Centrale uniforme manier van loggen is fundamenteel

Succesvol monitoren in een complexe SOA-omgeving

Vincent Jansen, Gebrian uit de Bulten en Michael Widjaja

Een Service Oriented Architecture orkestreert bedrijfsprocessen door functionele services en functies met elkaar te koppelen. De succesvolle regie in het monitoren en analyseren van fouten in een gehele SOA-keten vergt echter wel de nodige aandacht voor een fundamentele oplossing. Deze oplossing ondersteunt het analyseren van foutmeldingen over verschillende services heen, maar helpt ook bij het monitoren van functionele gebeurtenissen. Een centrale database is het fundament van de oplossing.

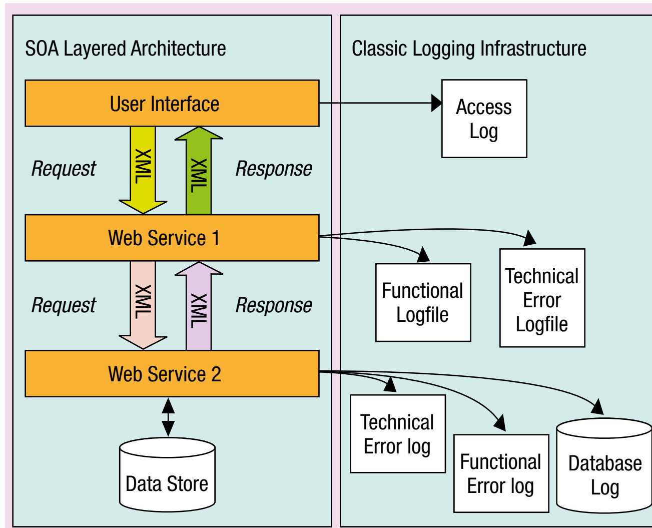
Waar SOA een aantal jaren terug nog in de kinderschoenen stond op het gebied van software architecturen, is het nu binnen de meeste ondernemingen al toegepast. Gartner's 2008 SOA Gebruikers Onderzoek [1] bevestigt dat ook met inzichten in het gebruik van SOA binnen ondernemingen met 1000 of meer werknemers. Het onderzoek laat echter ook zien dat er een afname verwacht wordt van verdere groei van SOA.
Belangrijkste struikelblokken zijn gebrek aan ervaring of kennis, onvoldoende onderbouwing met een business case, en de complexiteit bij het toepassen ervan.
Bedrijven hebben te maken met vraagstukken over het effectief toepassen binnen de organisatie, en hoe informatie effectief te managen over de keten van applicaties en services [2].
Voor het succesvol toepassen van een SOA is er behoefte aan *governance*. Het ontbreken ervan kan leiden tot redundantie, services die niet op elkaar aansluiten en het ontbreken van richtlijnen op het gebied van beheer en monitoring, met als uiteindelijk gevolg operationele problemen of een verminderde kwaliteit van de dienstverlening. Dit artikel gaat in op het inrichten van een uniforme manier van monitoren over de keten van services in een SOA-landschap. Een praktijkvoorbeeld toont op concrete wijze de toepassing.

Probleemdefinitie

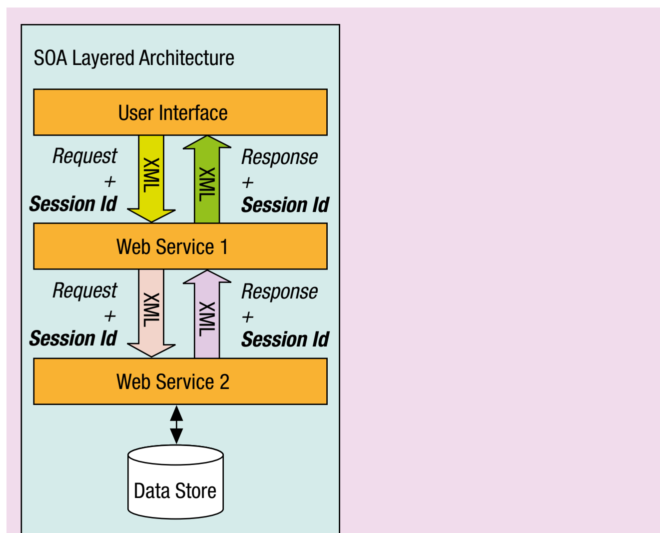
Eén van de belangrijkste struikelblokken is hoe om te gaan met analyse van problemen en specifieke functionaliteit over de verschillende services heen binnen een SOA. In een service georiënteerde architectuur levert een keten van services de gewenste functionaliteit naar de eindgebruiker. Daarbij is elke service in de keten een entiteit op zichzelf, met eigen logging-functionaliteit over functionele of technische fouten. Hoewel de gegevens binnen een enkele service te achterhalen zijn, is het zeer lastig om een volledig beeld te krijgen van wat er precies speelt als het om meerdere services tegelijkertijd gaat. Bovendien is het vaak moeilijk of onmogelijk berichten met elkaar te koppelen. Enige correlatie tussen de meldingen van de individuele services ontbreekt, waardoor oorzaak en gevolg niet achterhaald kunnen worden.
Een karakteristieke SOA bestaat uit een User Interface, met daaronder meerdere webservices, zie afbeelding 1. Hierbij verzorgt iedere applicatielaag zijn eigen logging met eigen formaat logberichten, detailniveau en verschillen qua inhoud en manier van opslaan.

