De architectuur van Ajax ontrafeld

Een abstract perspectief

De laatste tijd trekt een nieuw type webapplicaties veel aandacht, namelijk Ajax-webapplicaties. Het is een antwoord op de beperkte interactiviteit van de bestaande *toestandloze* webinteracties. In deze nieuwe benadering is het interactiemodel gebaseerd op een enkele, uit diverse gebruikersinterfacecomponenten samengestelde pagina, met het doel webapplicaties veel interactiever te maken. In plaats van de hele pagina te verversen, vinden wijzigingen plaats op componentniveau.

Ali Mesbah en Arie van Deursen

1 Asynchronous JavaScript and XML.
2 SPIAR: Single Page Internet Application ARchitecture.

Zoals gedefinieerd door Garrett (2005), is Ajax[1] gebaseerd op presentatie middels standaarden als XHTML en CSS, dynamische weergave en interactie door gebruik van het Document Object Model (DOM), data-interactie en manipulatie, asynchrone communicatie met de server met XMLHttpRequest, en ten slotte JavaScript om dit alles te integreren. Bekende voorbeelden van Ajax-applicaties zijn onder meer Google Suggest, Gmail en de nieuwe versie van Yahoo! Mail.
De term Ajax is in februari 2005 geïntroduceerd. Er zijn inmiddels talloze frameworks en bibliotheken geschreven die gebaseerd zijn op Ajax, en er komen nog dagelijks nieuwe bij. Ook verscheen over dit onderwerp een groot aantal artikelen in professionele tijdschriften en op websites. Niet alleen nieuw ontwikkelde toepassingen maken gebruik van Ajax, ook al bestaande websites worden meer en meer voorzien van Ajax-technologie.
Voor de software engineer die overweegt Ajax-technologie toe te passen, blijft echter nog een reeks van vragen open. Heeft het gebruik van Ajax invloed op de architectuur? Hoe verhouden de diverse frameworks zich tot elkaar? Wat verbergen deze frameworks, en wat maken zij juist mogelijk? Convergeren de diverse frameworks, of gaan ze juist allemaal een andere kant op? Welke Ajax-elementen zullen blijven voortbestaan en welke aspecten van Ajax worden wellicht vervangen door elegantere oplossingen?
Om dit soort vragen te beantwoorden hebben we geprobeerd een wat abstracter perspectief op Ajax te ontwikkelen. Ondanks de enorme aandacht voor Ajax, is er een gebrek aan een goed gedefinieerde en samenhangende verzameling van formele architectuureigenschappen voor Ajax-webapplicaties. Om hun architectuurkenmerken bloot te leggen hebben we diverse Ajax-frameworks bestudeerd, waarvan we er drie behandelen aan de hand van de door ons ontwikkelde SPIAR-architectuurstijl[2]. Deze stijl combineert een serie gewenste eigenschappen (zoals interactiviteit en ontwikkelgemak) met vereisten waaraan een Ajax-architectuur moet voldoen (zoals component-gebaseerde gebruikersinterfaces en delta-communicatie tussen client en server) om deze eigenschappen te realiseren.

Ajax-frameworks

We hebben diverse Ajax-frameworks bestudeerd om hun architectuurkenmerken in kaart te bren-

Samenvatting

De laatste tijd staan de zogenaamde Ajax-webapplicaties volop in de belangstelling. Ze zijn een antwoord op de beperkte interactiviteit van de bestaande toestandloze webinteracties. De auteurs hebben diverse Ajax-frameworks bestudeerd om hun architectuurkenmerken te ontrafelen. In dit artikel bespreken zij de door hen ontwikkelde SPIAR-architectuurstijl aan de hand van drie van deze frameworks. Deze stijl combineert een serie gewenste eigenschappen met vereisten waaraan een Ajax-architectuur moet voldoen om deze eigenschappen te realiseren.

