SOA, dobre. Ale čo kvalita?

IGOR AUFRICHT

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
1234.ia[zavináč]gmail[.]com

Abstrakt. Čoraz väčší rozmach internetu a webových služieb so sebou prináša nové princípy vývoja a implementácie systémov. Jedným z často používaných spôsobov je aj princíp SOA (architektúry orientovanej na služby). Princíp je známy, s ním však prichádzajú aj nejasnosti. Ideme vyvíjať nový systém. Chceme ísť s dobou a tak sa rozhodneme navrhnúť systém s využitím architektúry SOA. Vieme, o čo ide. Vieme, ako má systém fungovať. Vieme, ako ho navrhnúť. Vyvstávajú pred nami ale mnohé otázky. Medzi najpálčivejšie patria otázky týkajúce sa kvality systému. Bude systém dostatočne rýchly? Bude bezpečný? Bude sa dať jednoducho testovať? A čo rozšíriteľnosť? Možnosť prepojiť s inými systémami? Na tieto a zopár ďalších otázok spojených s kvalitou projektov implementovaných architektúrou SOA sa snaží odpovedať práve táto práca.

SOA, dobre

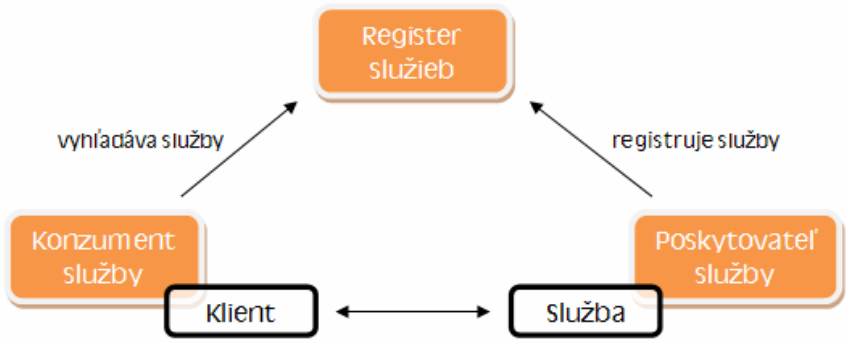
Architektúra orientovaná na služby (SOA – service-oriented architecture) patrí medzi najznámejšie a najviac skloňované pojmy v oblasti vývoja softvérových systémov. Čo však predstavuje?

SOA je *paradigma na organizovanie a využívanie distribuovaných schopností, ktoré môžu mať rozdielne pole pôsobnosti* [3]. Vo všeobecnosti sú rôzne entity (ľudia alebo organizácie) schopné riešiť problémy alebo podporiť riešenie problémov, ktoré zabraňujú iným entitám v dosiahnutí cieľa. Je prirodzené, ak potreby jedného človeka je schopný uspokojiť iný človek. Každý totiž vie riešiť nejaký problém a v konečnom dôsledku nás nemusí zaujímať, ako daný človek problém vyrieši. Prenesené do sveta softvérových systémov, na riešenie konkrétnych problémov sú určené konkrétne systémy a jeden systém môže využívať služby iného a pritom ho nemusí zaujímať, ako sa daný problém vyrieši.

V systéme sa vyskytujú dva druhy entít – poskytovatelia služieb a konzumenti služieb. *Poskytovateľ služby* je entita, ktorá zdieľa svoje schopnosti riešiť problémy z určitej oblasti. Entita, ktoré služby využíva, sa nazýva *konzument služby*. Oba druhy entít musia navzájom vedieť o svojej existencii, preto sa často v systéme objavuje ešte

Manažment v softvérovom inžinierstve, október 2007, s. 1-1.

register služieb. Ten obsahuje informácie o dostupných službách a ich poskytovateľoch (Obr. 1).



Obr. 1. Základný princíp SOA.

Ale čo kvalita?

Hlavnými architektonickými princípmi SOA sú enkapsulácia, modularita, znovupoužiteľnosť, vysoká abstrakcia a nízka previazanosť služieb [1]. Tieto vlastnosti sľubujú mnohé výhody pri použití architektúry SOA – napríklad jednoduchú rozšíriteľnosť, modifikovateľnosť a testovateľnosť. Na tému vytvárania SOA systémov bolo publikovaných veľa článkov (napr. [4]), ale relatívne málo výskumu sa venuje atribútom kvality výsledného produktu.

