voor het web. De informatie is gebaseerd op ervaringen en onderzoek van het kennisteam voor webapplicatie-ontwikkeling binnen ABN-AMRO.

thema

Een applicatie-architectuur voor het web bij De Bank

De pro´s en contra´s van toestandsloosheid

Het web is gebaseerd op een relatief eenvoudig model (zie figuur 1). Er zijn in dit model drie belangrijke onderdelen te onderkennen. In de eerste plaats is dat de webbrowser; de applicatie waarmee de gebruiker gegevens opvraagt en waarmee hij door deze gegevens kan bladeren. De webbrowser communiceert met de webserver. Deze applicatie is geïnstalleerd op een systeem in Internet of een intranet en stelt gegevens beschikbaar aan gebruikers. De communicatie tussen de webbrowser en webserver is toestandsloos; voor ieder verzoek wordt een nieuwe verbinding opgebouwd waarbij de webserver tussen verzoeken geen gebruikersspecifieke informatie bijhoudt. Tenslotte is er een informatiebron die de gegevens aan de webserver beschikbaar stelt. In veruit de meeste gevallen zijn dit gewoon (HTML-) bestanden waarin de gegevens zijn opgenomen. In dat geval wordt gesproken over een website. Zo'n website kan bijvoorbeeld een goed (en goedkoop) alternatief medium zijn voor het verspreiden van een interne nieuwsbrief onder medewerkers. Een nog interessanter geval is als de informatiebron een applicatie is, waardoor de mogelijkheden vrijwel onbegrensd zijn. In dat geval wordt gesproken over een webapplicatie. De definitie van een webapplicatie is dus: een applicatie die communiceert met de gebruiker via een webbrowser en webserver. Vrijwel alle administratieve applicaties kunnen als webapplicatie gerealiseerd worden.

De communicatie tussen webbrowser en webserver is toestandsloos

DUIZEND WERKPLEKKEN

Er is een aantal belangrijke voor- en nadelen verbonden aan applicaties die worden gerealiseerd volgens het webmodel ten opzichte van de traditionele monolithische en client/server-architectuur. De belangrijkste voordelen zijn de grote bereikbaarheid van webapplicaties, het oplossen van het softwaredistributie-probleem en de goede schaalbaarheid. Deze voordelen stellen een bedrijf bijvoorbeeld in staat om op een eenvoudige manier alle (potentiële) klanten te bereiken. De belangrijkste nadelen hebben voornamelijk betrekking op het protocol (HTTP) dat gebruikt wordt voor communicatie tussen webbrowser en webserver. Deze nadelen zullen echter voor gebruikers grotendeels verborgen blijven.

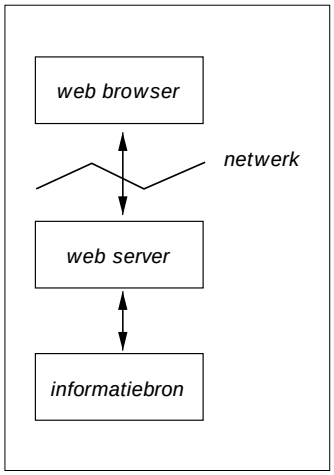
Het web is voor vrijwel iedereen bereikbaar doordat het gebaseerd is op geaccepteerde, open standaarden als HTML, HTTP en TCP/IP. Zo is TCP/IP voor vrijwel elk platform beschikbaar en is er voor vrijwel elk platform een webbrowser die HTTP en HTML implementeert. Dit alles leidt ertoe dat een webapplicatie automatisch beschikbaar is voor vrijwel alle mogelijke gebruikers en dat het schrijven van een variant voor elk platform niet nodig is.

Softwaredistributie heeft betrekking op het verspreiden van (nieuwe versies van) applicaties en is voor veel organisaties een groot probleem. Zo kost het installeren van een nieuwe versie van een e-mail-programma op een paar duizend werkplekken zoveel tijd en geld dat veel bedrijven noodgedwongen door blijven werken met oude versies. Dit probleem wordt opgelost door het gebruik van webtechnologie. Installatie van een webapplicatie hoeft maar op één plaats; gebruikers krijgen altijd automatisch de laatste versie.

Er wordt in de communicatie tussen webbrowser en webserver geen toestand bijgehouden. Alle tijdelijk benodigde gegevens bevinden zich dus binnen de webbrowser. Dit heeft als grote voordeel dat de server ontlast wordt van het bijhouden van deze gegevens en het opruimen daarvan. Door deze toestandsloosheid is het eenvoudig verzoeken te verdelen over meerdere webservers, waardoor de schaalbaarheid van webapplicaties goed is.