gen. Onze selectie bestond onder meer uit het veelgebruikte opensourceframework Echo2, de Google Web Toolkit (GWT) en het commerciële framework van het Amsterdamse Backbase. Deze frameworks delen – ondanks de verschillen – bepaalde gemeenschappelijke architectuureigenschappen en doelstellingen (zie kader). De frameworks realiseren deze doelstelling door het beschikbaar stellen van een bibliotheek van gebruikersinterfacecomponenten en een ontwikkelomgeving voor het creëren van herbruikbare componenten. Elke architectuur heeft een goed gedefinieerd protocol voor kleinschalige interactie tussen client-servercomponenten. Hierdoor wordt het dataverkeer over het netwerk significant verminderd, wat weer kan leiden tot betere responsetijden. De frameworks maken verder gebruik van *client-side processing*, wat kan resulteren in verhoogde gebruikersinteractiviteit, minder client-servercommunicatie en verminderde serverbelasting.

3 **Bekend als URL: de Uniform Resource Locator.**

Architectuurstijlen

Terminologie

Aansluitend op Perry en Wolf (1992) definiëren we *software-architectuur* als een configuratie van processing, connecting en data-elementen. Deze elementen begrenzen we in hun relaties om bepaalde gewenste architectuurkenmerken te realiseren. Als een verzameling begrenzingen voorkomt in meerdere concrete architecturen, noemen we zo'n verzameling een *architectuurstijl* (Fielding, 2002).

Gemeenschappelijke doelstellingen van de besproken frameworks:

- **het verbergen van de complexiteit van het ontwikkelproces van Ajax-applicaties, vaak een gecompliceerde en foutgevoelige taak;**
- **het verbergen van de incompatibiliteiten tussen verschillende webbrowsers en systemen;**
- **het verbergen van de complexiteit van client-servercommunicatie;**
- **realiseren van gewenste eigenschappen zoals interactiviteit en portabiliteit voor de eindgebruiker en ontwikkelgemak voor de ontwikkelaar.**

REST

Er bestaan al diverse netwerkgebaseerde architectuurstijlen zoals client-server- en n-tiersystemen. De meest complete en geschikte stijl voor het web is *Representational State Transfer* (REST), geïntroduceerd door Roy Fielding (Fielding, 2002). REST introduceert voor data en services het abstractieniveau van *Resources*, die door de client middels een bronnaam en adres[3] opgevraagd kunnen worden. REST richt zich op de eigenschappen waar het web aan zou moeten voldoen (zoals maximale netwerkperformance en minimale door de gebruiker ervaren wachttijden) en beschrijft een aantal eisen (zoals toestandloze data-uitwisseling) op de architecturale elementen die nodig zijn om deze eigenschappen te realiseren. De elegantie en eenvoud van REST hebben een belangrijke bijdrage geleverd aan het succes van het world wide web.

Een stijl voor Ajax

Er zijn diverse redenen waarom het, ondanks het succes van REST in het world wide web, minder voor de hand ligt om REST te gebruiken om de architectuur van Ajax-applicaties en -frameworks te beschrijven. Een belangrijke restrictie van REST betreft de eis *Uniform Interface*. Deze beperking maakt REST weliswaar geschikt voor grootschalige hypermediadataoverdracht, maar veel minder geschikt voor kleine data-interacties zoals die vereist zijn in Ajax. In tegenstelling tot REST, waarin de client een specifieke *hypergelinkte resource* opvraagt, verwacht de gebruiker in Ajax-applicaties een response op een bepaalde *actie*, net zoals in desktopapplicaties. Alle communicatie over en weer tussen client en server vindt

op een *synchrone request-responsestijl* plaats in REST. Ajax-applicaties vereisen een model waarin asynchrone communicatie mogelijk is. REST eist expliciet een *toestandloze* server. Dit betekent dat elk verzoek van de client geacht wordt alle informatie te bevatten die de server nodig heeft om het verzoek te begrijpen. Deze beperking verbetert de schaalbaarheid, maar brengt tegelijkertijd enkele nadelen ten opzichte van netwerkperformance en gebruikersinteractiviteit met zich mee, die relevanter zijn voor het ontwerpen van Ajax-architecturen.

