Integrácia systémov

Servisne a zdrojovo orientovaná architektúra

Peter Grec, Ľubor Šešera

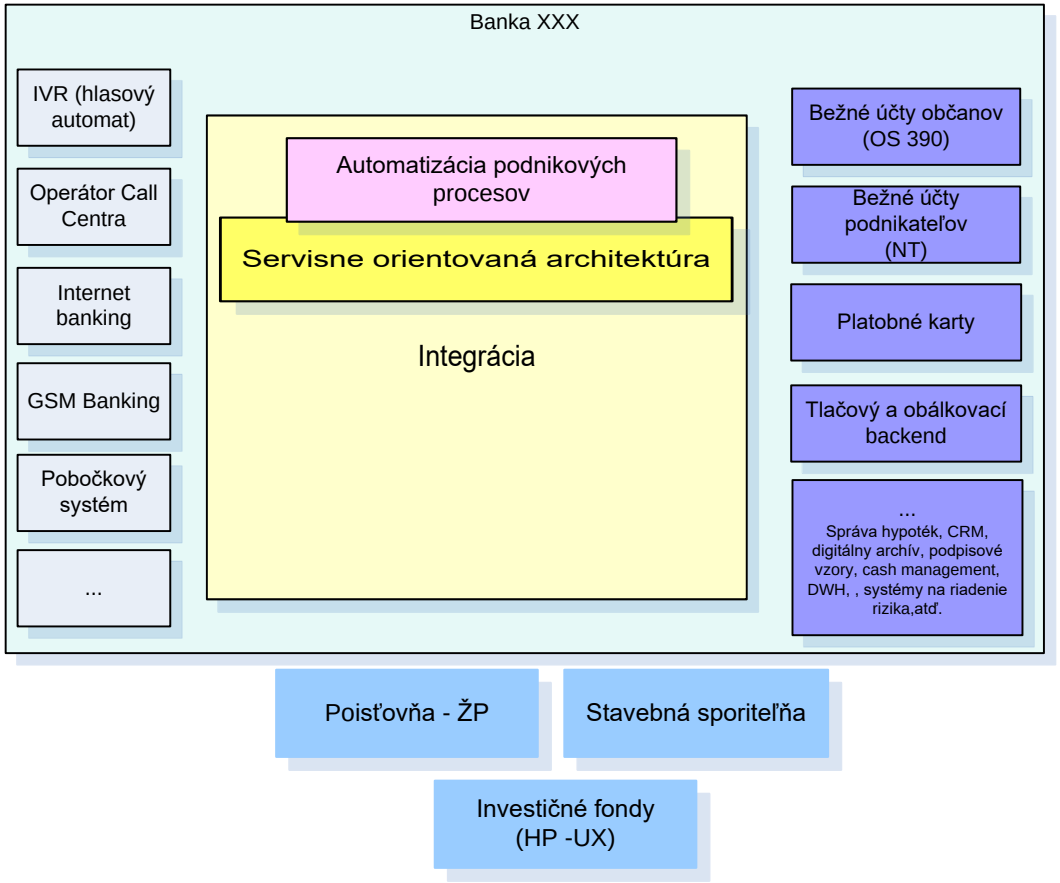
24.10.2023

SOFTEC

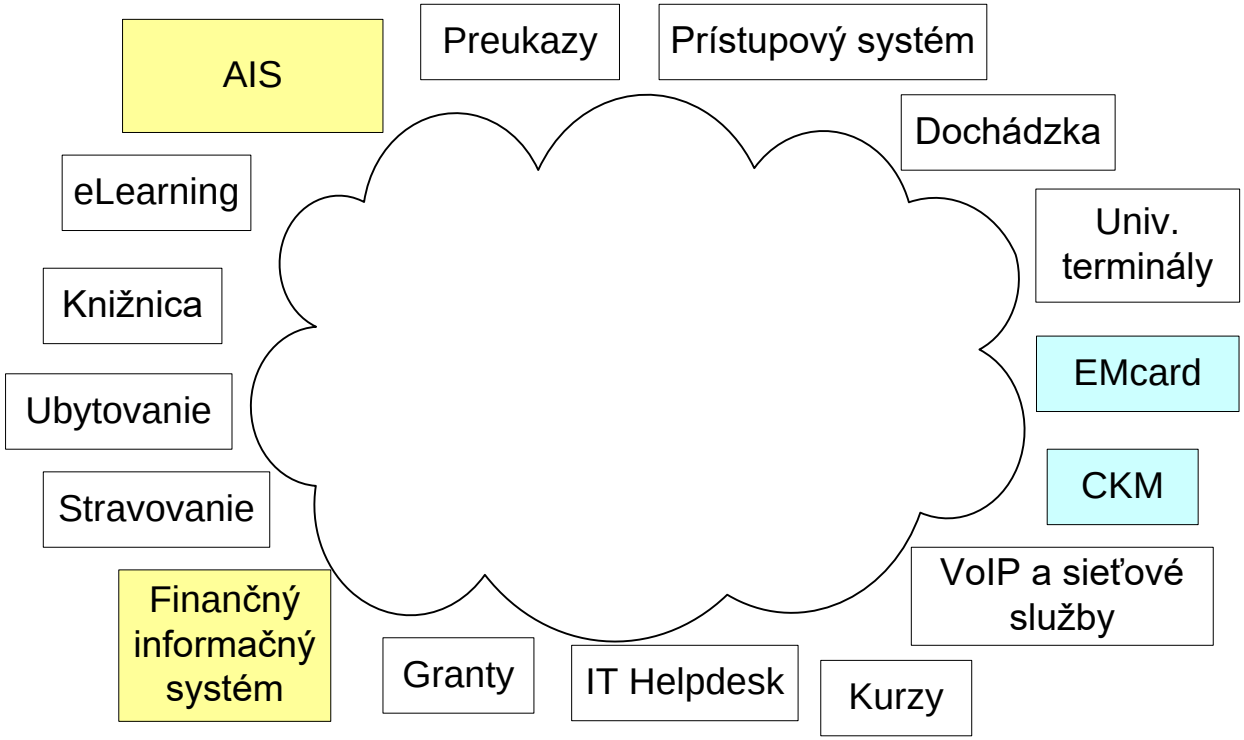
Potreba integrácie systémov

- Softvérové systémy vo firme neexistujú izolovane
  - Príklady:
    - Internet banking / mobil banking a Core banking systém
    - Akademický informačný systém a IS jedálne
- Integrácia aj nad rámec firmy
  - Príklady:
    - Platba platobnou kartou u obchodníka
    - Prenos údajov o študentoch medzi univerzitou a poskytovateľmi zliav

Príklad banka



Príklad - Univerzita



SOFTEC

Požiadavky na integráciu

- Funkcionálne požiadavky
  - Čo má integračné riešenie robiť resp. zabezpečiť
    - Vystaviť funkcionalitu a dáta aplikácie pre použitie v iných systémoch
    - Zabezpečiť "porozumenie" medzi rôznymi systémami - **štandardizácia** prenosových protokolov, formátu prenášaných dát, popisu rozhrania
  - Umožňuje automatizáciu vykonávania funkcií / procesov
    - Nemusíme robiť ručne napr. strata karty → okamžitá blokácia vo všetkých systémoch
- Ostatné požiadavky (kritériá kvality)
  - Aké vlastnosti musí integrácia poskytnúť
    - Spoľahlivosť - Prenos údajov cez sieť je rádovo menej spoľahlivý, než lokálny prenos
    - Dostupnosť - Systém, od ktorého sa požaduje služba, nemusí byť v danom čase dostupný
    - Bezpečnosť – Je vystavený bezpečnostným útokom
    - Výkonnosť - Prenos údajov cez sieť je výrazne pomalší než lokálna komunikácia
    - Konzistencia dát - Dáta sa do druhého systému môžu dostať s určitým časovým oneskorením
    - Modifikovateľnosť – Zmena v jednom systéme by nemala ovplyvniť ostatné integrované systémy
    - ...

