Till den som sitter med klistret

Byggklossansvaret kan vara en otacksam roll men här kommer råd

2009-05-07: Sven-Håkan Olsson

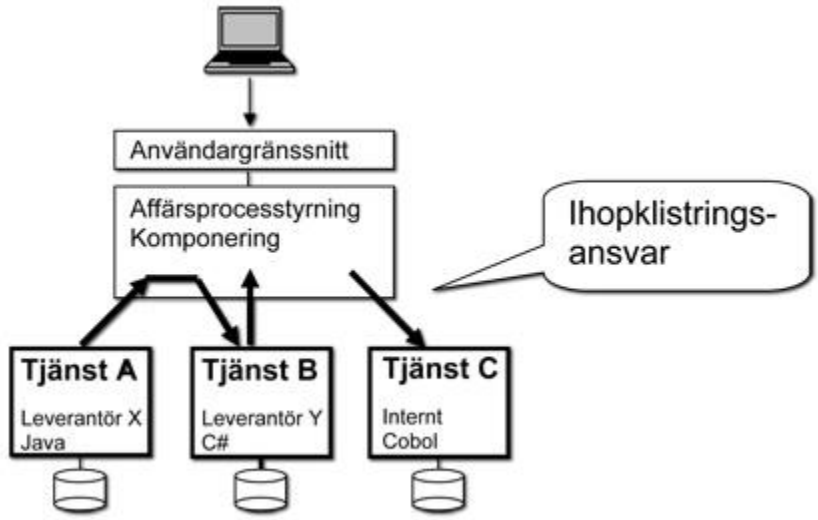
IHOPKLISTRING PÅ HÖG NIVÅ Potentialen med SOA (eventuellt kombinerat med cloud computing) är att man kan plocka ihop en lösning utifrån färdiga byggklossar. Men nu spirar en insikt i branschen att upplägget kräver en person som ansvarar för att byggklossarna blir rätt ihopklistrade. Här kommer några mycket konkreta råd för att bättre kunna felsöka och fördela felansvar.

Återanvändning av olika sorters mjukvarukomponenter har skett ända sedan it-industrins barndom. Det har rört sig om tidiga assemblermacros, gemensamma subrutiner, köpta komponentbibliotek, OO-objekt, tjänster enligt SOA-tanken och nu senast - tjänster driftade i datormoln (cloud computing).

Fördelarna är förstås stora. Det skulle kosta mycket pengar att nyskriva programkoden för en viss funktionalitet om och om igen. Samma affärslogik skulle bli programmerad på många ställen och det innebär både kvalitetsrisker och kostnader om affärsbehoven ändras. Dessutom är det ju en fördel att tidigare använd kod redan är avtestad och beprövad. Men den fördel som kanske väger allra tyngst är flexibilitet - att snabbt kunna plocka ihop en affärslösning när behoven ändrats och att kunna byta ut byggblock som inte längre räcker till.

Men tiderna har ändrats. När vi återanvände för 20 år sedan var det ofta inom samma organisation och inte sällan inom samma tekniska programmiljö. Därmed var ihopklistringen lättare. Idag har vi helt andra krav. Tjänsterna som ska klistras ihop kan

komma från helt olika leverantörer. De kan driftas internt, i något moln, i traditionell outsourcing eller hos dina affärspartners.



SOA har en stark betoning på interoperabilitet, ofta med hjälp av Web Services. Tidigare var det ordentligt svårt, till och med för ett så modernt språk som exempelvis Java, att anropa en C-rutin. Nu förväntar vi oss att tjänster från olika programmiljöer lätt ska kunna klistras ihop.

Detta är en fantastisk potential och det finns många riktigt positiva erfarenheter. Men vi måste vara medvetna om att den som ska använda ett antal tjänster från olika leverantörer, skapade av olika team, från olika verksamhetsdomäner, i olika teknikmiljöer, blir sittande med ihopklistringsansvaret.

Om helheten plötsligt upphör att fungera är det alltså den som skapat komponeringen ”som sitter i klistret”, i flera bemärkelser. De olika tjänsteleverantörerna tycker att de alla uppfyller sina tjänstekontrakt och skyller på andra. Bevisbördan landar hos ihopklistraren.

Några saker följer av detta:

1. Man måste alltid ha resurser, kunnig personal med budget och tid, beredda att rycka in för felsökning. I de lägen som verksamheten står och faller med den sammansatta lösningen kan det vara riktigt bråttom.
2. Man måste ha bra ordning på tjänstekontrakten så man vet vad som verkligen är tänkt att gälla. Observera att kontrakten inte bara utgörs av wsdl-filer, xsd-scheman eller liknande, det måste också finnas semantikförklaringar, Service Level Agreements, juridiska kontrakt, priskontrakt och säkerhetskontrakt.