A práve otázkou kvality systémov implementovaných architektúrou SOA sa budem zaoberať v nasledujúcich kapitolách.

Bude náš systém dostatočne rýchly?

Prvá vec, ktorá ma napadla, keď som uvažoval nad dopadom využitia SOA na kvalitu systému, bola rýchlosť. Inak povedané výkon systému.

Kvalitu systému z hľadiska výkonu ovplyvňujú rozličné atribúty, ako sú odozva systému, maximálny počet paralelne spracovávaných požiadaviek alebo rýchlosť spracovávania požiadaviek.

Keďže systém je zložený z viacerých na sebe nezávislých komponent, vyzerá to, že zrejme bude pomalší ako jeho ekvivalent s tradičnou architektúrou. Záleží to ale na viacerých faktoroch.

SOA systémy sú často implementované formou *distribuovaného systému*. Z toho vyplýva nutnosť prenosu údajov po sieti, čo evidentne prinesie spomalenie. Myslím ale, že pokiaľ je systém implementovaný ako distribuovaný, má to svoje dôvody. Je vhodné sa pri tvorbe nového systému zamyslieť nad tým, či je vôbec nutné vyvíjať systém ako distribuovaný. Implementovať interný systém na správu výkazov práce malej firmy ako webovú službu je evidentne zbytočné.

Ďalším faktorom je princíp *enkapsulácie*. Keďže služby majú byť čo najviac autonómne, je nutné stanoviť jednotný formát prenášaných dát (t.j. požiadaviek od konzumentov a odpovedí od poskytovateľov služieb), aby si účastníci navzájom rozumeli. To prináša na každej strane nutnosť konvertovať údaje, čo má za následok zníženie výkonu systému.

S tým súvisí aj *multiplatformovosť*, jednotlivé komponenty môžu byť implementované v rôznych jazykoch. Na jednej strane to môže spôsobovať ďalšie zníženie výkonu systému, ale na druhej strane vhodný výber implementačného jazyka pre jednotlivé služby môže priniesť aj výrazné zrýchlenie – mnohé úlohy sa v niektorých jazykoch dajú vyriešiť efektívnejšie ako v iných. Takže myslím, že čo sa týka vplyvu princípu enkapsulácie na výkon systému, je to nerozhodne. Podstatný je vhodný výber implementačných prostriedkov.

```
<env:Header>
 <m:reservation xmlns:m="http://travelcompany.example.org/reservation"
         env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
          env:mustUnderstand="true">
  <m:reference>uuid:093a2da1-q345-739r-ba5d-pqff98fe8j7d</m:reference>
  <m:dateAndTime>2001-11-29T13:20:00.000-05:00</m:dateAndTime>
 </m:reservation>
 <n:passenger xmlns:n="http://mycompany.example.com/employees"
         env:role="http://www.w3.org/2002/06/soap-envelope/role/next"
          env:mustUnderstand="true">
  <n:name>John Q. Public</n:name>
 </n:passenger>
</env:Header>
<env:Body>
 <p:itinerary
   xmlns:p="http://travelcompany.example.org/reservation/travel">
  <p:departure>
    <p:departing>New York</p:departing>
    <p:arriving>Los Angeles</p:arriving>
    <p:departureDate>2001-12-14</p:departureDate>
    <p:departureTime>late afternoon</p:departureTime>
    <p:seatPreference>aisle</p:seatPreference>
  </p:departure>
  <p:return>
    <p:departing>Los Angeles</p:departing>
    <p:arriving>New York</p:arriving>
    <p:departureDate>2001-12-20</p:departureDate>
```

Obr. 2. XML môže vzhľadom na nízku hustotu informácie pri veľkých objemoch dát spôsobovať problémy.