VERVELEND De belangrijkste nadelen van het webmodel hebben betrekking op het gebruikte protocol (HTTP) voor communicatie tussen webbrowser en webserver. Zoals eerder aangegeven is deze communicatie toestandsloos. Voor het versturen van enkele teksten is dit protocol goed geschikt. Applicaties hebben echter vaak behoefte aan een meer permanente verbinding van de gebruiker met de server waarbij in de server toestand wordt bijgehouden over de gebruiker en de fase waarin deze zich bevindt in een dialoog met het systeem. Dit om bijvoorbeeld verschillende verzoeken van een gebruiker binnen eenzelfde transactiecontext uit te voeren. Doordat er geen verbinding is tussen webbrowser en webserver, is het daarnaast voor een webapplicatie niet mogelijk om te detecteren dat een gebruiker de applicatie afsluit. Dit alles legt een zware last op de schouders van de server die verantwoordelijk is voor het associëren van een gebruiker met een toestand, het repliceren van deze toestand over meerdere webservers en het gebruiken van time-outs om gegevens op te ruimen en transacties af te breken. Tenslotte kost het extra tijd om voor elk verzoek een nieuwe verbinding op te bouwen. Dit laatste probleem wordt voor een deel verholpen door een nieuwe versie van het protocol, waarbij over een verbinding meerdere verzoeken kunnen worden afgehandeld.

De webapplicatie oefent geen controle uit op de webbrowser. Dit leidt tot een probleem als gebruikers gaan navigeren. Zo kan een gebruiker vragen om een vorige pagina, een nieuw browser-venster openen en een pagina opnieuw opvragen. Op het eerste gezicht misschien onschuldige operaties, maar voor applicaties buitengewoon vervelend af te handelen. Wat betekent het bijvoorbeeld als een gebruiker een pagina met een reeds eerder ingevuld formulier opnieuw opvraagt? Als er alleen informatie wordt opgevraagd is dat niet een groot probleem. Als het gevolg is dat er twee keer geld van een bankrekening wordt afgeboekt dan is dat echter vervelender.



FIGUUR 1: Het web

Een laatste probleem van webapplicaties die alleen gebruik maken van HTML is dat de gebruikersinterface veel beperkingen heeft. Dergelijke webapplicaties zien er gewoonweg niet echt mooi uit of zijn beperkt in hun mogelijkheden. Zo is het aantal mogelijke interfacecomponenten zoals knoppen en invoervelden beperkt en kan er geen logica worden uitgevoerd om bijvoorbeeld een veld te valideren. Indien er naast HTML ook componenten of scripts in een pagina zitten dan worden deze beperkingen opgeheven. In dat geval moet er echter op gelet worden dat geen gebruikers worden uitgesloten en dat de presentatielaag niet te dik wordt. Dit omdat componenten en scripts niet door alle webbrowsers op een zelfde manier ondersteund worden en misbruikt kunnen worden om applicatielogica in te definiëren. Webapplicaties die beschikbaar worden gesteld via Internet kunnen daarom beter alleen van HTML gebruik maken.

KERNBEGRIPPEN Een software-architectuur is een kijk op een softwaresysteem waarbij de onderverdeling van het systeem in subsystemen/componenten, hun verantwoordelijkheden en interacties centraal staan. Eenvoudig gezegd is software-architectuur een beschrijving van de structuur van software. Een applicatie-architectuur beschrijft de architectuur van een specifieke applicatie. Hierna wordt de aanbevolen architectuur voor webapplicaties, oftewel webapplicatie-architectuur, beschreven. Door applicaties te ontwikkelen die voldoen aan deze architectuur wordt een aantal van de eerder geschetste problemen opgelost. Hierbij spelen de drie begrippen componentgebaseerd, drie-lagen en meerdere-kanalen een belangrijke rol. Naast deze kernbegrippen wordt dieper ingegaan op een aantal componenten die deel uitmaken van de applicatie-architectuur. De resulterende architectuur is weergegeven in figuur 2. Merk op dat een groot aantal webapplicatie-ontwikkeltools en omgevingen reeds in meerdere of mindere mate tegemoet komen aan de beschreven architectuur. De architectuur zou dan ook gebruikt kunnen worden als evaluatie- en selectiemiddel voor deze tools en omgevingen.