Oplossing

Voor de oplossing is het noodzakelijk om structuur aan te brengen in de berichtenstroom van de verschillende individuele services om beter te achterhalen wat zich binnen een keten



Afbeelding 1: Karakteristieke SOA.



Afbeelding 2: Gebruik unieke Session Identifier.

afspeelt. Een systeem is kenmerkend gebruikergeoriënteerd, batchgeoriënteerd, of een combinatie van beide. Bij het achterhalen van wat zich binnen een systeem afspeelt is het wenselijk om de geregistreerde gebeurtenissen te kunnen koppelen aan een gebruikerssessie, of een bepaalde *batch run*. Dit kan gedaan worden door de sessie of batch run een uniek identificatienummer, of *Session Identifier* te geven.

Voor het definiëren van de oplossingsrichting gaan we uit van een aantal principes:

1. Berichten worden op een centrale locatie verzameld;
2. Berichten worden op een uniforme wijze vastgelegd. Daarbij,
   a. heeft ieder bericht dezelfde structuur (maar kunnen andere details worden vastgelegd);
   b. bevat ieder bericht bovengenoemde Session Identifier die het bericht koppelt aan de gebruikerssessie;
3. Verzamelde berichten worden periodiek gecorreleerd, en verwerkt tot voor de business nuttige informatie.

Hoewel er veel verschillende implementaties mogelijk zijn, is in dit voorbeeld gekozen voor een oplossing met een message queue in combinatie met een database, die als volgt werkt.

Fase 1: initiëren van de Session Identifier. Het belangrijkste is de Identifier te creëren die de gehele sessie uniek identificeert. Deze Id wordt vervolgens met elke serviceaanroep weer meegegeven. In een webservice call wordt deze unieke identifier mee gegeven in de *soapHeader*, om vervolgens deze identifier in een *local thread* bij te houden (zie afbeelding 2). Mogelijk is zo'n identifier al aanwezig binnen de SOA-keten, en hoeft deze niet apart te worden geïntroduceerd.

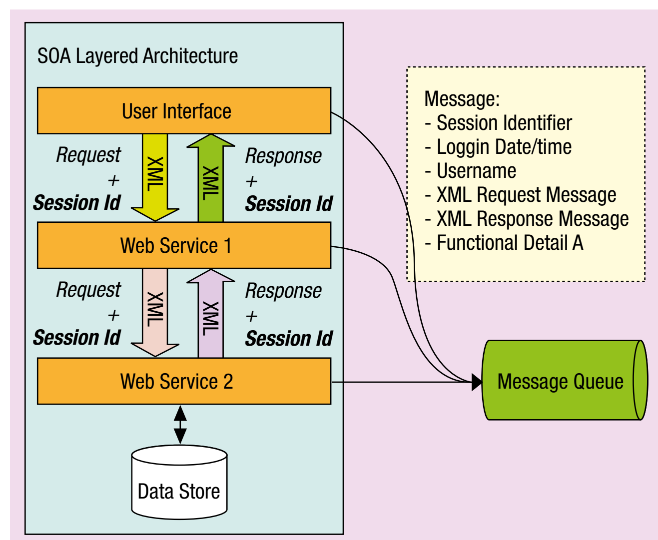
Fase 2: loggen van berichten op uniforme wijze. Nu kan op elk gewenst moment in de service een bericht in de queue worden gezet met deze unieke identifier. Dit kunnen dus de XML berichten zijn die binnenkomen op de webservices, maar ook errors die gecreëerd worden, of op specifieke onderdelen die men altijd gelogd wil hebben. Het belangrijkste is dat al deze berichten op een uniforme wijze in de message queue worden gezet. Dit kan door de webservice applicatie zelf worden verzorgd, maar mogelijk ook vanuit een ESB, zie afbeelding 3.