3. Man måste ha tekniska verktyg för felsökning.
4. Man måste ha förberett informationsströmmarna så de blir lätta att felsöka.

Nedan kommer jag att koncentrera mig på de två sista punkterna, men de andra är naturligtvis inte mindre viktiga för det.

Verktyg för felsökning

Den mest grundläggande principen för felsökning är att det ska gå att inspektera meddelandeströmmarna – alltså hur meddelanden ser ut i verkligheten och inte i teorin.

Den ökade användningen av XML är mycket positiv eftersom informationen är lättare att undersöka rakt upp och ner i en enkel textredigerare eller i ett XML-verktyg. Till och med en vanlig webbläsare kan visa rå XML överskådligt. Äldre meddelandetyper som används är ofta utförda som platt textinfo eller som hålkortsbilder (jodå, sådana finns fortfarande i drift) och de är också ofta lätta att inspektera. Däremot är binärformat och OO-kommunikation vanligen hopplösa att se in i utan speciella verktyg.

Förutsättningen är förstås att man överhuvudtaget lyckats fånga meddelandeströmmen.

En god potential för att kunna inspektera flödena är mellanprogramvara. Det finns många sorter, såsom kölösningar, ESB (Enterprise Service Bus), EAI (Enterprise Application Integration), WSM (Web Services Management) och SOA Governance. Se för övrigt också min trendspaning "ESB:n är död – leve ESB:n!" som framhåller att de sistnämnda typerna av mellanprogramvara finns. Driftövervakningsprogramvara som Tivoli, Unicenter, Openview eller Patrol kan också hjälpa till.

Jag vill faktiskt också slå ett slag för den gamla hederliga protokollanalysatorn, eller linjelyssnaren. Finns som enkel programvara till bärbar PC, koppla in den till samma nätverkssegment som servern eller anroparen, och voilà, du ser exakt vad som skickas. Och kanske lika viktigt i många fall, vad som returneras. Dessutom brukar det vara lätt att filtrera information så att du ser det som är relevant. Du kan vanligen få ut statistik, vilket är bra om det finns prestandaproblem eller om ni inte är överens om antalsbaserade kostnader. Den kanske största nackdelen med linjelyssnaren, förutom en viss inlärningströskel, är att informationen ibland är krypterad och då blir det trassligt. En bieffekt med att lyssna på trafik på det här sättet är för övrigt att du kanske råkar upptäcka strömmar som definitivt borde vara krypterade men som inte är det, detta har hänt mig flera gånger!

Ett annat sätt att fånga informationen är webbserverloggar. All användning av Web Services passerar via en http-server, i många fall en enkel Apache eller Microsoft IIS. Här finns vissa möjligheter att slå på ökad loggning för att se mer info. Ibland finns det

insticksmoduler enligt specifikationerna ISAPI, NSAPI eller Apache-API som kan tappa av info för felsökning.

Jag rekommenderar också starkt att applikationsprogrammen ska innehålla loggningsmöjligheter för informationsutväxlingen. Ni som utvecklar SOA-kod kommer förr eller senare att behöva försvara er i samband med att informationsutbyte inte visar sig bli interoperabelt i praktiken. Att då ha en egen logg att visa på, och att leta i vid felsökning, är guld värt. Black box-tänkandet inom SOA innebär att varje programmerare ska vara misstänksam mot inkommande data och göra stark indatavalidering.

Att också logga inkomna meddelanden gynnar denna nyttiga skepticism. Man kan ha en driftsparameter för att stänga av loggningen när en lösning väl är uttestad. Men jag påstår faktiskt att man gott kan behålla loggningen påslagen hela tiden, ifall det inte visar sig ge prestandaproblem. I många lägen har man intermittenta problem och vill felsöka bakåt i tiden och det går ju inte om loggen inte varit på. Kombinera med en enkel städrutin så att inte gammal logginfo fyller diskarna. För övrigt ska loggar vanligen ha säkerhetskopiering!

Förbered informationsströmmarna för felsökning

Om vi nu förutsätter att vi kan inspektera meddelandena, är det kanske något vi kan förbättra för att göra felsökning enklare vad gäller själva meddelandeinnehållet?

Jag anser att alla SOA-meddelanden bör utrustas med några obligatoriska extrafält. Att införa sådana fält när en tjänst nyutvecklas kostar i princip ingenting, men att tillföra dem i efterhand, när man väl sitter i klistret, är synnerligen krångligt och dyrt.