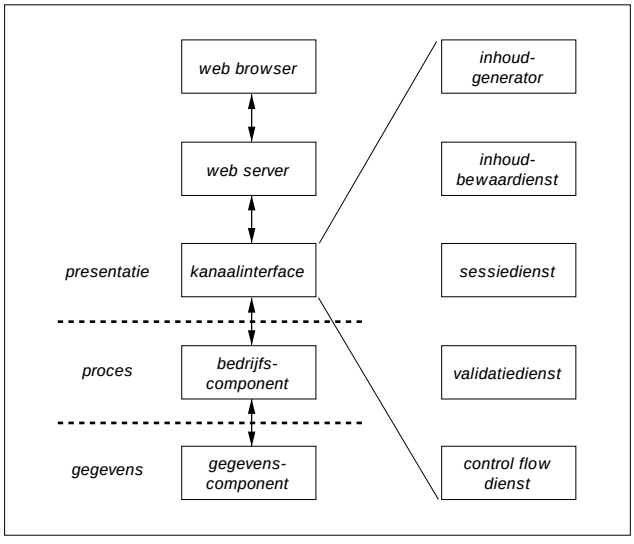
AFHANKELIJKHEID Door applicaties onder te verdelen in (binaire) componenten ontstaan onafhankelijke, herbruikbare eenheden [SZYPERSKI]. Infrastructurele componenten worden hierbij ook wel diensten genoemd. Het wijzigen of vervangen van een component is dan geen probleem zolang deze dezelfde operaties (functies) aan blijft bieden. Een component-gebaseerde architectuur zorgt dus voor applicaties die beter onderhoudbaar en flexibeler zijn dan monolithische applicaties. Uiteindelijk zal dit resulteren in lagere ontwikkel- en onderhoudskosten van dergelijke applicaties.

Het drie-lagen model is een standaard opdeling in componenten, waarbij drie soorten componenten worden onderscheiden [KIRTLAND]. Ieder soort component heeft eigen verantwoordelijkheden en communiceert zoveel mogelijk naar een ondergelegen laag. Het drie-lagen model onderscheidt de volgende lagen:

- Presentatielaag: componenten die presentatie en gebruikersinteractie afhandelen.
- Proceslaag: componenten die (elementaire) bedrijfsfuncties implementeren.
- Gegevenslaag: componenten die belast zijn met het beheren van gegevens.

Door deze drie lagen te onderscheiden worden presentatie, proces en gegevens van elkaar losgekoppeld, waardoor zij (tot op zekere hoogte) onafhankelijk van elkaar kunnen wijzigen en worden hergebruikt. Dit wordt nog eens versterkt doordat er geen afhankelijkheid bestaat van lagere naar hogere lagen. Uiteindelijk wordt de onderhoudbaarheid en flexibiliteit van de applicatie groter.

Dit drie-lagen model kan natuurlijk worden toegepast op webapplicaties. Hierbij is primair van belang de presentatielaag te scheiden van de proceslaag. In de presentatielaag worden verzoeken vanaf de webserver vertaald naar een formaat dat de proceslaag begrijpt en worden de resultaten geformatteerd in HTML. De presentatielaag bestaat idealiter zoveel mogelijk uit generieke, herbruikbare componenten. De daadwerkelijke applicatielogica bevindt zich in de proceslaagcomponenten, ook wel bedrijfscomponenten genoemd. Deze componenten worden gebruikt door de presentatielaag. Gegevenslaagcomponenten kunnen worden ingevuld door (component-gebaseerde abstracties op) databasemanagementsystemen.



FIGUUR 2: Aanbevolen webapplicatie-architectuur

Het drie-lagen-model wordt gezien als een verbetering op het veel toegepaste (twee-laagse) client/server-model. Het is tegenwoordig dan ook uitgangspunt voor veel applicaties en wordt in toenemende mate ondersteunt door ontwikkeltools en omgevingen.

SCHEIDING Naast webapplicaties zijn er ook allerlei typen applicaties die geen gebruik maken van een webbrowser zoals client/server-applicaties. Bovendien zijn er naast personal computers meer apparaten van waaruit met applicatielogica kan worden gecommuniceerd. Denk aan netwerkcomputers, elektronische organisers en mobiele telefoons. Een applicatie zou bijvoorbeeld kunnen worden benaderd via een mobiele telefoon waarbij de menselijke stem gebruikt wordt om de applicatie opdrachten te geven en waarbij de resultaten worden uitgesproken. Idealiter zouden alle applicatietypen en apparaten (ook wel kanalen genoemd) gebruik maken van dezelfde applicatielogica. Een dergelijk hergebruik zou leiden tot een kortere en dus goedkopere applicatie-ontwikkeling.