Požiadavky na integráciu (2)

- Neexistuje jediný optimálny spôsob integrácie
- Tento závisí
  - Od charakteristiky problému (business požiadaviek)
    - napr. dáta pre Data Ware House môžu byť prenášané dávkovo a nevadí 1 dňové oneskorenie, blokovanie prostriedkov na platobnej karte musí byť online
  - Od technických požiadaviek
    - napr. bezpečnosť integrácie v prostredí internetu, výkonnosť
  - Dostupných integračných technológií
    - Vyvíjali sa v čase

Spôsoby integrácie systémov

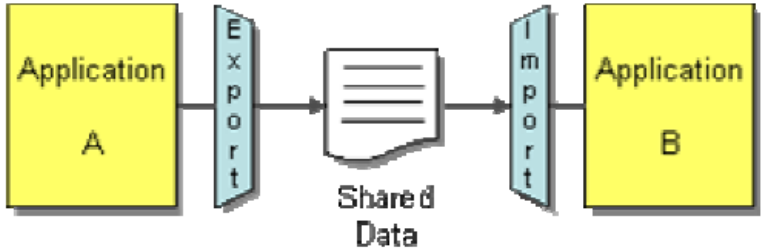
Spôsoby integrácie systémov

1. Prenos súborov
2. Zdieľaná databáza
3. Vzdialené volania
   a) Remote procedure call (first RPC generation)
   b) SOAP-ové webové služby (WS)
   c) REST-ové webové služby

   Synchrónny SOAP a REST sa nazývajú aj Request-Response integračné patterny
4. Posielanie správ
5. Event streaming

SOFTEC

1. Prenos súborov



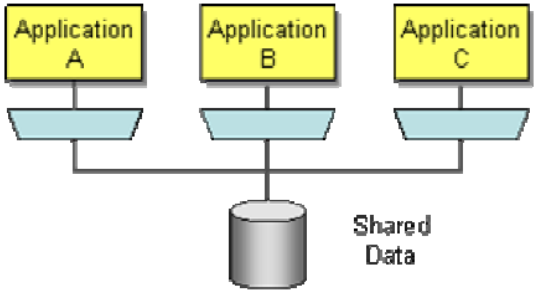
- Najstarší spôsob
- Formát, kódovannie, štruktúra a miesto súboru musia byť dohodnuté
- Súbory sa generujú v pravidelných intervaloch
  - Napr. 1x denne
- Automatizované nástroje na extrakciu, prenos a transformáciu a load dát (ETL)
- Príklad: Medzibankové prevody v SR
  - Každá banka posiela súbor do NBS, ktorý súbory spracováva a distribuuje
- Výhody:
  - Voľná väzba medzi systémami
    - Možná spolupráca heterogénnych systémov
    - Nie je potrebná interná znalosť druhého systému
    - Nedostupnosť druhého systému neohraničuje prvý systém
  - Výkonnosť – môže sa realizovať mimo hlavnej záťaže systémov (v noci)
  - Využitie v Data Ware House ako hlavný integračný pattern

SOFTEC

1. Prenos súborov

- Nevýhody
  - Niekedy je nutné odstaviť systém pri spracovaní dávok (v bankových systémoch t.z.v. uzávierky)
  - Niekedy môžu vzniknúť nekonzistencie dát (postupné spracovanie súborov, súbory za rozdielne obdobia, ...)
  - Vysoká latencia dát (napr. oneskorenie až 1 deň a viac)
  - Veľké súbory (spracovanie naráža na obmedzenia systému)
  - Nízkoúrovňová integrácia (nie je možné využívať služby)

2. Zdieľaná databáza



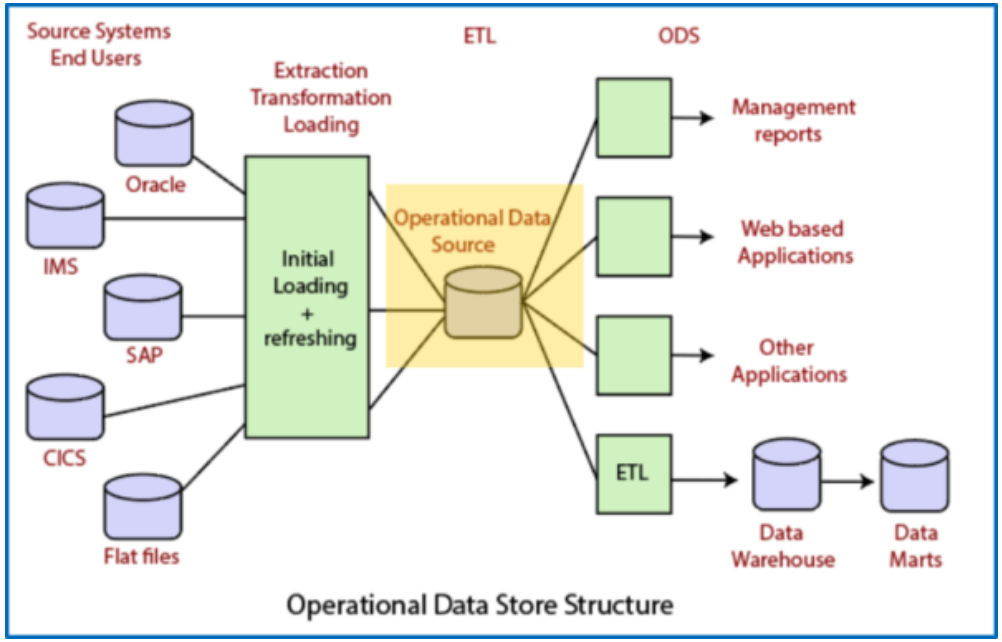
- Využíva štandardizáciu SQL
- Aplikácie zdieľajú
  - Databázový server (jeho zdroje)
  - Schému dát pre zápis a aj pre čítanie
- Príklady
  - Pobočkový systém získava údaje z Core banking systému cez SQL príkazy
- Varianty
  - ODS, Replikovaná DB

SOFTEC

2. Zdieľaná databáza - ODS

Operational data store/source (ODS) v banke

- Spoločná databáza, do ktorej jednotlivé systémy v noci ukladajú svoje vybrané údaje a systémy, ktoré ich potrebujú, si ich z ODS čítajú/skopírujú
- Kombinácia prenosu údajov (replikácie tabuliek) a spoločnej DB
- Znižuje záťaž v peak-och pri on-line integráciách, zvyšuje dostunosť systému
- Všetky dáta z organizácie na jednom mieste v jednom formáte, vzájomne prepojené
- Využitie ako zdroja dát (napr. transakcie, zostatok) ak je core systém neprístupný



Operational Data Store Structure

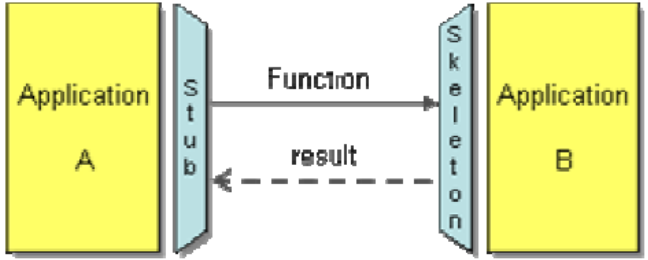
SOFTEC

2. Zdieľaná databáza

- Výhody
  - Dáta sú ihneď prístupné pre 2. systém
  - Vysoká konzistecia dáta - plynúce z transakčnosti systému
    - ACID (Atomicity, Consistency, Isolation, Durability)
  - Dostupnosť – pri oddelenej DB (napr. ODS)
- Nevýhody
  - Silná väzba medzi systémami
    - Jeden systém musí poznať internú štruktúru druhého systému – tá sa môže bez upozornenia zmeniť
    - SQL je synchrónny – záťaž jedného systému alebo siete spomaľuje aj druhý systém
  - Schéma na zápis nemusí vyhovovať rôznym aplikáciám na čítanie
    - Napr. nevhodná normalizácia spôsobujúca komplikované selekty, chýbajúce indexy, potreba agregácie údajov čo sa premieta do komplikovaných a výpočtovo náročných selektov,
  - Spoľahlivosť – SQL nebol navrhnutý pre nespoľahlivé siete
  - Bezpečnosť – SQL nemá osobitné prostriedky na bezpečnosť (integrácia len v dôveryhodnom prostredí)
  - Naviazanosť integrovaných aplikácii na konkrétnu technológiu (napr. SQL)

3a. Remote procedure call (RPC)

- Na rozdiel od integrácií 1) a 2) umožňuje zdieľanie funkcionality, nielen dát
- Lokálne volanie je transformované do sieťového volania
- Príklad
  - Blokovanie stratenej platobnej karty (hoci dnes sa už rieši inak)
- Výhody
  - Vysokoúrovňové programovanie
    - Nízkoúrovňové detaily zabezpečuje middleware alebo knižnice
  - Konzistencia dát – okamžitá zmena v druhom systéme, niektoré implementácia podporujú distribuované transakcie/dvojfázový komit
  - Bezpečnosť – zabezpečuje middleware



SOFTEC

3a. Remote procedure call (RPC)

- Nevýhody
  - Zvyčajne ohraničené na jedno prostredie Java RMI, EJB (v JavaEE/Jakarta), DCOM (Microsoft)
  - Pokus o štandard RPC medzi heterogénnymi prostrediami: CORBA
    - Zložitý, veľa rôznych implementácií, problémy s kompatibilitou
    - Neštandardný protokol – problémy s firewallmi
  - Úzka väzba medzi systémami
    - RPC je synchrónne volanie, spomalenie jedného systému spomaľuje aj druhý systém
    - Výpadok volaného systému spôsobí výpadok volajúceho
    - Zmena volaného spôsobí zmenu volajúceho
  - Výkonnosť
    - RPC navonok pripomína lokálne volanie a programátori si neuvedomovali, že ho treba používať opatrne

3b. SOAP-ové webové služby (WS)