Fase 3: verzamelen en verwerken van gegevens. Vervolgens worden deze berichten door een applicatie uit de queue gehaald en in een centrale database gezet. Daarmee staan dus alle gegevens in een centrale database. Tot slot rest er niets anders dan met behulp van query's op de database deze gegevens te verzamelen en in een andere database gestructureerd neer te zetten. Een beproefd mechanisme is om hierbij gebruik te maken van (horizontale) partitionering, en een round robin mechanisme dat de individuele partities verwerkt. Vervolgens kan deze informatie met een speciale rapporteringsapplicatie geanalyseerd worden.

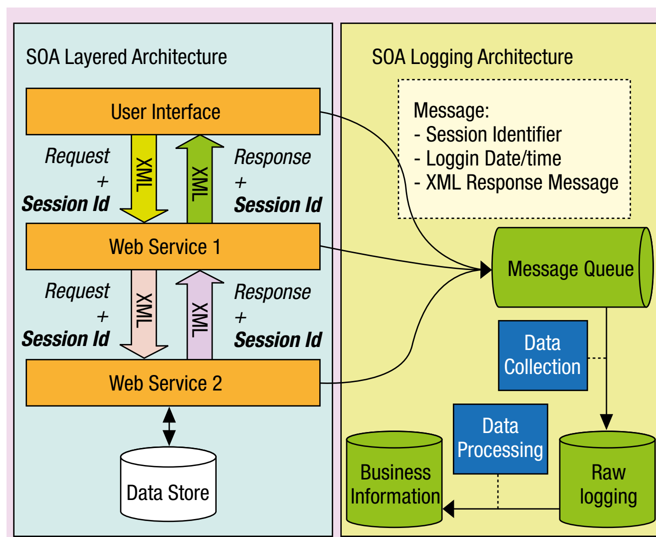
Ontwerpconsideraties

In de ontwerpfase dient een aantal cruciale beslissingen te worden genomen:

- Berichtenstructuur. Het is van belang van tevoren vast te stellen welke informatie moet worden vastgelegd. Zowel de applicaties als de databases dienen hierop te worden ingericht;
- Applicaties hebben vaak te maken met zogenaamde Global Transactions (ook wel 2 phase commit, of XA-transacties). Het is van belang in de ontwerpfase in kaart te brengen welke applicatietransacties hiervan gebruikmaken. Per transactie zou moeten worden gekeken of het wenselijk is dat het plaatsen van de loggingberichten onderdeel moet zijn van deze Global Transaction. Indien niet noodzakelijk is het aan te raden om deze logging *geen* onderdeel uit te laten maken van de transactie. Daarmee voorkom je dat het succesvol afronden van een applicatietransactie afhankelijk wordt van het beschikbaar zijn van de logging message queue;
- Bij het registreren van meerdere berichten per transactie kan de totale hoeveelheid te verwerken data sterk groeien. Hier dient binnen het ontwerp van de hardware infrastructuur al rekening mee te worden gehouden met een schaalbare oplossing. Denk hierbij aan het kunnen inzetten van meerdere



Afbeelding 3: De message queue.



Afbeelding 4: Verzamelen en verwerken van gegevens.

message queues, en aparte databases voor het verzamelen van de ruwe logdata en loginformatie.

Deze richtlijnen worden karakteristiek vanuit een IT/SOA governance orgaan opgesteld en gereguleerd.

Binnen dit mechanisme speelt de database dus een cruciale rol. Enerzijds dient deze als centraal verzamelpunt voor alle berichten die in de message queue zijn geplaatst. Daarnaast voorziet de database ook in het mechanisme om deze berichten te verwerken tot informatie die nuttig is voor de business. Tot slot maakt de inzet van de database het ook mogelijk om de oplossing op te schalen bij hoge datavolumes. Daarbij dient eveneens rekening te worden gehouden met de message queues en batchprocessen voor datacollectie en processing.

Voorbeeld

Een voorbeeld gaat verder in op het inzetten van de database voor deze functionaliteit. We nemen als business case een simpele online banking applicatie bij de fictieve bank BankRealSafeOnline. Deze bestaat uit een toplaag waaruit de UI bestaat. Deze roept vervolgens twee business services aan:

- Eén van deze services is de verwerking van de betaling bij een andere externe partij (webservice), genaamd *ExternalPaymentServices*;
- De andere service zorgt ervoor dat de interne verwerking plaatsvindt op het mainframesysteem (ook bereikbaar via een webservice), genaamd *InternalPaymentService*.