Het drie-lagen model zorgt voor een scheiding van applicatielogica in presentatie, proces en gegevens. Door deze scheiding aan te houden zouden componenten uit de proceslaag een nieuw kanaal kunnen ondersteunen door een nieuwe presentatielaagcomponent te maken. Een dergelijk presentatielaagcomponent vormt daarbij als het ware de interface naar het nieuwe kanaal en wordt daarom ook wel de kanaal-interface genoemd. Door kanaal-interfaces te definiëren bovenop de proceslaag ontstaat een meerdere-kanalen architectuur, ook wel multi-channel architectuur genoemd.

NIET-ONTWIKKELAAR Bedrijfscomponenten vormen de kern van een componentgebaseerde architectuur. In deze componenten wordt de eigenlijke functionaliteit gedefinieerd. Bedrijfscomponenten kunnen overeenkomen met objecten als bijvoorbeeld ‘klant’ of ‘bankrekening’, maar kunnen ook een meer functionele rol hebben. Denk bijvoorbeeld aan een component voor het controleren van de kredietwaardigheid van een klant. Er zijn geen web-specifieke zaken die een rol spelen bij bedrijfscomponenten, aangezien ze ook gebruikt moeten kunnen worden in andere kanalen dan het webkanaal. Er zijn wel algemene richtlijnen voor componenten. Zo zouden componenten generiek en samenhangend moeten zijn en weinig koppeling met andere componenten moeten hebben om hergebruik te bevorderen. Bovendien stellen andere componenten eisen aan bedrijfscompo-

nenten. Zo moet een component speciale maatregelen nemen om veilig te zijn en transacties te ondersteunen.

De inhoudgenerator is verantwoordelijk voor het genereren van de uiteindelijke inhoud van pagina's die in de webbrowser van de gebruiker getoond zullen worden. Het component dient hiertoe de resultaten van bedrijfscomponenten te formatteren in HTML. Hij kan ook onafhankelijk van bedrijfscomponenten functioneren om bijvoorbeeld een standaard-uiterlijk aan webpagina's te geven. Merk op dat ook andere inhoud kan worden gegenereerd, bijvoorbeeld in de vorm van XML. Het streven is naar een generieke inhoudgenerator, die wordt gestuurd door bijvoorbeeld templates. Het maken van templates is eenvoudiger dan programmeren, zodat zelfs een niet-ontwikkelaar zoals een interaction designer een nieuwe presentatie voor een bedrijfscomponent kan definiëren.

De inhoud-bewaardienst is verantwoordelijk voor het bewaren van uitvoer van de inhoud generator. Niet alleen kan hierdoor snel een antwoord worden gegeven op een herhaaldelijk gedaan verzoek; het maakt het ook mogelijk meerdere uitvoerpagina's te genereren voor een verzoek aan een bedrijfscomponent. Het weggooien van bewaarde inhoud kan op allerlei verschillende manieren plaatsvinden en zou instelbaar moeten zijn, bijvoorbeeld om opslagruimte te sparen. Daarnaast is het niet altijd toegestaan resultaten te bewaren omdat elke aanroep naar een webapplicatie een ander resultaat kan hebben. Dit component kan gezien worden als server-side tegenhanger van de cache in de webbrowser van de gebruiker.

Met de sessiedienst wordt het mogelijk één van de belangrijkste problemen van het web, de toestandsloosheid, op te heffen. De dienst biedt bedrijfscomponenten hiertoe de illusie van een verbinding (sessie) met een gebruiker. Daarnaast kan de dienst gebruikt worden om toestand van de gebruiker bij te houden. Dit maakt het bijvoorbeeld mogelijk om bij te houden welke acties een gebruiker heeft ondernomen zodat de applicatie hierop kan anticiperen. Als een gebruiker vaak dezelfde fout maakt dan zou de applicatie hem hierop kunnen wijzen en aanbevelingen kunnen doen om verdere fouten te voorkomen. Een geavanceerde sessiedienst maakt het mogelijk de toestand van de gebruiker op meerdere plaatsen (servers) bij te houden, waardoor een gebruiker eenvoudig een verzoek aan een andere server kan doen als de eerste bezet is (load-balancing). De sessiedienst dient een browser van een gebruiker te kunnen identificeren, maar dient ook in staat te zijn individuele browser-vensters te onderscheiden.