- Podobné ako RPC prvej generácie, ale namiesto proprietárnych riešení sa používajú štandardy
  - XML a SOAP pre kódovanie správ
  - HTTP pre prenos správ
  - WSDL pre popis rozhrania
- Často je vo veľkých firmách spolu s tzv. Zbernicou služieb (Enterprise Service Bus (ESB) – pozri ďalej Posielanie správ)
  - Aby sa nemusel spájať takmer každý systém s každým, ale systémy sa spájajú cez spoločný technologický komponent
    - Analógia hardvérovej zbernice
- Príklady
  - Dnes je najčastejším spôsobom integrácie systémov v bankách
    - Napr. integrácia pobočkového systému s registrom klientov
  - Väčšina veľkých bánk využíva Zbernicu služieb

Ukážka

```
POST /AccountServices HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8;
SOAPAction:http://xxx/AccountServices#GetAccountBalance
...
<?xml version="1.0"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
     < SOAP-ENV:Header>
          <wsa:MessageID>
              uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
          </wsa:MessageID>
     </ SOAP-ENV:Header>
     <SOAP-ENV:Body>
          <m:GetAccountBalance xmlns:m="Some-URI">
              <m:accountNumber>1234567890</m:accountNumber>
          </m:GetAccountBalance>
      </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Prenosový protokol

Typ správy + kódovanie

Povinný root element

Napr. tu je použitý doplnkový štandard WS addressing

Nepovinná hlavička

Rôzne technické info, napr. identifikátor správy

Názov volanej metódy

(Vráť zostatok na účte)

Parameter správy

(číslo účtu)

SOFTEC

3b. SOAP-ové webové služby (WS)

- Výhody
  - Možnosť integrácie heterogénnych systémov (napr. Java s .NET)
  - Konzistencia dát – okamžitá zmena v druhom systéme (webové služby sú však netransakčné – potreba kompenzácii pri rollback)
  - Spoľahlivosť – využíva štandardné a osvedčené internetové technológie
  - Bezpečnosť – dtto
  - Rozširovateľnosť – napr. security – podpisovanie/šifrovanie správ
- Nevýhody
  - Zložitosť – SOAP + rozšírenia je komplexná nadstavba nad HTTP/XML
  - Nezbavili sme sa úzkej väzby medzi systémami
    - Sú možné aj asynchrónne WS – komplexnejšie programovanie pri potrebe zaslania výsledku operácie

3c. REST-ové webové služby

- SOAP WS – primárny cieľ záujmu sú „**SLUŽBY**“ a princípy RPC
- REST – primárny cieľ záujmu sú **„DÁTA“** reprezentované resource a web pohľad na ne
  - URI - identifikácia zdroja (oproti SOAP WS – ID-čko)
  - Linkovanie – klient nemusí zostavovať volanie – len prechádza linkami (oproti SOAP WS – zostavenie volania/SOAP message)
  - Priame využitie HTTP
    - Metódy : GET, POST, PUT, PATCH, DELETE, content type, gzip, error kódy, ...
  - Ako formát dát sa používa JSON (oproti SOAP WS – XML)
- Príklady
  - Dva systémy, ktoré vieme prepojiť cez SOAP WS, vieme analogicky prepojiť cez REST WS
  - REST WS sa navyše používajú v moderných SPA aplikáciách na prepojenie front-endu (Angular, REACT,...) s back-endom (Java, .NET)
  - Namiesto ESB sa používa API Gateway

3c. REST-ové webové služby

Request

```
POST /api/v1/app-instances/7K92GbA48eiBHemijIZ1/cards/588735180161600941/limits HTTP/1.1
X-Message-ID: 7AjUm2SXkxJXA9ppe6Hv
X-Device-ID: 7K92GbA48eiBHemijIZ1
Date: Thu, 4 Nov 2021 14:54:16 +0100
Authorization: Bearer 07BA5816B018BCD263721F3470A77A0CBEF54498
Signature: signature="t5GESEqvrPwV+0v9RAp5kLxpsl8jl1S90hx5OoXvsBw="
Content-Type: application/json; charset=UTF-8
Host: api.example.com
Content-Length: 268

[ {
  "paymentLimitType" : "POS_DAY",
  "value" : 333,
  "valueCurrency" : "EUR"
}, {
  "paymentLimitType" : "ECOMMERCE_DAY",
  "value" : 2569,
  "valueCurrency" : "EUR"
}, {
  "paymentLimitType" : "ATM_DAY",
  "value" : 1073,
  "valueCurrency" : "EUR"
} ]
```

Metoda, URL, verzia protkolu

Nepovinná hlavička

Rôzne technické info, napr. identifikátor správy

Typ obsahu, kódovanie

Telo správy (limity na kart)

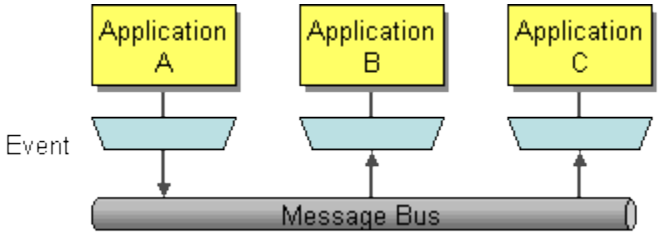
Linkovanie