SPIAR

Uit het voorgaande blijkt dat Ajax moeilijk in te passen is in REST. Om die reden hebben we een alternatieve architectuurstijl ontwikkeld, genaamd SPIAR (Mesbah, 2007). Deze stijl beschrijft de essentie van wat Ajax-frameworks bevatten, alsmede de voornaamste architecturale kenmerken. Bovendien schrijft de stijl architectuurbeperkingen voor om de gewenste kenmerken te waarborgen.

»Voor Ajax is er geen goede verzameling van formele architectuureigenschappen«

Architectuureigenschappen

Het startpunt van SPIAR zijn de volgende essentiële architectuureigenschappen (ook kwaliteitsattributen genoemd) van Ajax.

Gebruikersinteractiviteit

De bestaande literatuur over mens-machine-interactie definieert interactiviteit als de mate waarin deelnemers in een communicatieproces dat proces kunnen beïnvloeden. Het verbeteren van interactiviteit door kortere responsietijden en betere grafische weergave op componentenniveau is een van de peilers waarop Ajax rust.

Door de gebruiker ervaren vertraging

De door de gebruiker waargenomen vertraging is de periode tussen het opvragen van een stukje data en de eerste indicatie van het ontvangen ervan. Er zijn twee belangrijke manieren om de ervaren vertraging te verbeteren. De eerste is het verkorten van de tijd die nodig is om data heen en weer te sturen. De tweede manier is het toepassen van asynchrone interactie met de server, waardoor de gebruiker na elke actie meteen de controle terugkrijgt van de browser.

Netwerkperformance

Netwerkperformance wordt beïnvloed door *throughput* (de hoeveelheid dataverkeer per tijdseenheid op het netwerk) en *bandbreedte* (maximumhoeveelheid data die per seconde over een verbinding verstuurd kan worden). Het reduceren van de hoeveelheid en grootte van de verzonden data kan leiden tot een stijging van de netwerkperformance.

Eenvoud

De inspanning nodig om een webapplicatie te begrijpen, ontwerpen, implementeren of wijzigen duiden we met eenvoud aan.

Schaalbaarheid

In gedistribueerde omgevingen wordt schaalbaarheid gedefinieerd door de mate waarin een softwaresysteem om kan gaan met een groeiend aantal componenten. In webgebaseerde applicaties bepaalt de mate waarin een client door verschillende servers bediend kan worden de schaalbaarheid van het systeem.

Portabiliteit

Portabiliteit is de mogelijkheid om software van de ene omgeving naar de andere omgeving over te zetten en daar te gebruiken, met minimale inspanning. Het gebruik van webstandaarden en standaardbrowsers heeft een gunstig effect op portabiliteit.

Zichtbaarheid

Zichtbaarheid wordt bepaald door de mate waarin een externe intermediair in staat is om de interactie tussen twee componenten te begrijpen. De zichtbaarheid in client-serverinteracties bij de huidige Ajax-frameworks is beperkt, omdat elk framework zijn eigen protocol toepast.

Architectuurelementen

Nu we vastgesteld hebben aan welke architecturale eigenschappen Ajax-applicaties dienen te voldoen, kunnen we onderzoeken wat voor proceselementen, data-elementen en verbindingselementen nodig zijn om deze eisen te realiseren. Hieronder bespreken we deze drie categorieën van elementen. Figuur 1 toont de interactie tussen verschillende componenten nadat een initiële pagina is opgevraagd.

Proceselementen

De voor de gebruiker direct zichtbare procescomponent is de webbrowser. Een browser ondersteunt standaarden zoals HTTP, HTML, CSS, JavaScript en het Document Object Model (DOM). De laatste wordt binnen de browser gebruikt als representatiemodel voor een webpagina en voor het verwerken van de gebruikersinterface.