Extrafälten är till för olika sorters felsökning, men också för mer tjänsteadministrativa ändamål. Skulle till exempel en SOA Governance-lösning senare tillföras och denna erbjuder finesser som överlappar funktionen hos något av fälten spelar det ingen roll att det blir både hängslen och livrem. De här korta fälten kostar ingenting prestandamässigt. Och även en SOA Governance-lösning kan ha buggar...

Nedan följer konkreta förslag på sådana fält, där jag också försöker motivera varför de är nyttiga. Listan är relativt lång, så självklart finns det fall där färre fält behövs, och andra fall där det finns anledning att ha ytterligare fler. Ansvaret för att ett optimerat antal obligatoriska fält införs, hamnar förmodligen hos en övergripande arkitektroll. Denna roll måste ha ett tydligt mandat, och i fall då man köper lösningar, blir fältspecifikationen en viktig del i kravdokumenten inför upphandling.

Testindikator
Berättar ifall detta är ett testmeddelande.
Ex: ”TEST” eller ”PROD”.
Observera att SOA-system bör designas så att produktionsmiljöerna kan acceptera att forsla även testmeddelanden. Orsaken är att det är så mycket av driftsinställningar som kan fallera, och man behöver ofta kunna funktionstesta även produktionsmiljöerna utan att äkta operativt data påverkas.
Likaså kan meddelanden för prestandatest behöva skickas genom produktionsmiljöer.

AnroparensTidsstampel
Finupplöst tidsinfo för när anroparen/sändaren skickade ut meddelandet.
Ex: ”2009-04-21 14:38:21.012 GMT”
Behövs för att vid felsökning relatera detta meddelande till andra meddelanden, samt till ev. vidareförmedling. Behövs också som underlag för prestandamätning.

Observera att klockan kan gå olika rätt i olika datorer, varför ”löpande delta” kan behöva användas vid prestandamätning. Observera också att vissa klock-API:er inte har så hög precision som det ser ut på ytan.

Granssnittsversion
Versionsidentitet hos gränssnittet (kontraktet).
Ex: ”1.2.5”.
Här har vi egentligen en stor fråga: Hur väljer man att versionshantera SOA-gränssnitt? Beroende på hur man gör kan det vara mycket vettigt att explicit skriva in i detta fält vilken version sändaren förväntar sig att det ska vara på gränssnittet. Denna förväntan bör förstås också valideras i mottagande ända.

InternMjukvaruversion
Versionsidentiet hos bakomliggande programkod i anropare/sändare.
Ex: ”3.5.1.45”.
Man kan tycka att black box-principen innebär att ingen utomstående ska få vara intresserad av intern mjukvaruversion – det är kontraktets version som betyder något. Men i stora, sammansatta system är det vid felmisstanke i praktiken inte alls säkert att man snabbt kan ta reda på att i Server4 var det programversion 3.5.1.44 som var i drift kl 07:15 den 3:e juni. Då underlättar det mycket att kunna citera till motparten vilken mjukvaruversion man anser medförde ett problem.

AnropadUtav
Ange anropande/sändande applikationsnamn e.dyl.
Ex: ”EMEA_Jeeves”.
I större sammanhang, med anrop mellan många SOA-parter, kanske över hela världen, blir det svårt att få spårbarhet, lyckas pensionera gamla gränssnitt, undersöka olämplig anropsfrekvens med mera, ifall man inte lätt hittar vad det är för system som anropar.

Man kan invända att en säkerhetsarkitektur ska kunna ge denna info, men ibland behövs inte en hög säkerhetsnivå, och ibland är certifikatsinfo etc. för trasslig att leta i, medan ett enkelt systemnamn är lätt att se för programmeraren.

SOA Governance-verktyg kan också ge liknande spårbarhetsinfo, men den kan vara alltför tekniknära (IP-nummer etc).

AnropandesApplikationsansvarige
Ange t.ex. funktions-e-postlåda till systemansvarig för den anropande/sändande applikationen.
Ex: ”sysadmin_Jeeves@EMEA.xyz.se”.
Samma motivering som föregående, men just en e-postadress är extra behändig att kunna kontakta via i samband med att man vill ta bort gamla tjänsteversioner när man har 18 i drift t.ex.

Meddelandesekvensnummer
Ange sekvensnumret för detta meddelande, ifall det är i en serie relaterade meddelanden.
Ex: ”12345”
I vissa fall kräver verksamhetslogiken att meddelandeordning vidmakthållas. I samband med kölösningar, lastdelning etcetera är det inte säkert att ordningen kan garanteras. Med detta sekvensnummer kan slutmottagaren sortera rätt.