```
{
  "href" : "http://localhost:8080/rest-api-examples/tickets/2",
  "id" : "2",
  "name" : "Porusenie bezpecnosti",
  "status" : "N",
  "dateCreated" : "2015-03-16T11:16:45.85Z",
  "createdBy" : {
    "href" : "http://localhost:8080/rest-api-examples/users/pgr"
  }
}
```

3c. REST-ové webové služby

- Výhody
  - Analogické ako pri SOAP WS:
    - Integrácia heterogénnych systémov, konzistencia dát, spoľahlivosť, bezpečnosť
  - Plus jednoduchosť v porovnaní so SOAP WS
- Nevýhody
  - Stále pretrváva úzka väzba medzi systémami
    - Väčšia odolnosť voči zmenám (napr. pridanie atribútu)
    - Je možné aj asynchrónne spracovanie HTTP dotazu – napr. jeho uloženie do JMS a jeho následné spracovanie

4. Posielanie správ

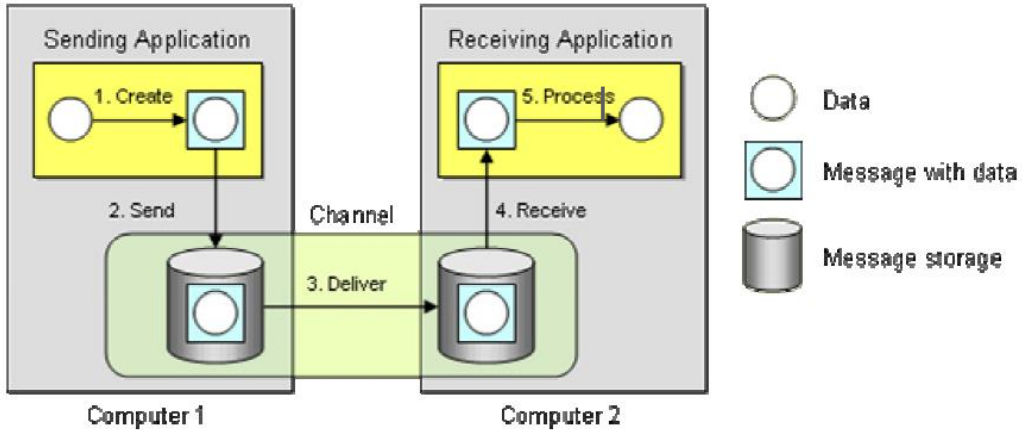


- Podobné, ako posielanie súborov, ale na kvalitatívne vyššej úrovni
  - Prenos „súborov“, presnejšie správ, zabezpečuje Message Bus (Communication backbone)
    - Granularita nie je súbor ale správa (heder, body), prenos a spracovanie sa deje „near online“
    - V Message Bus sa medzi odosielateľom a prijímateľom definuje tzv. jednosmerný kanál/y
    - Cez kanál sa posielajú správy
    - Základné patterny : point-to-point (one-to-one), publish-subscribe (one-to-many)
  - Príklady Message Bus: RabbitMQ (open source), Active MQ (open source), IBM MQ
  - Štandardizované API – napr. Java JMS API

SOFTEC

4. Posielanie správ

- Posielanie správ je vo svojej podstate **asynchrónne**

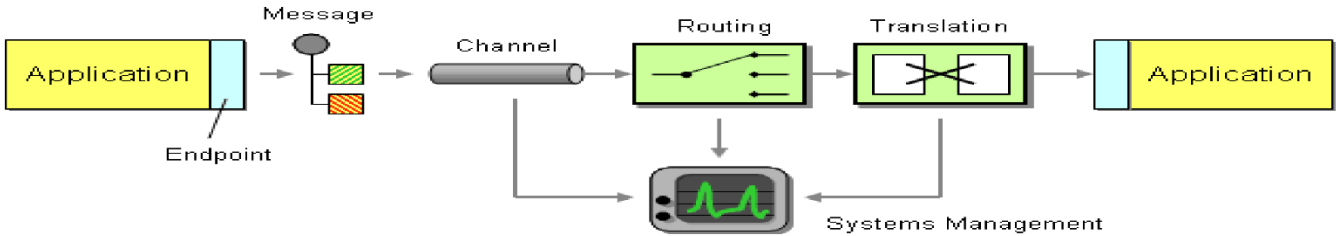


- Princípy:
  - „Odošli a zabudni“
    - Po odoslaní správy odosielateľ na správu „zabudne“ a pokračuje ďalej
  - „Ulož a postúp“
    - Message Bus uloží správu na počítač odosielateľa a zabezpečí jej doručenie na počítač prijímateľa
- Dôležité vlastnosti pre návrh vysoko spoľahlivej komunikácie
  - Zaručené doručenie (exactly one, at least one), zaručené poradie správ, tranzakčnosť, timeout
- Message Bus je možné použiť aj na synchrónnu komunikáciu
  - Nasimuluje sa cez 2 opačné kanály s čakaním na odpoveď

SOFTEC

4. Posielanie správ

- Message Oriented Middleware
  (vo svete webových služieb je to ESB)
  - Je zvyčajne nadstavba nad základným Message Bus
    - Navyše oproti strohému MB má sadu adaptérov na rôzne typy služieb (PLSQL, JCA,...)
    - Navyše umožňuje rôzne transformácie správ, smerovania podľa obsahu, jednotné riadenie, a iné
  - Príklady: IBM ACE (predtým IBM Integration Bus, je nadstavba IBM MQ), webMethods, WSO2 ESB, Apache Camel, Mule ESB, ...

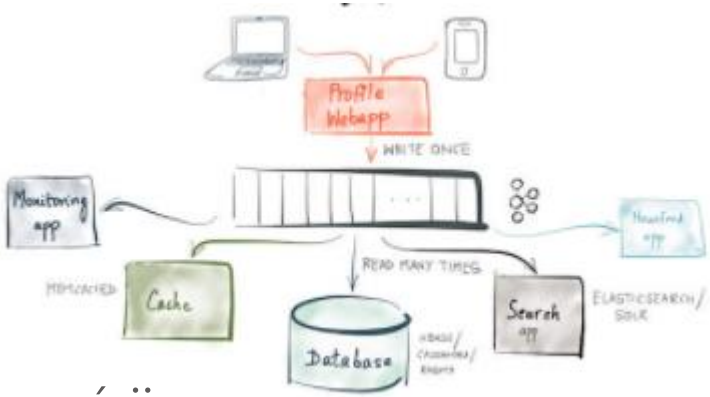


4. Posielanie správ

- Výhody
  - Integrácia heterogénnych systémov
  - Voľná väzba medzi systémami → Vysoká dostupnosť, odolnosť voči preťaženiu, odolnosť voči zmenám
  - Spoľahlivosť – zabezpečuje MOM (napr. podporuje distribuované transakcie, odolnosť voči výpadkom – clustrované riešenia)
  - Veľmi výkonné (aj tisícky správ za sekundu) – majú však limity
  - Lepšia konzistencia dát ako pri dávkovom spracovaní – správy sa posielajú často
  - Bezpečnosť – medzi integrovanými systémami nie je priame http spojenie
- Nevýhody
  - Potrebujeme komplexnú infraštruktúru (message bus, MOM alebo ESB)
  - Komplexnejší vývoj a ladenie systému
  - Nekonzistencia dát, ak doručenie správy trvá dlho
  - Možná strata messages pri publish-subscribe ak nie je konzument pripojený (používa sa durable subscribers, message bus si však potom musí pamäť nedoručené správy – klesá priepustnosť riešenia)
  - Dodatočná réžia oproti RPC (ukladanie správ, prenos, zaručenie doručenia, ...)

5. Event streaming

- Systémy zapisujú udalosti do spoločného „event logu“ - streamu
- Ďalšie systémy (aj tie isté) z tohto „event logu“ udalosti čítajú
  - Každý môže mať svoj smerník na inej pozícii
- Udalosti sú v logu perzistentné
  - Po prečítaní sa nevymazávajú!
- Platformy: Kafka (najrozšírenejšia)
  - Cloudové: Amazon Kinesis, Google Pub/Sub, Azure Event Hubs



SOFTEC

5. Event streaming

- Príklad: Nahradenie Operating Data Store v banke streamingovou platformou
- Výhody:
