Service model en use case model : een lastige combinatie

Hans Admiraal
mei 2008

In een moderne automatiseringsomgeving zijn zowel serviceoriëntatie als use-caseoriëntatie sterk verankerd. Maar passen die twee wel bij elkaar? Er is geen één-op-éénrelatie tussen services en use cases. In het eerste deel van dit artikel schetsen we welke rol use cases spelen bij het ontwerpen van services en vice versa. In het tweede deel gaan we in op de vraag wat de eenheid van planning het beste kan zijn: een service of een use case. Ten slotte zullen we de conclusies samenvatten.

1. ONTWERPEN

Het service model

Een professionele informatievoorziening bestaat niet alleen uit de benodigde hardware en software, maar omvat ook de nodige documentatie. Voor de aanpak van een IT-project is met name het ontwerp relevant: hoe komen we van het ontwerp van de bestaande situatie naar een ontwerp van de nieuwe situatie?

Laten we eerst eens kijken naar de opbouw van het ontwerp van de bestaande situatie. Het mag duidelijk zijn, dat dit niet het product is van één project, maar door de tijd ontstaan is door alle projecten die het IT-landschap gevormd hebben. Het ontwerp staat onder de centrale verantwoordelijkheid van het architectuurteam; niettemin bestaat het uit verschillende deelproducten met elk zijn eigen redactie. Meestal zijn er twee dimensies te onderkennen: het soort ontwerp (bijvoorbeeld een functioneel ontwerp en een technisch ontwerp) en het domein (bijvoorbeeld een ontwerp voor een bepaald bedrijfsonderdeel of een bepaald aandachtsgebied).

Bij het toepassen van een servicegeoriënteerde architectuur (SOA) wordt een nieuw soort ontwerp onderkend, dat verschillende benamingen kent, maar dat we hier het "service model" noemen, in navolging van de methode RUP. Het service model is het geheel aan informatie over alle services in de organisatie, zowel de grote lijnen als de details. Het service model kan worden opgedeeld in afzonderlijke modellen voor de diverse domeinen.

Use cases in een ander licht

Sinds de uitvinding van de use case door Ivar Jacobson in 1986, is het uitwerken van functionele eisen in use cases steeds meer in zwang geraakt. We kunnen dit zien als een soort ontwerp. Naast het service model hebben we dan een zogenaamd use case model. In plaats van losse systeemfuncties op te sommen, wordt in een use case het hele scenario geschetst van hoe een gebruiker een bepaalde taak uitvoert, een bepaald doel bereikt. In het traditionele (niet-servicegeoriënteerde) applicatieparadigma wordt het te bouwen systeem dan als black box beschouwd en worden de gebruikers en de te koppelen systemen benoemd als "actoren".

Wat gebeurt er nu, als we "systeem" vervangen door "service"? Een use case heeft dan betrekking op één service en alle andere services zijn actoren. Er ontstaat een probleem, wanneer een gebruiker in het kader van een bepaalde taak of bepaald doel niet met één, maar met meerdere services interacteert. Bijvoorbeeld bij het opnemen van een bestelling, zal de verkoper de CRM-service gebruiken om klantgegevens in te voeren, de catalogus-service gebruiken om product- en prijsgegevens te vinden en de order-service om de bestelling op te nemen en definitief te maken. Het opsplitsen van de use case in drie use cases zou het voordeel van de use-casetechniek, dat deze het hele scenario beschrijft van hoe de gebruiker de taak uitvoert, teniet doen. Bovendien is het bij het maken van de use case nog niet altijd duidelijk welke functionaliteit bij welke service belegd zal worden. Dit is voor de materiedeskundige, die de use case moet kunnen valideren en goedkeuren, ook niet relevant.

In principe kan men alle informatietechnologie samen als "het systeem" beschouwen en dus alle individuele services binnen de organisatie niet als actoren benoemen. De mens heeft een dialoog met het beeldscherm en de rest is een black box. Wat we los moeten laten, is dat een use case naast een functionele scope ook nog een technische scope heeft, namelijk "de te bouwen applicatie". De technische scope wordt bepaald door de architect, die aan de hand van de use case gaat bezien welke services er veranderd of bijgebouwd zullen worden. Wat wel een nuttige systeemafbakening is, is de organisatie zelf, met andere woorden: als er informatie uitgewisseld wordt met externe partijen, dan kunnen (de systemen van) deze externe partijen als actoren opgevoerd worden in de use cases. Het is namelijk ook voor de materiedeskundigen belangrijk om het onderscheid te definiëren tussen wat "wij" doen en wat "zij" doen.