Ešte jedna vec môže ale ovplyvniť výkonnosť SOA systému, a to je *štandardný formát správ*. Existujú určité štandardy pre prenos dát a mnohé z nich (napríklad protokol SOAP) sú založené na jazyku XML (Obr. 2). Ten má vzhľadom na svoju štruktúru nízku hustotu obsiahnutej informácie, čím sa zväčšuje množstvo prenášaných dát (môže sa vyskytnúť až 10-násobný nárast veľkosti oproti binárnemu formátu). To môže pri veľkých množstvách dát spôsobovať spomalenie zapríčinené prenášaním veľkého množstva údajov. Takisto je nutné pri každej požiadavke a odpovedi na oboch stranách XML parsovať, validovať a transformovať, čo pri veľkých objemoch údajov môže byť náročné na výpočtové prostriedky. Môj názor ale je, že pri menších systémoch, kde používanie XML spôsobuje rozdiely rádovo v mili- až

mikrosekundách, to nie je problém. Taktiež je na architektovi systému, aby sa vyhol prenášaniu veľkého množstva dát. Vo väčšine prípadov sa tomu dá totiž predísť.

Pre tvorcov a prevádzkovateľov systému z toho vyplýva nasledovné: v prvom rade sa snažiť *vytvoriť systém čo najrýchlejší a s minimálnymi nárokmi na prostriedky*. Ak sa ale napriek tomu objaví spomalenie systému, netreba vešať hlavu. Vždy je možné *skontrolovať výkonnosť služieb*, ktoré systém využíva (tie môžu byť príčinou problému), prípadne *zvýšiť prostriedky* dostupné *na beh systému* (rozšírenie linky, pridanie serverov, zvýšenie kapacity servera, atď).

Bude bezpečný?

Ďalším kvalitatívnym atribútom systémov je bezpečnosť distribuovaných systémov. Na tú sa dá pozerať z dvoch pohľadov – jedným je zabezpečenie prenášaných dát a druhým autentifikácia.

Zabezpečenie dát

Keďže SOA systémy sú distribuované, a teda mnohé z nich prenášajú údaje cez nedôveryhodné siete, je nutné zabezpečiť ochranu údajov pred zneužitím. To znamená zabezpečiť prenášané údaje tak, aby k nim nemali prístup nepovolané entity. Nikto z nás (česť výnimkám) by si predsa neželal, aby nejaký systém posielal po internete voľne čitateľné napríklad jeho autorizačné údaje pre prístup do e-bankingu!

Preto je viac než vhodné posielané údaje *šifrovať*. Myslím ale, že o šifrovaní údajov je dostupných množstvo materiálov a existuje množstvo riešení tohto problému, takže sa touto témou nebudem ďalej zaoberať. Stačí len vybrať a použiť to správne riešenie. A správne ho použiť.

Autentifikácia

Táto téma je už omnoho zaujímavejšia. Je nutné autentifikovať všetkých účastníkov využívania služieb, ako poskytovateľa a konzumenta, tak aj register služieb.

Ako prvé je nutné zabezpečiť autentifikáciu *konzumenta služby* (užívateľa). Nie je žiadúce, aby mohol službu využívať ten, kto k nej nemá mať prístup. Takisto treba rozlišovať, ktorí užívatelia majú ku ktorým častiam systému prístup. Tu je na mieste použiť nejaký autorizačný mechanizmus.

Často sa tiež vyskytnú situácie, v ktorých musí systém využívať služby tretích strán. Tu je dôležité zabezpečiť správnu autentifikáciu *poskytovateľa služby*.

V niektorých prípadoch sa objavuje potreba autentifikovať aj *register služieb*, aby nedošlo k tomu, že budeme využívať nastrčenú službu.

Takže ako zabezpečiť systém?

Na odstránenie týchto problémov existuje mnoho riešení - bezpečnosť prenášaných dát sa môže zvýšiť použitím vrstvy *SSL* (Secure Sockets Layer) alebo jeho nasledovníka *TLS* (Transport Layer Security). Použitie je väčšinou bezproblémové a tak si myslím, že vývojár by mal siahnuť po tomto riešení. Obávam sa ale, že aj tak nebude zaručená úplná bezpečnosť (hlavne pri použití staršieho SSL), takže pri aplikáciách náročnejších na bezpečnosť je dobre sa poobzerať po inom riešení (tieto riešenia sa nemusia navzájom vylučovať). Ďalšou možnosťou je využitie *digitálneho podpisu*, ako aj *asymetrického šifrovania*. Podľa môjho názoru ale na zabezpečenie menšieho systému stačí použiť vyššie spomenuté vrstvy SSL a TLS. Samozrejme ak ich použitie nie je možné, je vhodné siahnuť po ďalších menovaných riešeniach.