Het probleem dat zich hier voordoet is dat de verwerking van een betaling bij twee onderdelen moet worden belegd. Nu gaan sommige van deze betalingen mis, echter dit komt pas aan het licht wanneer blijkt dat de logginggegevens van business service 1 en 2 elkaar tegenspreken.

Vanwege de *loosely coupled nature* van verschillende services is het vaak onmogelijk om de diverse stappen aan elkaar te koppelen en dus de daadwerkelijke errors te zoeken.

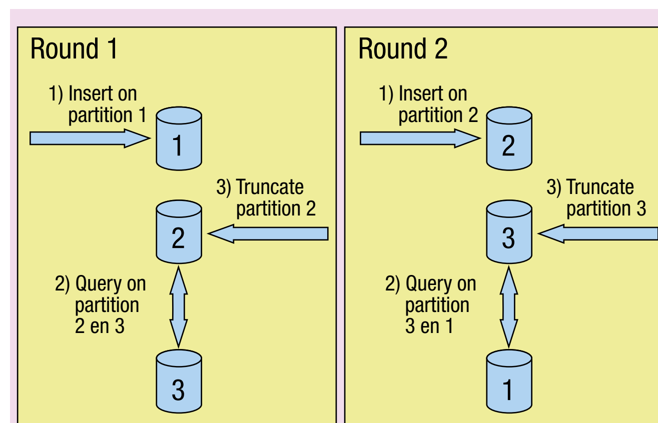
Binnen het voorbeeld speelt zich het volgende scenario af:

1. UI (klant) geeft aan 100 euro over te willen maken naar een andere bank;
2. ExternalPaymentServices (service die external payment service aanroept) doet een overdracht van 100 euro naar de externe, 3rd party payment service;
3. De externe payment service geeft aan dat de betaling mis is gegaan omdat de adresgegevens ontbreken (00000001 error);
4. De klant krijgt een nette melding dat zijn adresgegevens ontbreken;
5. De klant probeert het opnieuw, nu met de adresgegevens correct ingevuld;
6. ExternalPaymentServices doet opnieuw een poging tot het uitvoeren van de transactie van 100 euro en ditmaal komt het bericht terug dat alles in orde is;
7. Vervolgens wordt InternalPaymentServices (service die internal verwerkingsservice aanroept) aangeroepen om het eigen banksysteem te updaten. Door een bug in het systeem geeft deze service echter de status van de eerste aanroep door (dat de betaling is misgegaan);
8. De InternalPaymentServices doet een validatiecheck om te zien of de gegevens kloppen. Echter deze geeft een 00000002 error dat deze betaling wel is doorgezet.

Het is duidelijk vanuit de beschrijving dat in stap 7 van het betalingsproces iets grondig verkeerd gaat. Bij het nazoeken in de logging is dit echter niet snel te achterhalen en bij de foutafhandeling in dit scenario doen zich verschillende complicaties voor:

- InternalPaymentServices ziet nooit het oorspronkelijke probleem, omdat deze bij de aanroep al met de verkeerde data is voorzien.
- Bij het nazoeken binnen de ExternalPaymentServices logging is wel een probleem te zien, maar dit lijkt een normale, functionele gebruikersfout te zijn.
- Beide problemen zijn gerelateerd aan elkaar, maar laten niet de werkelijke oorzaak van de problemen zien. Zonder het complete scenario te registreren is het vrijwel onmogelijk te achterhalen wat er fout gaat.

Bij het registreren van alle SOA-berichten worden dus alle web-



Afbeelding 5: Round robin procedure.

service-berichten in een logfile en in een centrale queue gezet, met daarbij ook de *timestamps* en ook de volgorde van deze berichten. Voor dit voorbeeld worden de volgende onderdelen in de queue gezet: HTML, applicatie-errors en XML berichten die naar de webservices worden gestuurd. De berichtenstructuur die in de queue wordt gezet is volgens een standaard XML patroon. Sommige waardes zullen optioneel zijn, zoals errors, andere zullen verplicht zijn; bijvoorbeeld transactionID. Voor dit voorbeeld zullen we onderstaande structuur pakken.

```
<Message>
<transactionID>1</transactionID>
<messageNummer>3</messageNummer >
<serviceName>externalPayment</serviceName>
<messageBody>
<![CDATA[<soap:Envelope xmlns:soap=
                    "http://schemas.xmlsoap.org/soap/envelope/">
<soap:Header>
<bol:TransactionData xmlns:bol="http://www.accenture.com/
                    correlation" transactionId="transactionID">
</bol:TransactionData>
</soap:Header>
<soap:Body>
<ns1:DoPayResponse xmlns:ns1="http://accenture.com/banking/
                                              dopay/response">
<Errors>
<Error description="missing address" code="00000001"/>
</Errors>
</ns1:DoPayResponse>
</soap:Body>
</soap:Envelope>]]>
</messageBody>
<messageDate>2010-12-29T13:33:23</messageDate>
<messageName>DoPayResponse</messageName>
<messageType>XML</messageType>
<errorCode>00000001</errorCode>
```