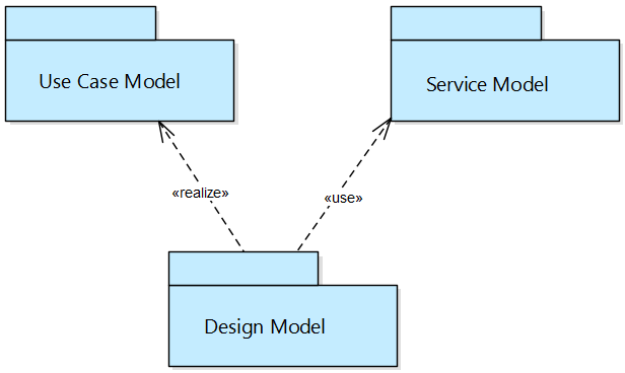
Relaties tussen use cases en services

Zowel het use case model als het service model beschrijven de informatievoorziening, de eerste in termen van gebruiksscenario's en de tweede in termen van beschikbare services. Hoe verhouden de twee genoemde modellen zich nu tot elkaar? Er is zeker een verband, maar het is allereerst belangrijk om de scheiding tussen beide te benadrukken. Een service is met name krachtig, als deze generiek genoeg is om een rol te spelen in allerlei use cases. Een CRM-service bijvoorbeeld, speelt een rol in alle use cases die te maken hebben met klantrelaties: van acquisitie en

orderverwerking tot aan klachtenafhandeling. Generieke services zullen dus niet gerelateerd worden aan één of meer specifieke use cases, juist om de herbruikbaarheid te waarborgen.

Echter, niet alle benodigde functionaliteit is te vertalen in het gebruik van generieke services. Er zijn ook services, die bedoeld zijn voor het ondersteunen van een specifieke use case. De use case "Maak een offerte" kan bijvoorbeeld worden ondersteund door een offerte-service. Zulke services maken vaak weer gebruik van services die wel generiek zijn. De offerte-service maakt bijvoorbeeld gebruik van de productcatalogus-service. Services die specifiek zijn voor een bepaalde use case zullen in het service model verwijzen naar de betreffende use case in het use case model.

Andersom staan er in het use case model geen verwijzingen naar services. De wijze waarop een use case gerealiseerd wordt in termen van services staat in een ander soort ontwerp, het design model (of hoe het ook genoemd wordt). Hierin wordt voor elk scenario aangegeven welke operaties van welke services achtereenvolgens worden aangeroepen. De relaties tussen de use cases en de services worden dus in het design model volledig in kaart gebracht.



Business process model

Naast het use case model, het service model en het design model kunnen er nog andere soorten ontwerp een rol spelen. Dit is afhankelijk van de methode en de architectuur. Een SOA vergemakkelijkt de inzet van een business process engine, waarvoor een business process model gemaakt zal moeten worden. Vanuit een dergelijk model kunnen zowel use cases als services worden afgeleid.

Het ontwerpproces

Terug naar het begin. Voor een nieuwe functionele behoefte wordt een project opgestart. De functionele eisen worden in volgorde van prioriteit uitgewerkt in use cases. Dit kunnen nieuwe use cases zijn, maar ook wijzigingen op bestaande use cases. Vervolgens wordt getracht deze wijzigingen in het design model te verwerken, zodanig dat elke use case wordt gerealiseerd door het aanroepen van bestaande services, die in het service model gedocumenteerd zijn. Voor zover dit niet lukt, worden nieuwe services of nieuwe operaties op bestaande services ontworpen.

Een goed versiebeheer op de modellen is hierbij onontbeerlijk. Het moet duidelijk zijn welke versie van welke modellen de status quo ("IST") vertegenwoordigen en welke versies de toekomstige situatie ("SOLL") weergeven. Het verschil tussen de IST en de SOLL versies is datgene wat in het kader van het project ontwikkeld moet worden. De ontwikkelaars ontvangen taken, waarin beschreven staat welke delta in de modellen in het kader van de taak ontwikkeld dient te worden. Na afloop worden de SOLL-versies van de modellen gepromoveerd tot IST-modellen.

Overigens verloopt het ontwerpproces niet zo lineair als hierboven geschetst. Meestal gaat de voorkeur uit naar een agile aanpak, waarbij de diverse ontwerp-, programmeer- en testactiviteiten min of meer gelijktijdig plaatsvinden.