Čo sa autentifikácie a autorizácie užívateľov (konzumentov) týka, taktiež existujú funkčné riešenia, napríklad systém *Kerberos*. Ten je založený na posielaní tzv. *lístka* (angl. ticket), ktorý určuje, či a ktoré operácie má užívateľ právo vykonať.

Na autentifikáciu poskytovateľov služby je možné použiť *digitálny certifikát*, na internete bežne používaný.

Ďalšou netriviálnou možnosťou je použiť riešenie *WS-Security* [2]. Tento vyvíjajúci sa štandard predstavuje komplexný model zabezpečenia pre webové služby. Špecifikácie WS-Authorization, WS-Privacy, WS-Trust, WS-Federation, WS-Policy, and WS-SecureConversation myslím už svojím názvom napovedajú, na čo sú určené. Tento štandard bude pravdepodobne jedným z najdôležitejších štandardov na zabezpečenie webových služieb.

Existuje zopár ďalších štandardov (napríklad *SAML* a *XACML* [2]), ktoré tu ale nebudem rozoberať.

Moje odporúčanie tvorcom systémov je teda nasledovné – všetky problémy už boli vyriešené (alebo sa o to niekto aspoň snažil), takže pre *malé systémy* by som použil *štandardné riešenia* - SSL, TLS, prípadne Kerberos, vzhľadom na náročnosť aplikácie by som uvažoval aj nad digitálnym podpisom. Jednoznačne som za použitie digitálnych certifikátov na autentifikáciu služieb tretej strany.

Určite by som sa zatiaľ *vyhol štandardu WS-Security*, nakoľko sa len vyvíja a tým pádom sa neustále mení. Taktiež preň pravdepodobne nie sú úplne vhodné implementácie. V budúcnosti po dokončení a štandardizácii by som po vhodnom riešení založenom na tomto štandarde zrejme siahol – pre veľký systém určite, pre menší podľa toho, aké implementácie by boli dostupné.

Bude sa dať jednoducho testovať?

Testovanie SOA systémov je kapitola sama o sebe (preto aj má extra kapitolu). Z uhľa pohľadu testera môžeme testovanie rozdeliť na dva spôsoby. Testovanie metódou čiernej skrinky a bielej skrinky (existuje aj metóda šedej skrinky, ale prekvapivo je to kombinácia predchádzajúcich dvoch).

Testovanie metódou *čiernej skrinky* (black box testing) považuje systém (v našom prípade službu) za čiernu skrinku (Obr. 3). Nezaujíma nás, ako služba vnútorne funguje. Testovanie touto metódou je ako stvorené pre SOA systémy, keďže pozostáva v podstate iba z toho, že pošleme požiadavku na službu a skontrolujeme výsledok. Aj tak sa ale môžu objaviť problémy. Ak sa služba nachádza na inom serveri – server môže byť nedostupný a to spôsobí chybu, ktorej príčina sa môže ťažko určovať. Ďalším problémom môže byť použitie registra služieb. Teda pred samotným zavolaním služby nemusíme vedieť, ktorý poskytovateľ nás obslúži (niekedy to môže byť problém zistiť aj po prijatí odpovede). Ešte jeden, aj keď malý, problém môže nastať, keď prídu údaje v zlom formáte (nevalidné XML, nedodržanie protokolu, atď.). Tento problém sa ale ľahko rieši.

Obr. 3. Black box testing. Ako šité na mieru SOA.

Metóda *bielej skrinky* (white box testing) napriek tomu môže byť zložitejšia, keďže sa spolieha na prístup k zdrojovému kódu služby. Ten však nemusí byť prístupný (veď to je aj hlavný princíp SOA). Takže touto metódou je lepšie testovať len nami implementované služby a v prípade chýb v iných službách (k zdrojovému kódu ktorých nemáme prístup) upovedomiť vývojára danej služby.