  - Real-time spracovanie udalostí
  - V porovnaní s posielaním správ:
    - Je bežné spracovávanie udalostí viacerými systémami (pri Posielaní správ je vzor Publish-Subscribe skôr výnimočný)
    - Je možné čítanie aj historických udalostí
    - Výrazne vyššia výkonnosť/priepustnosť
  - Nové architektúry naviazané na streaming (event sourcing)
  - Zachovávajú ostatné výhody z Posielania správ:
    - Integrácia heterogénnych systémov, Voľná väzba a Dostupnosť, Spoľahlivosť, Bezpečnosť, Výkonnosť, Konzistencia dát
- Nevýhody:
  - Nová paradigma,
  - Komplexnejšie riešenia
  - Komplexná infraštruktúra
- Detailnejšie na osobitnej prednáške

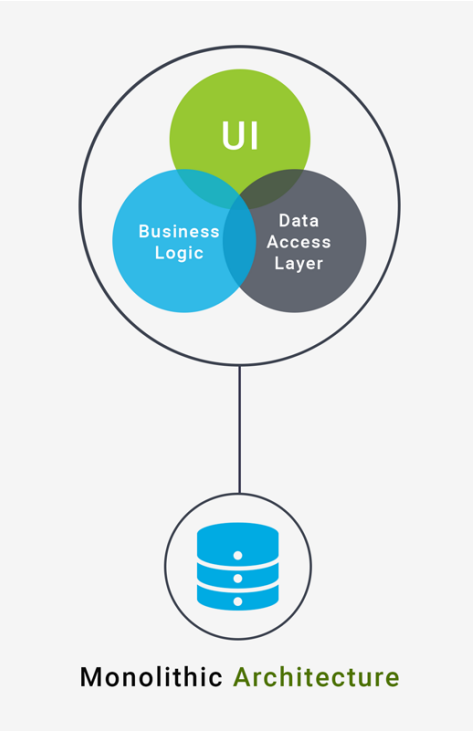
Integračná Architektúra

Integračná architektúra

- Zložky architektúry komplexných Inf. Sytémov
  - Business (procesy, business funkcie, ...)
  - Dátová (životný cyklus entít, ich štruktúra, dátové toky)
  - **Integračná** (aplikácie a ich väzby)
  - iné napr. fyzická architektúra, vývojová, prevádzková
- Spôsoby integrácie sa vyvíjali v čase

Monolitická architektúra

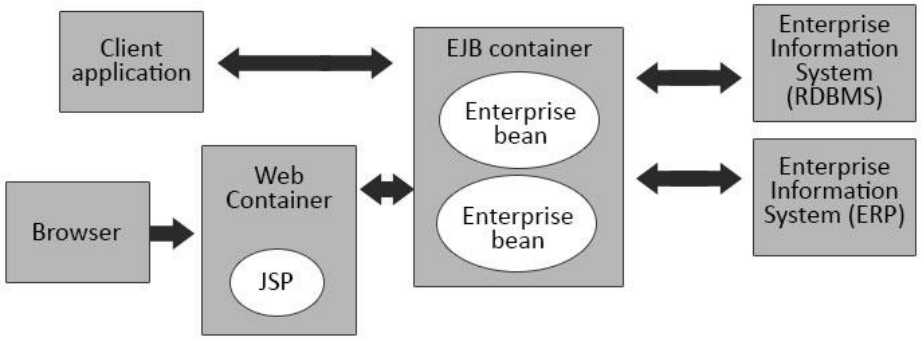
- Systém nie je vnútorne členený do nezávislých elementov
  - Aj keď používa funkcie/podfunkcie, resp. triedy
  - Často špagetový kód
- S externými systémami komunikuje prostredníctvom súborov
- Príklad: Core banking system v banke



SOFTEC

Komponentová architektúra

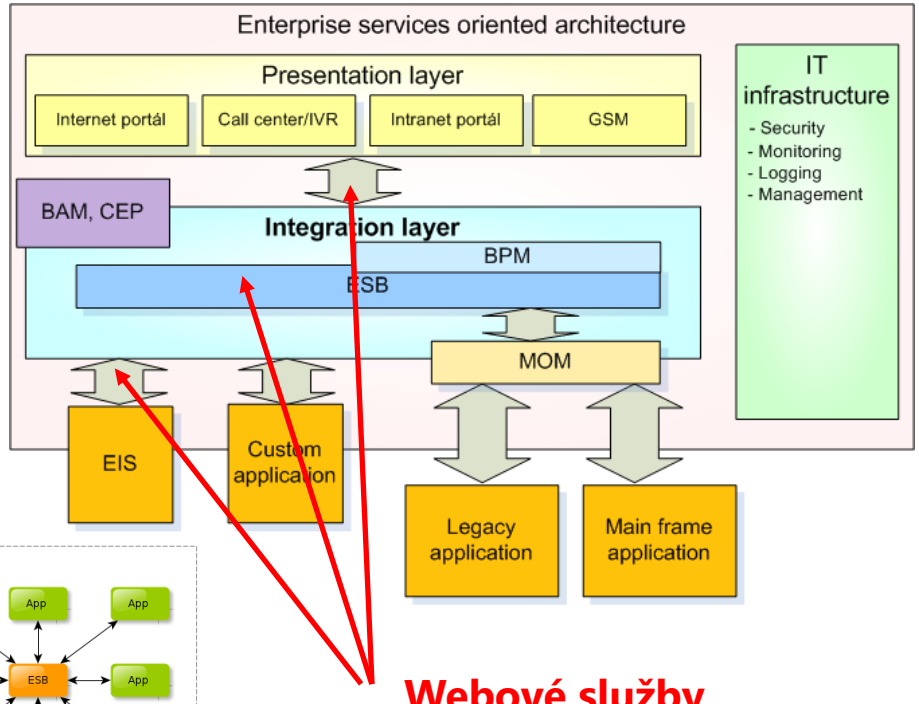
- Pokus členiť systém na vymeniteľné komponenty
  - DCOM, EJB
- RPC pattern
- Platformovo závislé
- Dnes na ústupe
  - na úrovni integrácie medzi aplikáciami sa nepoužíva
  - na úrovni realizácie jednotlivých služieb sa ešte používa

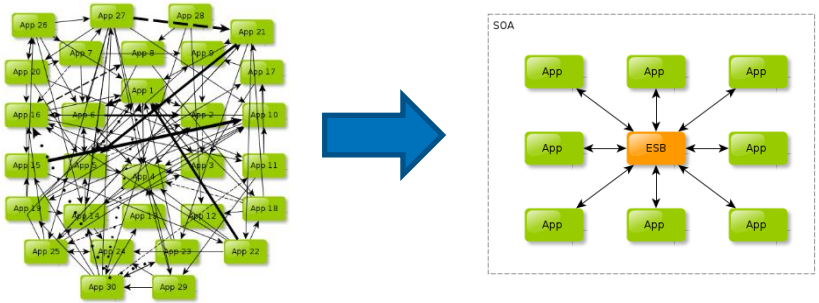


SOFTEC

Servisne orientovaná architektúra (SOA)

- Virtualizácia služieb
  - Prístupné na jednom mieste v štandardizovanej forme – webové služby
- Použitie SOAP-ových správ a Zbernice služieb (ESB)
  - Pôvodne aj Register služieb
- Pôvodne bol zámer aj vnútorne členiť architektúru systému podľa služieb
- V praxi sa nepodarilo
  - Službami sa de facto iba obalili monolity
- Nie je už najmodernejšia architektúra, ale stále veľmi rozšírená (banky, telco) a ešte dlho sa nepodarí nahradiť

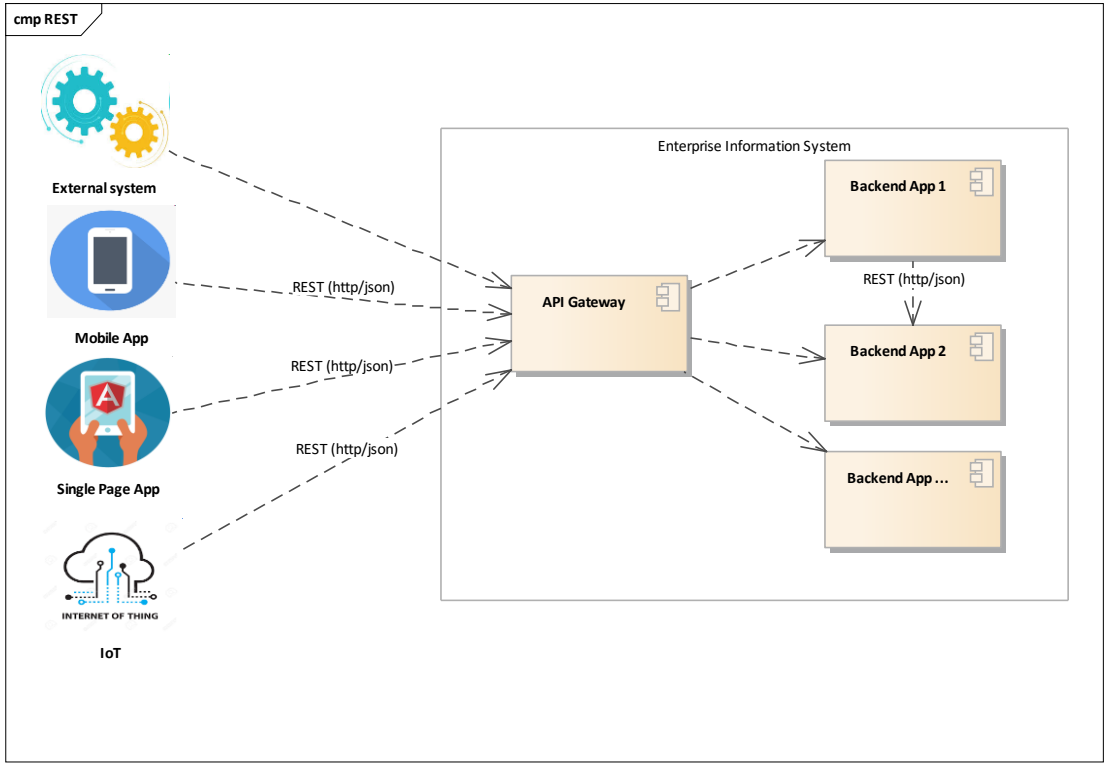




SOFTEC

Zdrojovo orientovaná archit. (ROA)

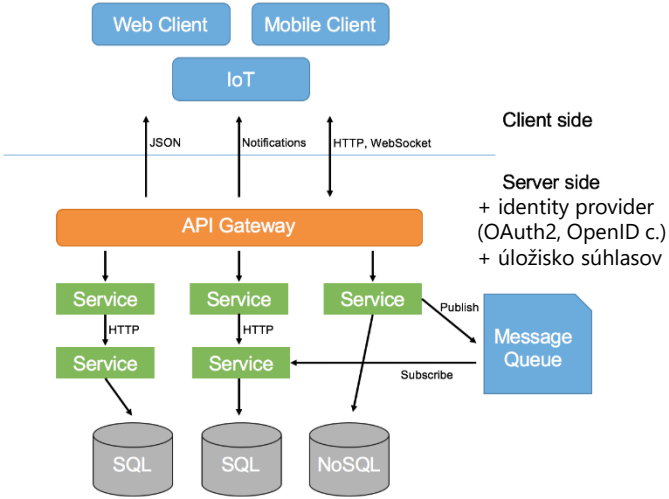
- **Použitie REST služieb**
  - URI/HTTP/JSON/web princípy
- Moderná architektúra využívaná v podnikoch a aj na WEB-e
- API gateway namiesto ESB

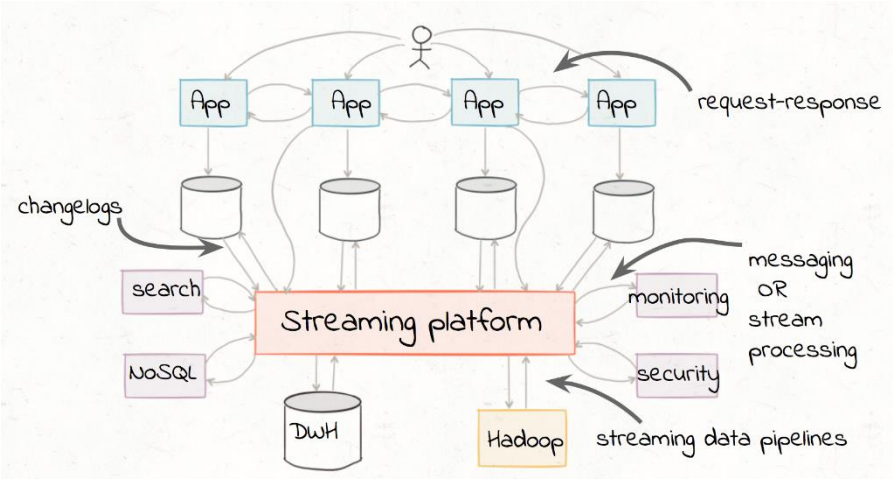


SOFTEC

Mikroslužby

- Dekompozícia aplikácií na menšie samostatné jednotky (bude prednáška)
- Prvej generácie používali REST
- V súčasnosti postupný postup k streamingu





SOFTEC

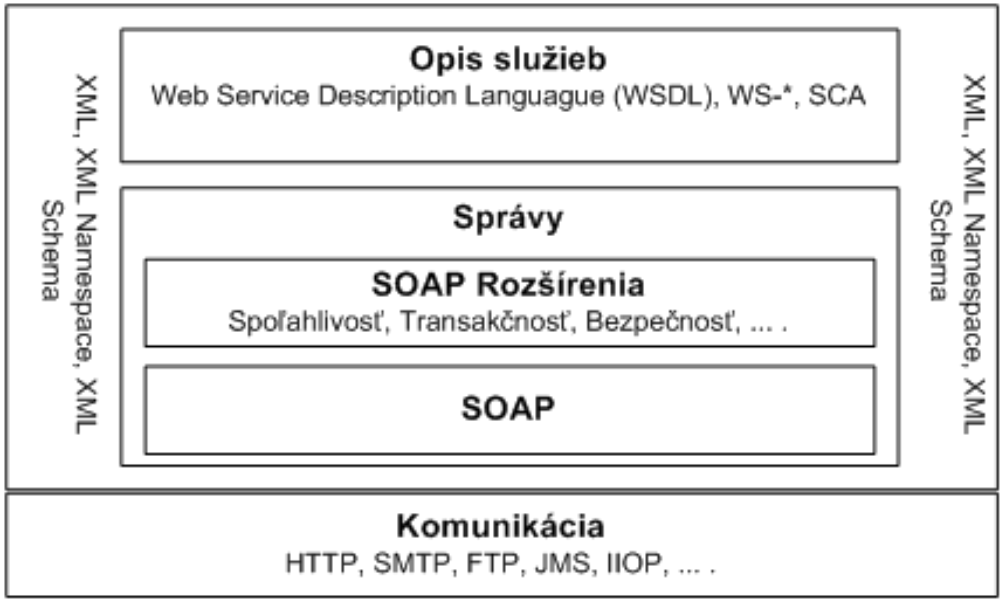
SOA

Vybrané technické detaily

Základné štandardy SOA

SOA sa líši od predchádzajúcich integračných architektúr tým, že je postavená na široko používaných štandardoch:

- Základné štandardy: **HTTP a XML**
  - HTTP – jeden zo základných štandardov internetu
  - XML – platformovo nezávislý jazyk, tiež vysoko rozšírený štandard
- **SOAP** (Simple Object Access Protocol) - Protokol pre posielanie XML dokumentov v distribuovaných systémoch
  - Zvyčajne použitím HTTP
- **WSDL** - jazyk na špecifikáciu rozhrania distribuovaných komponentov nazývaných webové služby



SOFTEC

SOAP

- SOAP (opakovanie)- protokol pre posielanie štruktúrovaných informácií vo formáte XML v distribuovanom prostredí
- Hlavné charakteristiky:
  - Platformová nezávislosť – od programovacieho jazyka, od operačného systému a platformy
    - Definuje štruktúru správy
  - Protokolová neutrálnosť – na transport môže byť použitý ľubovoľný protokol
    - Definuje spôsob, akým sa správa „naviaže“ na prenosový protokol a ako môžu byť využité špecifické vlastnosti protokolu
    - Okrem typického HTTP, napr. aj SMTP, FTP, JMS (je asynchrónne) a ďalšie
  - Rozšíriteľnosť – môže byť ďalej rozšírený nadstavbovými štandardmi
    - Napr. pre bezpečnosť, adresnosť, transakčnosť a ďalšími

Request

HTTP

Response