2. PLANNING EN RISICOBEHEERSING

Omdat services relatief onafhankelijk van elkaar ontwikkeld kunnen worden, is het verleidelijk om services als eenheid van planning te gebruiken. Specificatie-, bouw- en testactiviteiten rond een service vormen immers een hechte eenheid. Services die elkaar niet aanroepen, kunnen parallel ontwikkeld worden, zonder onderlinge afstemming. Als die services volledig gebouwd en getest zijn, kunnen vervolgens andere services, die daar gebruik van maken, worden ontwikkeld en tenslotte wordt het geheel samengesmeed, bijvoorbeeld door middel van orchestration software en een portal (user interface).

Deze benadering heeft als nadeel, dat twee belangrijke projectrisico's te laat worden bestreden. Het eerste is, dat er technische problemen zijn in de aansluiting van de diverse componenten op elkaar. Deze komen vaak pas aan het licht als er daadwerkelijk een user interface gebouwd is en het samenspel van alle onderdelen zich waar moet maken. Het

tweede risico is, dat gebruikers en andere belanghebbenden bij het zien van het eindresultaat tot het inzicht komen dat het systeem in bepaalde opzichten niet voldoet. Aangezien dit eindresultaat lang op zich laat wachten, komt de feedback in een stadium waarin de meeste services al gebouwd en getest zijn. Het is dan erg kostbaar om terug te gaan naar de tekentafel.

Een derde nadeel van service-driven planning is, dat een service vaak ten dienste staat van functionaliteit met uiteenlopende businessprioriteiten en het ontwikkelen van een service in dat geval niet alleen belangrijke, maar ook minder belangrijke software oplevert, die in de loop van het project misschien buiten de scope geplaatst wordt.

Het is dan ook beter om niet de service, maar de use case als eenheid van planning te hanteren. Een use case implementeert een complete gebruikerstaak, terwijl een losse service slechts een deeloplossing is. Het opleveren van een use case maakt de voortgang zichtbaar aan alle betrokkenen en geeft meer houvast om feedback te krijgen. Door het opleveren van complete functionele delen worden zowel functionele als technische risico's vroegtijdig zichtbaar, in een stadium waarin wijzigingen nog relatief goedkoop kunnen worden doorgevoerd. Hierbij worden use cases, die vanuit de business gezien het meest urgent of waardevol zijn, of die het grootste projectrisico vormen (bijvoorbeeld door de technische complexiteit ervan), het eerst aangepakt.

Is dit nu in alle gevallen de meest optimale strategie? Nee. Niet in het zeldzame geval dat er geen technische problemen optreden en dat alle gewenste functionaliteit ook daadwerkelijk wordt opgeleverd en zelfs in één keer goed. De use-case driven planning kost in dat geval meer tijd en dus geld en is moeilijker te managen dan een service-driven planning. Misschien hebben vele use cases impact op dezelfde services en moeten deze services telkens weer "overhoop", totdat de laatste use case af is. De services blijven lang instabiel en daardoor ontstaan fouten in reeds opgeleverde functionaliteit.

Dit effect noopt tot twee maatregelen.

1. Probeer bij het uitbreiden of nieuw ontwikkelen van een service meteen te anticiperen op later geplande use cases die impact hebben op diezelfde service. Mocht duidelijk zijn, welke behoeften bepaalde toekomstige use cases hebben ten aanzien van onder handen zijnde services en die behoeften zijn zonder veel vertraging mee te nemen, stel dit dan niet uit tot de betreffende use cases aan de beurt zijn, maar doe het meteen.

2. Als kan worden ingeschat, dat de technische en functionele risico's voldoende zijn afgedekt, wijzig dan de planningsstrategie van use-case driven naar service driven. Dit zou met name in de latere fasen van het project het geval kunnen zijn.

3. CONCLUSIE

Bij het bedrijven van automatisering is het raadzaam om zowel serviceoriëntatie en use-caseoriëntatie toe te passen. Dit artikel heeft enige houvast proberen te bieden bij het combineren van beide. Hierbij hebben we uitgelegd wat het betekent om een service model te onderhouden en verdedigden wij de stelling dat use cases service-overstijgend zijn en onafhankelijk van de servicegrenzen gespecificeerd dienen te worden. Qua planning adviseren wij in principe een use-case driven benadering, waarbij onder voorwaarden ook aspecten van een service-driven aanpak meegenomen dienen te worden.