In Ajax-applicaties draait binnen de webbrowser een *Ajax*-engine. Deze engine wordt bij het eerste contact automatisch geladen en heeft geen extra functionaliteit (zoals plug-ins) nodig om te functioneren. De engine is verantwoordelijk voor de initialisatie en manipulatie van het representatiemodel en behandelt de *events* die gegenereerd worden door de gebruiker. De engine communiceert met de server en is in staat acties aan de kant van de client uit te voeren (*client-side processing*).

De *serverapplicatie* op de server accepteert HTTP-requests van de clients en handelt deze af. Daartoe maakt het gebruik van de *serviceprovider* die de businesslogica bevat en toestandswijzigingen door kan voeren aan de hand van binnenkomende acties. De service-provider heeft toegang tot alle resources (zoals een database of externe webservices).

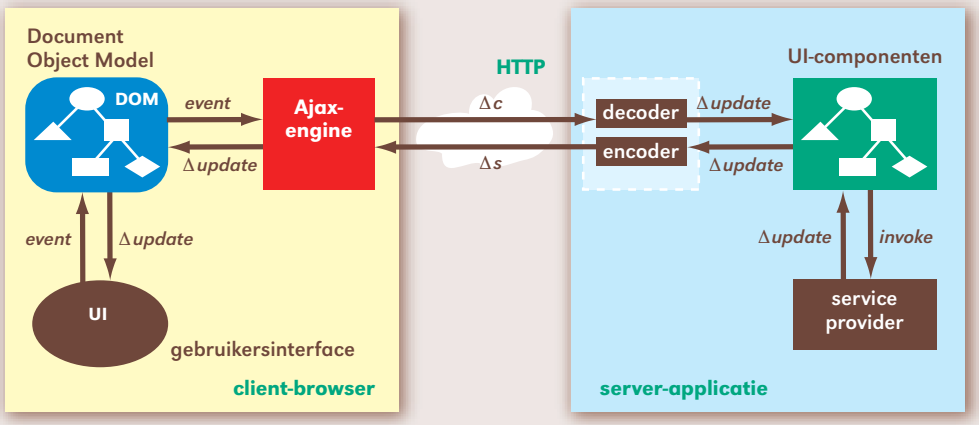
De server houdt een verzameling *gebruikersinterfacecomponenten* bij. Elke component is in staat om het corresponderende gedeelte van het representatiemodel aan de kant van de client te produceren. Er bestaan verschillende aanpakken waarmee de client-userinterfacecode gereproduceerd kan worden. De Google Web Toolkit bijvoorbeeld genereert de volledige clientcode tijdens het compileren van de Java-componenten op de server. Echo2 daarentegen genereert de componenten in runtime en houdt een boom van componenten bij aan zowel de client- als de serverkant. Aan elke component kunnen listeners hangen die de functionaliteit van de serviceprovider kunnen aanroepen.

Communicatie tussen client en server is niet gebaseerd op het over en weer sturen van de volledige pagina, maar op het verzenden van slechts de verschillen (delta's) tussen opeenvolgende pagina's. Uitgaande en binnenkomende deltaberichten worden verwerkt door de elementen *Delta Encoder* en *Delta Decoder*. Dit is het punt waarop het communicatieprotocol tussen de client en de server wordt gedefinieerd en verborgen achter een interface. Deze op delta's gebaseerde vorm van communicatie verbetert de door de gebruiker waargenomen vertraging en netwerkperformance.

Data-elementen

De proceselementen van SPIAR maken gebruik van een reeks data-elementen, de SPIAR-data-elementen. Een daarvan is het element *representatie* dat net als in REST alle relevante mediatypes omvat, zoals XHTML, CSS en plaatjes.