VEILIGHEIDSDIENST Alle gegevens die gebruikers invoeren in velden op formulieren dienen te worden gecontroleerd op correctheid. Verkeerde waarden kunnen immers onvoorspelbaar gedrag geven, zoals het crashen van een applicatie. Dergelijke validaties moeten in ieder geval in een bedrijfscomponent zelf worden uitgevoerd, maar kunnen om allerlei redenen ook eerder plaatsvinden. De validatiedienst maakt het mogelijk validaties in de vorm van validatieregels op een generieke manier te definiëren en te controleren. Validatieregels kunnen variëren van eenvoudige typecontroles tot bedrijfsregels, zoals het testen of een waarde binnen een bepaald bereik ligt of dat een waarde een valide bankrekeningnummer is. Validatieregels worden opgeslagen in een bedrijfscomponent. Het valideren gebeurt door de validatiedienst aan te roepen, waarna deze de validatieregels ophaalt uit de bedrijfscomponent (en bewaart). Ook is het mogelijk de validatiedienst op verschillende plaatsen in te zetten, zoals op de computer van de gebruiker en de webserver, om validaties ook eerder in het proces uit te voeren. Dit maakt het mogelijk een gebruiker direct te attenderen als een veld verkeerd wordt ingevuld.

Eerder werd aangegeven dat het feit dat gebruikers op allerlei manieren door gegevens kunnen navigeren een probleem oplevert voor applicaties. Deze problemen kunnen worden opgelost door een generieke control-flow-

"De sessiedienst heft het probleem van de toestandsloosheid op"

dienst. Hierbij wordt het verloop van de applicatie gedefinieerd en voor ieder verzoek gecontroleerd. Als een gebruiker een pagina opvraagt die niet meer valide is dan zal deze dienst dat dus signaleren. Daarnaast zou deze dienst voor alle acties mogelijke vervolgacties kunnen genereren, bijvoorbeeld in de vorm van een menu. De dienst is niet verantwoordelijk voor het splitsen en combineren van verzoeken. Dit splitsen en combineren wordt typisch in een nieuw component gedefinieerd, maar kan ook gebruik maken van generieke middleware zoals een message-broker.

Bovengenoemde standaardcomponenten (met uitzondering van de bedrijfscomponenten) vormen tezamen het kanaalinterface en zijn specifiek gericht op het webkanaal, alhoewel zij ook generieker gedefinieerd zouden kunnen worden. Hierbij bestaat het webkanaal uit de webbrowser en webserver. Merk op dat het ook valide zou zijn alleen de webbrowser als webkanaal te definiëren, waarbij de webserver deel uit maakt van de kanaalinterface. Naast de web-specifieke componenten zijn er ook een veelheid van algemene componenten te definiëren. Voorbeelden hiervan zijn een transactiedienst, load-balancing dienst, fail-over-dienst, veiligheidsdienst

Lees verder op pagina 50

Vervolg van pagina 17

en databasedienst. Merk op dat veel van deze diensten tegenwoordig beschikbaar zijn in applicatieservers en object-transaction-monitors, ook wel aangeduid als component oriented middleware [SESSIONS]. Deze diensten zullen hier niet nader worden toegelicht, aangezien zij niet web-specifiek zijn.

HOOGWAARDIG Het web biedt een eenvoudig model voor het realiseren van applicaties. De voordelen van dit model zijn de bereikbaarheid, automatische softwaredistributie en de goede schaalbaarheid. De belangrijkste nadelen hebben voornamelijk betrekking op het HTTP-protocol. Veel van deze nadelen kunnen worden opgelost door het gebruik van een webapplicatie-architectuur. Deze architectuur is component-gebaseerd, opgedeeld in drie lagen en ondersteunt meerdere kanalen. Daarnaast bestaat de architectuur uit een aantal standaardcomponenten. Door gebruik te maken van deze architectuur kan een ontwikkelaar snel kwalitatief hoogwaardige, goed onderhoudbare en flexibele applicaties ontwikkelen. Het best kan deze architectuur worden gerealiseerd door een raamwerk waar ontwikkelaars gebruik van kunnen maken. Veel van de verkrijgbare webapplicatie-ontwikkeltools en omgevingen bieden vergelijkbare diensten en zouden dus kunnen worden geselecteerd op basis van de beschreven applicatie-architectuur.

Drs. Danny Greefhorst
is senior onderzoeker bij het Software Engineering Research Centre. Hij is gedurende langere tijd verbonden geweest als adviseur aan het kennisteam voor webapplicatie-ontwikkeling bij ABN-AMRO.

Literatuur

D. Greefhorst, R. Koelman, Web Application Architecture, ABN-AMRO (Intern rapport), 1998
M. Kirtland, Designing Component Based Applications, Microsoft Press, 1999
I. Jacobson, UML Unified Process, Addison-Wesley, 1999
R. Sessions, Pain and misery, ObjectWatch Newsletter Number 17, 1998
C. Szyperski, Component Software, Addison-Wesley, 1998
J. Visser et al., IDC High Level Design, ABN-AMRO (Intern rapport), 1998
D. D'Souza, A. Wills, Objects, Components, and Frameworks with UML, The Catalysis Approach, 1998