In dit voorbeeld krijgen we van de externalPayment service een error terug, alsmede de internal service. Deze wordt in bovenstaand format in de queue gezet. Dit bericht wordt samen met alle andere berichten die bij deze transaction horen onder sessionID1 in de queue gezet. Dus alle XML-, service exceptions- en HTML-berichten van deze gebruiker worden met sessionID1 in de queue gezet.
Vervolgens worden door een andere applicatie die in deze queue luistert al deze gegevens in de database gezet. In een highload-omgeving wordt dit door middel van batch inserts gedaan.
Op dit moment zijn alle gegevens op één centrale plaats opgeslagen. Vervolgens draait er een apart proces dat alle berichten die bij elkaar horen verzamelt en deze gegevens op een gestructureerde manier in een andere database zet. Vervolgens worden de berichten die verwerkt zijn verwijderd uit de database.
Met grote transactievolumes moet voorkomen worden dat een proces iets weghaalt terwijl het andere proces het nog moet verwerken. Voor deze laatste stappen wordt het round robin mechanisme ingezet samen met het gebruik van partitionering. Alle berichten worden op partitie 1 gezet. Vervolgens worden op partitie 2 en 3 alle gegevens verzameld (dit gebeurt over twee partities om er zeker van te zijn om geen gegevens te verliezen die net een seconde later gebeuren na de switch van partities, ook worden hier dubbele berichten eventueel gefilterd) en worden deze gegevens in een andere database gezet. Als laatste wordt partitie 2 afgekapt.
De volgende run is de tweede stap van de round robin en worden alle berichten op partitie 2 gezet en worden op partitie 3 en 1 alle gegevens opgehaald en wordt partitie 3 weer geschoond. De volgende stap gaat zo verder en het begint daarna weer opnieuw. Men kan de round robin strategy zo aanpassen dat het een gedeelte van de gegevens snel verwerkt (omdat men er zeker van is dat het laatste bericht binnen is) of dat men nog wacht op gegevens die bij een bepaalde transactie horen.
In de gestructureerde database kan een front-end applicatie worden ingezet om te zoeken naar specifieke errors. In de bankapplicatie wordt gezocht op error code 00000002, de gebruiker ziet vervolgens alle berichten en zal ook de 00000001 error zien. Met daarbij de volgorde van de berichten en hij zal zien dat de respons van de eerste error wordt doorgestuurd naar de service en daardoor ontstaat uiteindelijk error 00000002.
Dit praktijkvoorbeeld lost een vaak voorkomend complex probleem op waarin defects van 60 procent ‘unknown’ problemen terug te brengen is naar 5 procent; variërend van *bots* tot *endless loops* tot aan klanten waar de naam op een SQL statement lijkt. Deze implementatie kan opgezet worden voor zeer grote tot zeer kleine applicaties. Ook wordt er vaak nog additionele logging toegevoegd, denk aan IP, clone naam enzovoort.

Conclusie

Met de forse groei van SOA in de afgelopen jaren en het aan elkaar koppelen van steeds meer systemen en services is ook de behoefte gegroeid naar sterke governance en monitoring. Een centrale uniforme manier van loggen is fundamenteel, waarbij de gehele stroom van berichten in de keten wordt vastgelegd, en maakt het mogelijk door te groeien bij hogere transactievolumes. De beschreven oplossing stelt bedrijven in staat om niet alleen meer te reageren, maar ook proactief te analyseren wat er binnen de systemen gaande is.

Literatuur

1. *Sholler, D. (26 September 2008). 2008 SOA User Survey: Adoption Trends and Characteristics. Gartner, Artikel ID Number: G00161125.*
2. *Bradley, A. (14 March 2008). Key Issues in Application Architecture, 2008. Gartner, Artikel ID Number: G00156063.*

Vincent Jansen is consultant, **Gebrian uit de Bulten** is senior system analyst en **Michael Widjaja** is Partner en verantwoordelijk voor de Nederlandse technologie Architectuur praktijk, allen werkzaam bij Accenture.