Na záver tejto kapitoly teda môj odkaz vývojárom – testujte vlastný kód ako zvyčajne a pridajte testovanie služieb ako celkov. Pre menšie systémy sa takéto testovanie zaobíde bez väčších problémov. No a v prípade chýb v iných službách sa obráťte na ich vývojára. A samozrejme nezabudnite dať dôraz na rýchlosť, dostupnosť a spoľahlivosť systému.

A čo rozšíriteľnosť? Modifikovateľnosť?

Opäť sa dostávame ku kapitole, kde je použitie SOA výhodou. Rozšíriteľnosť (modifikovateľnosť) je možnosť rozšíriť (upraviť) systém jednoducho a s minimálnymi nákladmi.

Rozšíriť SOA systém pridaním služieb je relatívne jednoduché, keďže jednotlivé služby sú navzájom nezávislé.

Pri *modifikovaní* funkčnosti existujúcej služby treba dať pozor na to, aby sme zachovali funkčnosť ostatných častí programu. Problém ale vzniká, keď meníme rozhranie služby. Na ňom totiž môže závisieť veľa iných služieb.

Z toho vyplýva dôležitosť správneho návrhu rozhraní a dôraz na modularitu systému (snažme sa teda o čo najmenšiu previazanosť medzi modulmi). Myslím, že dôsledné dodržanie týchto odporúčaní sa nám odvďačí jednoduchosťou údržby systému a jeho zmien.

Bude dostupný a spoľahlivý?

Spoľahlivosť je schopnosť systému fungovať po dlhý čas bez zlyhania. Dôležité je, aby sa poslané správy doručili neporušené a aby boli služby systému spoľahlivé.

Ak využívame niektoré zo štandardných riešení prenosu správ, je často najlepšie zabezpečenie *spoľahlivosti správ* prenechať použitým riešeniam, keďže tie s takouto možnosťou rátajú. V prípade, že to nie je možné, dajú sa použiť bežne dostupné algoritmy na zabezpečenie doručenia. S týmto problémom ráta aj štandard WS-Reliability a WS-ReliableMessaging, ku ktorým už existujú funkčné implementácie. Myslím ale, že pre malý systém by to nemal byť problém.

Spoľahlivosť služieb znamená, že služba funguje bezchybne a v prípade zlyhania dá vedieť užívateľovi. Vzhľadom na distribuovanosť systému je hlavným problémom zabezpečenie konzistencie dát. Riešením sú transakcie. Na podporu transakcií existuje niekoľko štandardov ako BTP (Business Transactions Protocol) a WS-Tx (Web Services Transaction). V prípade menšieho systému by som si pravdepodobne dané štandardy prezrel, v prípade prílišnej komplexnosti by som ale zvolil iné (možno aj vlastné) riešenie.

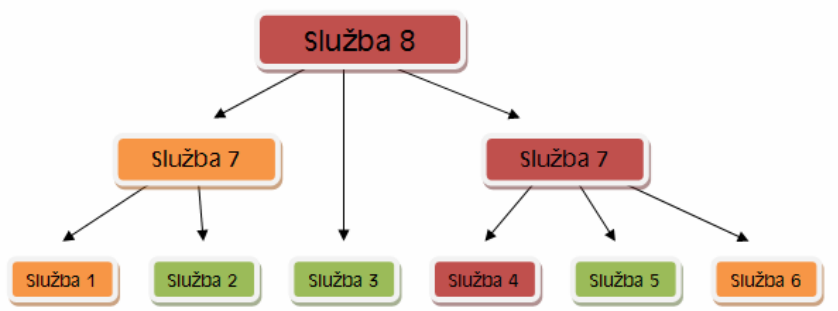
Dostupnosť systému vyjadruje percento času, ktoré je systém funkčný a dostupný. Túto hodnotu znižujú hlavne výpadky serverov. V prípade menších systémov by som riešenie problému prenechal na správcu serveru (napr. v podobe serverového poľa).