Het *representatiemodel* is een runtime abstractie van de structuur van een gebruikersinterface binnen de webbrowser. De browser gebruikt



Figuur 1. **Runtime perspectief op een op SPIAR gebaseerde architectuur**

hiertoe het standaard Document Object Model, dat een belangrijke rol speelt in Ajax-applicaties. Door dynamische manipulatie van dit representatiemodel wordt het bouwen van interactieve webapplicaties mogelijk. Sommige frameworks, zoals Backbase, gebruiken een domeinspecifieke taal om het representatiemodel declaratief te definiëren. Andere frameworks als de GWT gebruiken een directe benadering door middel van JavaScript.

Communicatie tussen client en server verloopt via *deltaberichten*. Binnen de client maakt de Ajax-engine een *client delta* aan om de toestandsovergangen en de bijbehorende acties aan de server door te geven. De server verwerkt dit bericht en maakt een *server delta* aan, waarin onder meer veranderingen in de gebruikersinterface (toon resultaten, nieuwe button erbij, enzovoort) kunnen worden weergegeven. Er worden verschillende formaten voor deze data-elementen gebruikt, zoals XML, BXML van Backbase, en de JavaScript Object Notation (JSON).

Connectoren

Events zijn de basis van het interactiemodel in Ajax-applicaties. Een event wordt geïnitieerd door elke actie van de gebruiker op de gebruikersinterface en doorgegeven aan de *Ajax*-engine. Afhankelijk van het type event kan een (a)synchroon verzoek naar de server of een partiële update van de gebruikersinterface plaatsvinden. Op de server kan een event ervoor zorgen dat een service daadwerkelijk aangeroepen wordt, ofwel direct, ofwel via de *event listener* van een corresponderende gebruikersinterfacecomponent.
De *Delta Connectoren* zijn lichtgewicht communicatiekanalen over HTTP die de Ajax-engine met de server verbinden. *Delta updates* worden gebruikt om het representatiemodel op de client- en de gebruikersinterfacecomponenten op de server in overeenstemming te brengen met de doorgevoerde toestandswijzigingen. Deze updates vinden doorgaans plaats in de vorm van procedure-aanroepen.

Architectuurrestricties (constraints)

Architectuurrestricties worden gebruikt als beperking van de rollen van de architectuurelementen om gewenste architectuureigenschappen te garanderen. Tabel 1 is een overzicht van de restricties en bijbehorende geïnduceerde eigenschappen. SPIAR berust op de volgende restricties.

Asynchrone interactie

Asynchrone communicatie stelt de gebruiker in staat op een willekeurig moment een verzoek naar de server te sturen en onmiddellijk de controle terug te krijgen in de webbrowser. Een dergelijk verzoek wordt door client en server in de achtergrond afgehandeld. De gebruikersinterface wordt overeenkomstig het antwoord van de server automatisch aangepast. Dit interactiemodel zorgt voor een toename van de gebruikersinteractiviteit en een lagere door de gebruiker waargenomen vertraging op het web.

Deltacommunicatie

In Ajax worden alleen de toestandswijzigingen doorgegeven tussen de client en server. Deze deltawijzigingen vormen de basis waarmee kleine updates op de webinterface mogelijk worden gemaakt. Deltacommunicatie verbetert de netwerkperformance direct, omdat er minder overbodige data heen en weer hoeft te gaan. Als gevolg hiervan wordt de door de gebruikers waargenomen vertraging en de gebruikersinteractiviteit verbeterd.

Gebruikersinterfacecomponenten

SPIAR is opgebouwd op een single-page gebruikersinterface die is samengesteld uit componenten die vergelijkbaar zijn met die van desktopapplicaties, zoals AWT's[4] gebruikersinterfacecomponentenmodel. Gebruikersinterfacecomponenten verbeteren de eenvoud, omdat ontwikkelaars herbruikbare componenten kunnen creëren en gebruiken om een webpagina samen te stellen. De gebruikersinteractiviteit wordt hierdoor ook verbeterd omdat het interactieniveau daalt naar het componentenniveau, net als in desktopapplicaties.