```
POST /AccountServices HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8;
SOAPAction:http://xxx/AccountServices#GetAccountBalance
...
<?xml version="1.0"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV=http://schemas.xmlsoap.org/soap/envelope/

xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"/>
    < SOAP-ENV:Header>
        <wsa:MessageID>
            uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
        </wsa:MessageID>
    </ SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m:GetAccountBalance xmlns:m="Some-URI">
            <m:accountNumber>1234567890</m:accountNumber>
        </m:GetAccountBalance>
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

SOAP

Ak sa použije WS addressing

Vráť zostatok účtu č. ....

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset=utf-8
...
<?xml version="1.0"?>
<SOAP-ENV:Envelope
    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
    < SOAP-ENV:Header>
        <wsa:MessageID>
            uuid:23A9FC40-CA47-1067-B31D-00DD0AADE31
        </wsa:MessageID>
        <wsa:RelatesTo>
            uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
        </wsa:RelatesTo>
    </ SOAP-ENV:Header>
    <SOAP-ENV:Body>
        <m: GetAccountBalanceResponse xmlns:m="Some-URI">
            <Balance>11234,56</Balance>
            <Currency>EUR</ Currency >
        </m: GetAccountBalanceResponse >
    </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Zostatok účtu je ....

Technická špecifikácia WS: WSD

- WSD (angl. Web Service Description) – opis / špecifikácia rozhrania webovej služby
  - Strojovo spracovateľný opis
    - Je možné z neho generovať kód
  - Vyjadrený v jazyky WSDL
    - WSDL je založený na XML
    - Na popis štruktúry dát je použitá XSD (XML Schema Deinition)

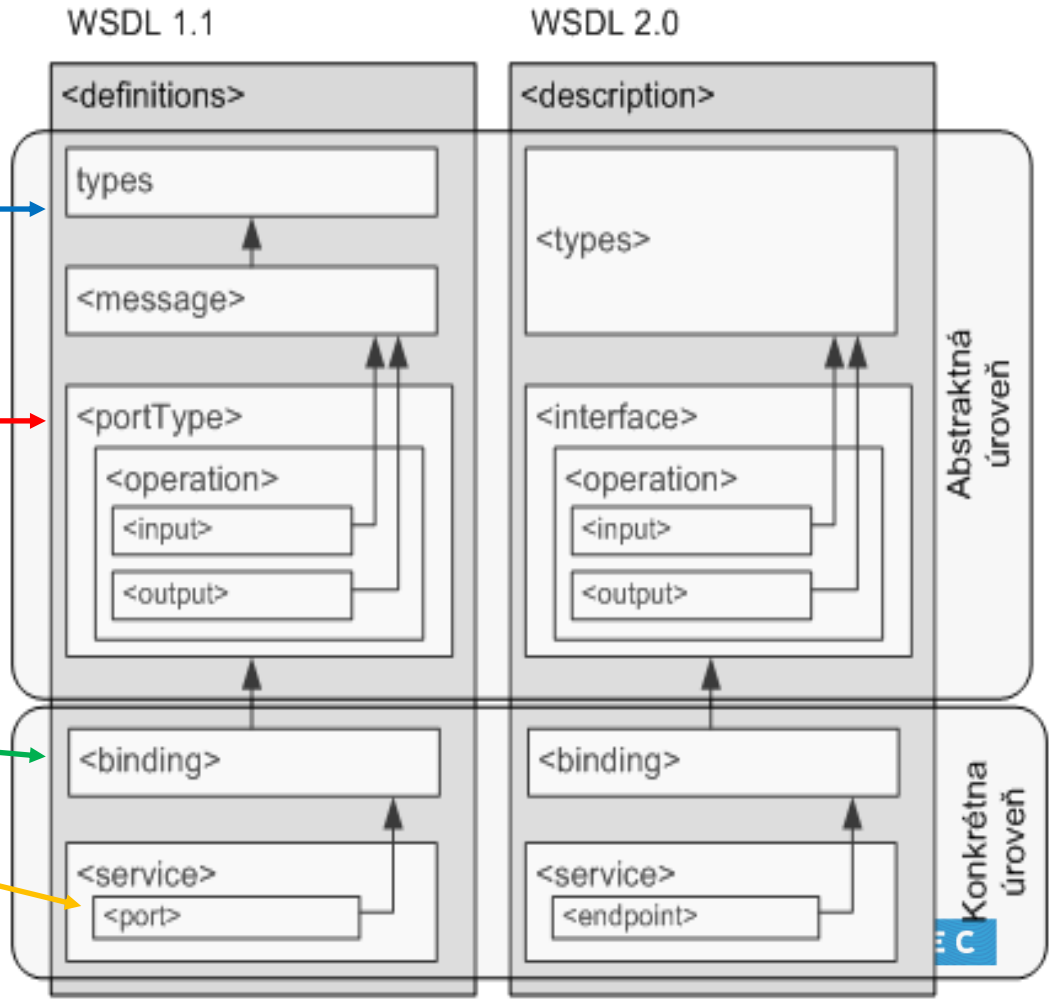
Štruktúra WSD

Definícia štruktúry dát

Definícia operácií

Definície: Prenosový protokol, serializácia, MEP ...

Definícia koncového bodu



Definícia štruktúry dát

- Použitím XSD (XML Schema)
- Best practices (dedenie, verziovanie, vytváranie typov , ...) – doporučujem naštudovať
  - Nie všetky konštrukcie v XSD Schema vie Java spracovať

```
<complexType name="AccountBalanceType">
 <sequence>
  <element name="Balance" type="decimal"   minOccurs="1" maxOccurs ="1"/>
  <element name="Currency" type="string" minOccurs="1" maxOccurs ="1"/>
 </sequence>
</complexType>

<element name="GetAccountBalanceResponse" type="AccountBalanceType" >
```

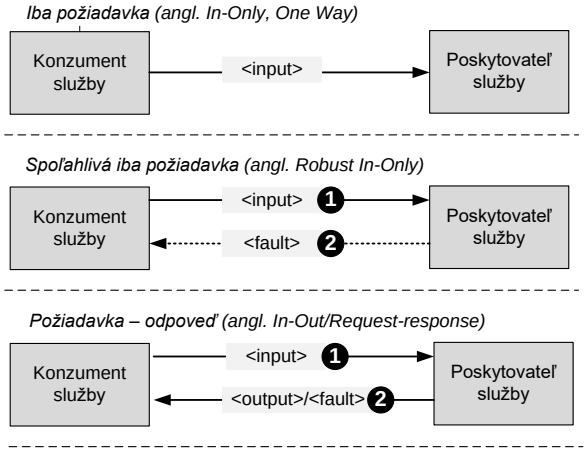
Definícia typu

Definícia elementu príslušného typu

SOFTEC

Definícia operácií

- Interfejs / Port type – logické zoskupenie operácií
- Na operáciu sa môžeme pozerať z pohľadu
  - Remote Procedure Call
    - Vstup, výstupy z volania alebo chyba
  - Spôsobu výmeny správ
    - WSD Message Exchange Patterns (MEP)