UniktMeddelandeID
Ange en identitet som är unik inom verksamhetsdomänen, eller ännu hellre globalt unik, såsom ett GUID.
Ex: ” 3F2504E0-4F89-11D3-9A0C-0305E82C3301”
Omsändningar av data kan förekomma vid tekniska felsituationer. Driftpersonal kan råka åstadkomma att samma data sänds igen i en kö. Det kan ske vid återläggning av backup i något av skikten. Lösningar för att ersätta ACID-transaktioner kan resultera i omsändning.
I många fall leder det till verksamhetsproblem ifall sådant omsänt data bearbetas flera gånger.

Dubletteliminering är därmed viktig (eng. idempotency) och kan baseras på unika meddelandeidentiteter. I enstaka fall kan ett välkonstruerat sekvensnummer istället användas för detta ändamål, men observera omvänt att ett GUID inte nödvändigtvis ger sekvensinformation.

DebiteradPart
Ange den organisatoriska part som ska debiteras för anropet/sändningen.
Ex: ”Orgnr55665566-6003_Kst3023”
Beroende på hur priskontrakt är utformade kan det behövas fält av detta slag för att debitering och fakturering ska kunna ske. Samt inte minst viktigt, även för att intern kostnadskontering ska kunna utföras för att fördela avgifter.

InfoversionForeUppdatering

Ange version hos info före uppdatering.
Ex: ”1234567”
I de fall som en meddelandeutväxling utförs bakom ett användargränssnitt där registervård och andra uppdateringar kan ske, behövs oftast någon slags lösning för ”optimistic locking”.

Fältet innehåller versionsnummer eller liknande hos informationen när den lästes från datalagringen. Samma nummer skickas tillbaks vid uppdatering. Ifall datalagringens versionsnummer då är ett annat, innebär det att någon annan har uppdaterat datat i mellantiden, så lämplig undantagshantering måste utlösas.

Kommentar: I de fall där kommunikationsmönstret är ”request-response”, ska ett svarsmeddelande ha fältinnehåll som avspeglar den svarandes förhållanden – man ska inte eka anroparens info.

SOA-ping

En synnerligen användbar liten funktion för felsökning är en applikations-ping eller SOA-ping. Här menar vi alltså inte det traditionella ping-kommandot på nätversknivå, utan en liknande funktion på applikationsnivå.

En SOA-ping är ett tjänstegränssnitt som egentligen inte utför något, men som returnerar ett svar. Att skapa ett sådant extra gränssnitt i samband med att en viss tjänst eller gruppering av tjänster nyutvecklas kostar i stort sett ingenting, men är dyrare att tillföra senare.

En SOA-ping kan bland annat användas till följande:

- Att verifiera att en SOA-lösning verkligen är igång. Driftövervakningsplattformar kan lätt fås att skicka ett SOA-pinganrop exempelvis varje minut så att larm kan skapas proaktivt ifall tjänsten gått ner (extra nyttigt för tjänster som har lägre användningsfrekvens men som ska ha hög tillgänglighet).
- Svaret kan innehålla bakomliggande programkods interna versionsnummer. På det sättet kan man verifiera att man lyckats installera rätt programkod på rätt server. Viktigt för drift, men ännu mer användbart vid integrationstester där det kan vara svårt att synka vilken version som vem testar var.
- Se även det föreslagna fältet InternMjukvaruversion ovan.
- Testa viss prestanda och latenstid för ett skikt i en SOA-lösning.
- Därvid kan eventuellt tidsinformation skickas med i meddelandet, se det föreslagna fältet AnroparensTidsstampel ovan.

- För mer info om latensproblemet, läs gärna trendspaningen ”Tidsglapp hotar Molnet?”.

Dock bör inte en SOA-ping skickas vidare, i samband med flerskiktade lösningar. Det finns till och med risk för ohanterliga ping-stormar om en sammansatt tjänst pingar vidare till ett antal andra tjänster, och någon av dem råkar återanvända ett tjänstegränssnitt i den första tjänsten. Om man vill testa en hel kedja ska man hellre använda testmeddelanden som är verksamhetsspecifika. Se det föreslagna fältet Testindikator ovan.

Sven-Håkan Olsson är en fristående konsult som särskilt arbetar med att kombinera verksamhetsnytta med teknikhöjd. Han har en lång karriär sedan 70-talet som it-konsult (it-arkitektur, systemdesign, programmering, reviewer, utredningar, kursledning). Sven-Håkan är också medgrundare av Know IT och var dess teknikchef 1990-2003. Han utsågs till en av "Sveriges topputvecklare" av Computer Sweden 2008.

Sven-Håkan håller regelbundet kurser åt Dataföreningen Kompetens, till exempel ”Cloud Computing integration och migration”. Läs gärna mer på hans blogg www.definitivus.se.

Sven-Håkan Olsson

- - - - -

Som publicerat på www.trendspaning.sei maj 2009
Sven-Håkan Olsson, Definitivus AB
www.definitivus.se