4 De Java Abstract Window Toolkit.

Webstandaarden

Het leggen van beperkingen op webelementen zodat ze aan standaarden moeten voldoen, veroorzaakt de gewenste portabiliteiteigenschap. Deze restrictie sluit aanpakken uit waarin extra functionaliteit buiten deze standaarden ligt (zoals een virtuele machine of een plugin als Flash) en maakt de client compatibel.

architectuurelement	gebruikers-interactiviteit	vertraging	netwerk-performance	eenvoud	schaalbaarheid	portabiliteit	zichtbaarheid
•asynchrone interactie	+	+					
•delta-communicatie	+	+	+		–		–
•client-side processing	+	+	+				
•gebruikersinterface-componenten	+			+			
•webstandaarden				+		+	
•stateful benadering	+	+	+		–		–

Tabel 1. **RESTricties en geïnduceerde eigenschappen**

Client-side processing

In Ajax hoeft de server niet alleen alle taken uit te voeren, ook de client kan een actieve rol vervullen bij het verwerken van bepaalde acties, zoals sorteren van items. Hierdoor kunnen sommige taken van de server naar de client overgedragen worden. *Client-side processing* verhoogt de gebruikersinteractiviteit en verbetert de zichtbare vertraging door het verkorten van de tijd die nodig is om data over en weer te sturen. De clientperformance wordt zeker een afweging als er veel componenten (widgets) op de client gecreëerd worden. Google's GWT gaat wat betreft client-side processing tot het uiterste door het genereren van JavaScript-code voor de gehele gebruikersinterface die op de client draait.

Stateful

Een server die stateless of toestandloos is, behandelt elk clientverzoek als een onafhankelijke transactie, niet gerelateerd aan reeds gemaakte verzoeken. Omdat een toestandloze benadering op het web de netwerkperformance vermindert (door de verhoging van repetitieve data) en vanwege de componentgebaseerde manier van interactie in Ajax, is een stateful benadering gunstiger, zij het dat dit ten koste gaat van schaalbaarheid en zichtbaarheid.

Conclusie

We hebben aan de hand van onze SPIAR-architectuurstijl laten zien hoe concepten vanuit de architectuurwereld zoals architectuureigenschappen, -restricties en –elementen, ons kunnen steunen om een complexe en dynamische technologie als Ajax te begrijpen. Ons werk bouwt op de grondslagen van de REST-stijl en biedt een analyse van deze stijl met betrekking tot het bouwen van Ajax-webapplicaties. SPIAR beschrijft de software-engineeringprincipes die door ontwikkelaars gebruikt kunnen worden tijdens het bouwen en analyseren van Ajax-applicaties.

Literatuur

Fielding, R. & R. N. Taylor (2002). Principled design of the modern Web architecture. *ACM Transactions on Internet Technology*, Vol. 2, No. 2, 115–150. www.ics.uci.edu/~taylor/documents/2002-REST-TOIT.pdf.

Garrett, J. (2005). *Ajax: A new approach to web applications*. Adaptive path. adaptivepath.com/publications/essays/archives/000385.php.

Mesbah, A. & A. van Deursen (2007). An architectural style for Ajax. In: Proc. WICSA '07: 6th Working IEEE/IFIP Conference on Software Architecture.

IEEE Computer Society, 2007 (te verschijnen). arxiv.org/pdf/cs.SE/0608111.

Perry, D. E. & A. L. Wolf (1992). Foundations for the study of software architecture. ACM Sigsoft Software Engineering Notes, 17(4):40–52. www.ece.utexas.edu/~perry/work/papers/swa-sen.pdf.

Ali Mesbah
is promovendus software engineering aan de Technische Universiteit Delft. E-mail: a.mesbah@tudelft.nl.

Arie van Deursen
is hoogleraar software engineering aan de Technische Universiteit Delft en senior onderzoeker bij het Centrum voor Wiskunde en Informatica. E-mail: Arie.vanDeursen@tudelft.nl.