WSDL 2.0:

```
<operation name="GetAccountBalance" pattern="http://www.w3.org/ns/wsdl/in-out">
  <input messageLabel="In" element="ab:GetAccountBalanceRequest"/>
  <output messageLabel="Out" element="ab:GetAccountBalanceResponse"/>
</operation>
```

Message Exchange Pattern

Odkaz na štruktúru dát

WSDL 1.1:

```
<message name="GetAccountBalanceResponse">
  <part name="body" element="ab:GetAccountBalanceResponse "/>
</wsdl:message>
```


SOFTEC

Naviazanie (binding)

- Vo všeobecnosti definuje ako sa logická správa mapuje do fyzickej správy:
  - Transportný protokol (HTTP, JMS, SMTP, ...)
  - Štýl naviazania
    - RPC - metóda a jej parametre
    - Document – XML správa (predpokladá sa použitie SOAPAction)
  - Spôsob serializácie objektov
    - podľa SOAP (tzv. Encoded) alebo podľa XSD (tzv. Literal)

Naviazanie

WSDL 2.0:

```
<binding name="accountBalanceSOAPBinding"
          interface="tns:accountBalanceInterface"
          type="http://www.w3.org/ns/wsdl/soap"
          wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/">
        <operation ref="tns:GetAccountBalance "
          wsoap:action="http:// acountservices.example.org/getAccountBalance"
          wsoap:mep="http://www.w3.org/2003/05/soap/mep/request-response" />
</binding>
```

WSDL 1.1

```
<wsdl:binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
   <soap:binding style="document“ transport="http://schemas.xmlsoap.org/soap/http"/>
   <wsdl:operation name="GetLastTradePrice“>
      <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
      <wsdl:input>
         <soap:body use="literal"/>
      </wsdl:input>
     <wsdl:output>
       <soap:body use="literal"/>
     </wsdl:output>
   </wsdl:operation>
</wsdl:binding>
```


SOFTEC

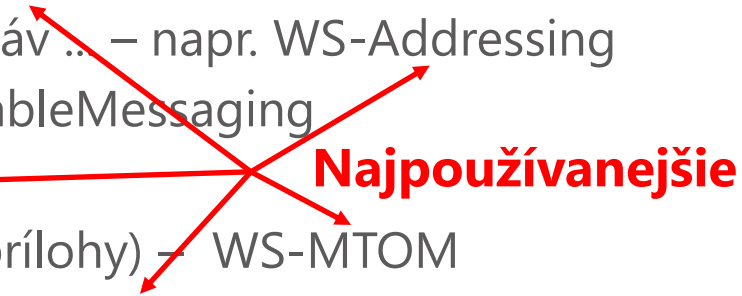
Koncový bod

- Koncový bod (angl. Port/Endpoint) definuje adresu, respektívne miesto pripojenia k danému fyzickému rozhraniu:
  - HTTP - URL,
  - JMS - JMS URI (definícia frontu správ),
  - PL/SQL - JDBC connection string,
  - ...
- Služba – logické zoskupenie interfejsov podľa koncových bodov.

```
<service name="accountBalanceService" interface="tns:accountBalanceInterface">
   <endpoint name="SOAPHTTP" binding="tns:accountBalanceSOAPBinding"
      address="http://example.com:8080/accountBalance“/>
   <endpoint name="JMS" binding="tns:accountBalanceSOAPJMSBinding"
      address="jms:jndi:myQueues/accountBalance"/>
</service>
```

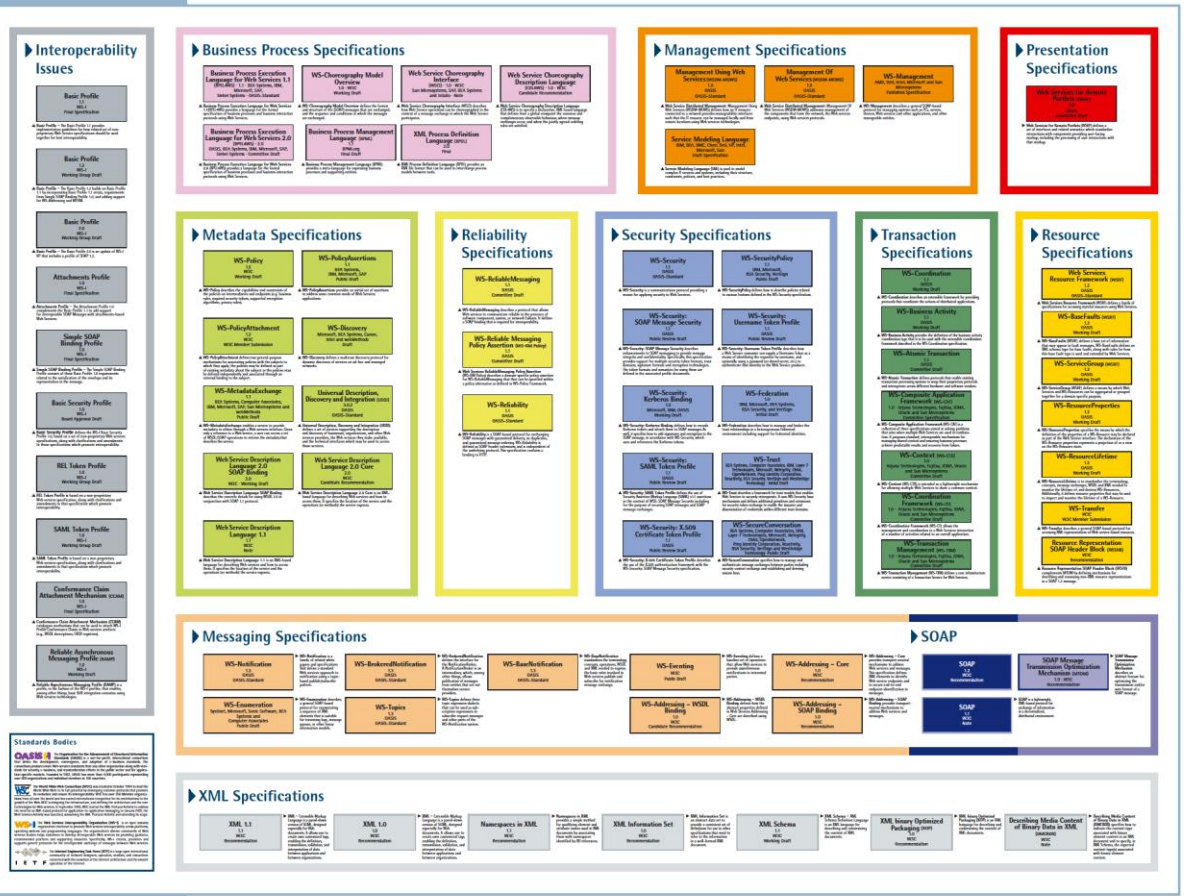
Rozširujúce štandardy WS-*

- Veľké množstvo štandardov, ktoré pokrývajú rôzne oblasti:
  - Interoperabilita (I) – SOAP nebol v tomto smere dosť presný, rôzne systémy si ho vysvetľovali rôzne
    - Viacero štandardov, základný je WS-I Basic Profile
  - Adresovanie zdroja, cieľa, korelácia správ … – napr. WS-Addressing
  - Spoľahlivosť prenosu – napr. WS- ReliableMessaging
  - Bezpečnosť – napr. WS-Security
  - Optimalizácia prenosu (napr. binárne prílohy) – WS-MTOM
  - Spôsob dopĺňania metadát do WSD – WS-Policy
  - Transakčnosť – napr. WS- Atomic Transactions
  - Riadenie a monitoring – napr. WS-Management