Časti systému určujú kvalitu systému ako celku

Posledným, nemenej dôležitým atribútom je kvalita systému ako celku. Architektúra SOA už z princípu umožňuje skladanie a vnáranie služieb, takže jedna služba je zložená z viacerých ďalších služieb. Notoricky známe porekadlo, že *reťaz je len tak silná ako jej najslabší článok*, platí aj tu (Obr. 4). Ak systém využíva nekvalitné služby, je veľká pravdepodobnosť, že aj systém ako celok bude nekvalitný.

Samozrejme nie je zriedkavé, že na poskytovanie jednej služby existuje viac poskytovateľov. Vtedy si treba správne vybrať toho, ktorý je pre nás najkvalitnejší (napríklad pokiaľ tvoríme real-time aplikáciu, kde je nutná veľmi rýchla odozva, môžme zvoliť rýchlejšiu službu s menším množstvom poskytovaných údajov pred komplexnejšou, avšak pomalšou službou).

Z toho jednoznačne vyplýva - dobre si preštudujte kvalitatívne atribúty služieb, ktoré chcete využívať. Hlavne také, ktoré ste neimplementovali vy sami. Najlepšie ešte pred tým, ako ich používať začnete.



Obr. 4. Zlá voľba služieb môže viesť k nízkej kvalite systému. Na obrázku sú kvalitné služby zobrazené zelenou farbou, menej kvalitné oranžovou a nekvalitné červenou. Ako si iste ľahko domyslíte, stačí aby jedna služba bola nekvalitná a môže byť nekvalitný celý systém.

Aké z toho plynie ponaučenie?

Zazvonil zvonec a poúčania je koniec. Mohli ste si prečítať objektívne ako aj subjektívne hodnotenie niektorých kvalitatívnych atribútov, ktoré podľa mňa odlišujú SOA od iných, hlavne klasických architektúr. A samozrejme aj nejaké odporúčania autora. Niektorým oblastiam som sa venoval viac, iným menej, ale dúfam, že ste si niečo z čítania mojej práce odniesli.

Počkať, počkať, ešte nekončíme! Možno si teraz vravíte – hm, a nezabudol on na *použiteľnosť*? Veď z hľadiska používateľa to je jedna z hlavných kvalít systému (ak nie tá najdôležitejšia!). Túto tému som vynechal naschvál, keďže si myslím, že použiteľnosť systému závisí na tom, ako je spracované používateľské rozhranie (čo sa pri SOA systémoch vo všeobecnosti nijak nelíši od rozhrania klasických architektúr) a do akej miery sú splnené ostatné požiadavky na kvalitu systému. Takže túto tému nechám na iných.

Na záver len toľko - nič nie je čiernobiele. Na každý problém existuje viac spôsobov riešenia, len ich treba objaviť. A práve toto je jeden z hlavných kľúčov k vývoju kvalitných projektov (a nemusia to byť len SOA systémy).

Použitá literatúra

1. Balzer, Y.: *Improve your SOA project plans*, IBM, 2004.
2. O'Brien, L., Merson, P., Bass, L.: *Quality Attributes for Service-Oriented Architectures*. SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments, IEEE Computer Society, Washington, DC, USA, 2007.
3. Organization for the Advancement of Structured Information Standards: *OASIS Reference Model for Service Oriented Architecture V 1.0*. Dostupné na internete: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf (25. 10. 2007)
4. Tsai, W.T.: *Service-Oriented System Engineering: A New Paradigm*, IEEE International Workshop on Service-Oriented System Engineering (SOSE), Beijing, 2005.

Annotation

SOA, ok. But what about quality?

Ever bigger boom of the Internet and Web Services brings some new principles of system development and implementation. One of the most frequently used method is the SOA (service-oriented architecture) principle. Fundamentals are well-known but uncertainties appear. We are going to develop a new system. We want to go with times so we decide to implement a system using SOA architecture. We know what this is in aid of. We want how the system is supposed to work. We know how we can design it. But a lot of questions come up. Among the most relevant issues fall questions about system quality. Will the system be fast enough? Will it be secure? Will it be easily testable? And what about extensibility? Possibility to interconnect with other systems? This and some other questions related to the quality of SOA systems will be answered in this paper.