DENKEN OVER SOA: CHANGE YOUR MIND(SET)

Ik weet niet hoe het met jullie zit, maar alles rondom Service Oriented Architecture (SOA) boeit mij enorm. En dan heb ik het niet alleen over de technologie - hoewel die ook zeer boeiend is. Denk eens aan de SOA-principes: hoe bedenken we die, waar halen we daarvoor de inspiratie vandaan? Of denk aan de samenwerking op dit gebied tussen bedrijven die elkaar om het hardst beconcurreren: Microsoft, IBM en nu zelfs SUN die samenwerken bij het opstellen van webservices-specificaties. Denk aan de torenhoog gespannen verwachtingen van veel SOA/Web Services proponenten versus de cynische houding van de sceptici: er is niets nieuws onder de zon. In dit blad is terecht veel aandacht voor de 'harde' techniek – productbesprekingen, programmeerprincipes, previews van nieuwe technologie. Maar soms is het ook goed om wat afstand te nemen en te filosoferen over een technologie. Ik noem het 'Denken over SOA'.

Het eerste dat mij opvalt is dat de grootste uitdaging van SOA niet in de technologie zit, maar in de 'mindset' van de mensen die er mee moeten werken en dat daar mijns inziens veel te weinig aandacht aan besteed wordt. En hoe makkelijker Microsoft het gebruik van de technologie maakt – 'point and click', 'drag and drop' en niet te vergeten 'tag and prop(erty)' - hoe meer het noodzakelijk is dat het met die 'mindset' goed zit. Laat het duidelijk zijn: dat Microsoft zijn best doet om technologie minder complex te maken is een zeer goed streven. Maar de eenvoud in het hanteren van de tools kan leiden tot de gedachte dat het maken van architectonisch verantwoorde, robuuste, schaalbare enterprise applicaties geen grondig denkwerk en goed ontwerp vereist. En zonder dat kunnen software-engineers met geweldige tools de grootste rotzooi maken. Om de mindset naar een SOA-denkkader te veranderen is een paradigmawisseling nodig: een radicale verandering van je denken. Stephen Covey geeft in zijn boek "The seven habits of highly effective people" een mooi voorbeeld van een paradigmawisseling. Een oorlogsschip is met een militaire oefening bezig en vaart al dagen op zee. Het is bijzonder mistig en de kapitein van het schip besluit op de brug te blijven. Opeens ziet men een licht aan stuurboord dat recht op het schip afkomt. De kapitein seint aan het andere schip: "aanvaring dreigt, verander uw koers 20 graden." Het andere schip seint terug: "advies aan u: verander uw koers 20 graden." De kapitein raakt geïrriteerd en seint: "ik ben kapitein: verander uw koers 20 graden." Het antwoord: "ik ben stuurman tweede klas, verander uw koers 20 graden." Nu is de kapitein razend en seint: "Dit is een oorlogsschip: verander uw koers onmiddellijk!" Daarop kwam het antwoord: "Dit is een vuurtoren." En de kapitein veranderde zijn koers.

De geschiedenis herhaalt zich hier natuurlijk, want hoe vaak hebben we dit niet al eerder gezien in de ICT? De ouderen onder ons hebben de verschuiving van proceduregeoriënteerd denken naar objectgeoriënteerd en componentgebaseerd denken in het ontwerpen en programmeren meegemaakt. De nog ouderen onder ons hebben ook de verschuiving meegemaakt van 'platte bestanden', hiërarchisch of netwerkgebaseerd denken over data naar relationeel.

En let wel, ik weet uit eigen ervaring: het valt ook niet mee. Iets loslaten dat je gewend bent - 'free your mind' - en je iets nieuws laten aanleunen. Uiteindelijk lukt dat bij de meesten wel, maar het kost tijd. En sommige mensen leren het nooit, of in elk geval te laat, met soms desastreuze gevolgen. Een praktijkvoorbeeld.
Ongeveer 10 jaar geleden werd mij als zelfstandig consultant gevraagd eens een grondige analyse van een omvangrijk informatiesysteem te doen. Er waren enorme performanceproblemen. Het systeem draaide bij een centrale overheidsinstantie en was gebouwd door een zeer grote en gerenommeerde systeemintegrator. Het ontwerp, de bouw en implementatie had jaren in beslag genomen en vele miljoenen gekost. Het systeem was gebouwd met een 4GL met daarachter een rdbms: een klassieke two-tier client/server. In het begin liep het systeem vlekkeloos, maar met het vullen van het rdbms kwamen de problemen. Het werd trager en trager. Wat kon de oorzaak zijn?

Ik kreeg dikke pakken papier met de broncode van de 4GL en een complete beschrijving van het databaseontwerp inclusief de stored procedures. Ik voelde me als een crime scene investigator avant la lettre – Gruesome Grissom - en begon het bewijsmateriaal grondig te onderzoeken. Uiteindelijk kwam het lek boven en bleek de oorzaak van alle ellende in de architectuur te liggen. Alleen een volledig herontwerp en herbouw zou het probleem verhelpen. Hoewel gebruik werd gemaakt van een 4GL/rdbms toolset was de architectuur van het systeem opgezet alsof gebruik gemaakt werd van een 3GL met 'platte bestanden'. Een voorbeeld: als in een scherm een set van gegevens werd ingelezen, dan werden die gegevens preventief gelockt in de rdbms; aan het eind van het scherm werden alle gegevens weer teruggeschreven naar het rdbms, ook als er maar een attribuutje werd gewijzigd. Er werd geen gebruik gemaakt van de standaard concurrency & caching-faciliteiten van een rdbms. Zo werden veel andere basale ontwerpfouten gemaakt. De verkeerde 'mindset' van de verantwoordelijke architect was hier uiteindelijk de oorzaak. Hij zag de 4GL/rdbms toolset als een 'black box' en paste zijn vertrouwde op 3GL gebaseerde principes toe, niet wetende dat een andere technologie een compleet andere aanpak vereist.

Ik ben bang dat als de applicatiearchitect en applicatieontwikkelaar van het aankomende SOA-tijdperk zijn/haar denkkader niet verandert en de oude, bestaande principes blijft toepassen, we met soortgelijke rampen geconfronteerd gaan worden. Mijn advies is dan ook: spendeer tijd om SOA en de principes daarachter te doorgronden. Probeer het te vatten en laat je niet in de luren leggen door de eenvoud van de toolset: change your mindset!

Dik Bijl
Enterprise architect
Enterprise & Partner Group