SOFTEC

Web Services Standards Overview



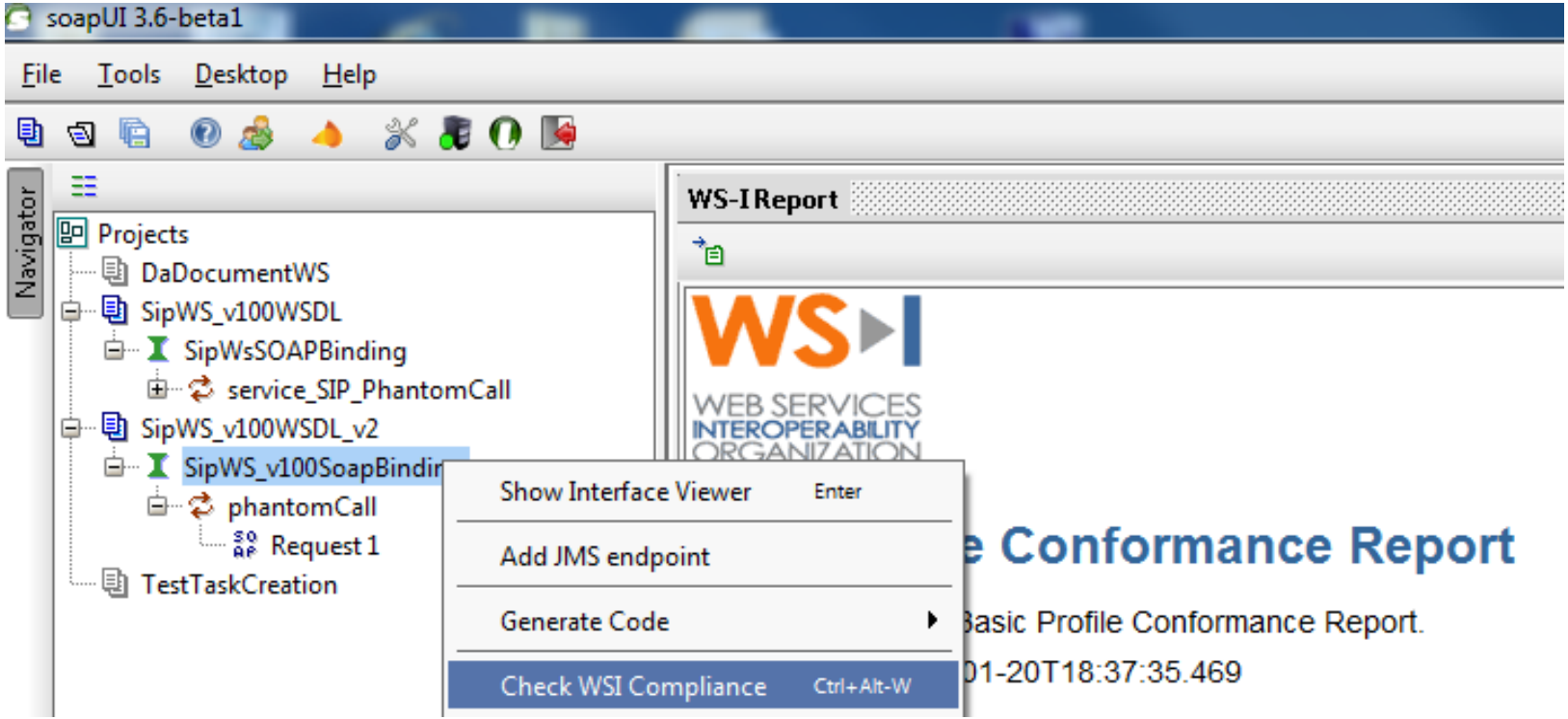
Version 3.0 - February 2007

innoQ Deutschland GmbH
Halskestraße 17
D-40880 Ratingen
Phone: +49 21 02 . 77 162 - 100
info@innoq.com - www.innoq.com

innoQ Schweiz GmbH
Gewerbestrasse 11
CH-6330 Cham
Phone +41 41 743 01 11

SOFTEC

WS-Interoperability Test tool



SOFTEC

JAX-WS

- JEE štandard definujúci Java API na Webové služby
  - **J**ava **A**PI for **X**ML **W**eb **S**ervices
- Prístupy k tvorbe WS
  - Java kód –> WSD
    - Je rýchle a málo prácne
    - Ale malá kontrola nad štruktúrou WSD a málo odolné voči zmenám
  - WSD -> Java kód
    - Je potrebné vytvoriť transformačnú vrstvu (z WSD sa vygenerujú osobitné triedy)
    - Prácnejšie
    - Ale plná kontrola nad štruktúrou WSD a odolnejšie voči zmenám
- Implementácie
  - Java JDK (Simple), Apache Axis, Apache CXF, Metro (Sun - Oracle)

```java
import java.math.BigDecimal;

@WebService
//@SOAPBinding(style = Style.RPC)
@SOAPBinding(style = Style.DOCUMENT,  parameterStyle=ParameterStyle.BARE)
public class AccountService {

    private Map<Integer, Account> accounts;

    public AccountService() {
        accounts = new HashMap<Integer, Account>();
        accounts.put(1000000001, new Account(1000000001, "3141.59"));
        accounts.put(1000000002, new Account(1000000002, "-20.13"));
        accounts.put(1000000003, new Account(1000000001, "0"));
    }

    @WebMethod(action = "getAccountBalance")
    public BigDecimal getAccountBalance( @WebParam(name = "accountNumber")  int accountNum
        Account account = accounts.get(accountNumber);
        if (account != null) {
            return account.getBalance();
        } else {
            throw new NoSuchAccountException(accountNumber);
        }
    }
```

ROA a REST detailnejšie

Rest Oriented Architecture a RESTFull web services

SOFTEC

Nastáva ÉRA verejného/cloud API

- Mobil – brána do sveta business funkčností
  - Aplikácie + **API**
  - Za API je schovaná reálna funkčnosť
- Kompozitné aplikácie
  - Integrujú API z viacerých zdrojov
    - Overenie Google (OAuth 2), Hry Google, Reklama, Google drive, Paypal , YouTube, Open Bank API ...
    - Cloud - Software as a service – nové obchodné modely
  - SDK alebo úroveň API
  - Príklad RaiPay – kartové API + NFC payments, vernostný systém, vouchers, security zariadenia, business monitoring – analytika, google maps, ...
- Omni channel architektúra
  - Obchodný prípad môže „prechádzať" viacerými kanálmi (aplikáciami)
- IoT

SOFTEC

Resorce Oriented Architecture

- V súčasnosti najviac používaná architektúra internetového API je založená na princípoch REST
- REST - Representational State Transfer
  - **Štýl softvérovej architektúry** pre distribuované systémy
  - **Resource** – softvérový komponent (reprezentovaný dátami a/alebo službou)
  - **HTTP protokol** (a sieťová infraštruktúra) – API na prácu s Resource, zabezpečuje interoperabilitu
- RESTFull služby
  - Postavené na princípe REST
  - Ustálené použitie JSON ako formátu dát
  - **Štandardizácia** – protokol, formát dát, volaní

SOFTEC

Využitie HTTP

- **Identifikácia zdrojov pomocou URI**
  - URI stačí aby som vedel získať zdroj (opak ID – čka)
- **Manipulácia so zdrojom pomocou HTTP metód** (GET, POST, PUT, PATCH, DELETE)
- **Využitie metadát** (samo popisné správy)
  - HTTP hlavička (súčasť HTTP štandardov)
    - content type negotiation,
    - languague negotiation,
    - compression,
    - charset,
    - Security (HTTPS, autorizačné schémy)

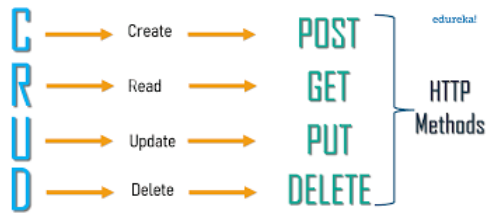
- **Security** (OAuth2, Open ID, JSON web tokens, ...)

- **Bezstavovovosť**
  - Stav na klientovi (nákupný košík), alebo stav je reprezentovaný zdrojom
  - Prelinkovanie
- **Cachovanie**
- **Viacvrstvový systém** (client, firewall, webcache, gateway, load balancer, ...)
  - Využitie týchto komponentov

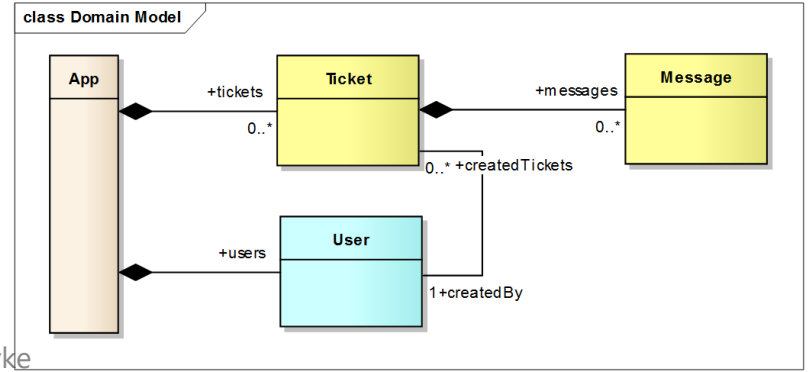
- Využitie nadstavieb nad HTTP
  - **Security** (OAuth2, Open ID Connect, JSON web tokens, ...)

54

Zdrojovo orientované API



- **URI**
  - Identifikácia zdroja
  - Zdroj – dáta ale aj metóda
- **CRUD+** – základná množina operácií nad zdrojom
  - **GET /tickets** – Načítanie zoznamu požiadaviek
  - **GET /tickets/12** – Načítanie konkrétnej požiadavky
  - **POST /tickets** – vytvorenie požiadavky
  - **PUT /tickets/12** – update požiadavky #12
  - **PATCH /tickets/12** – čiastočný update požiadavky #12
  - **DELETE /tickets/12** – zrušenie požiadavky #12
  - **POST /tickets/12/fork** – volanie metódy fork na požiadavke #12
  - **GET /tickets?severity=high** – Načítanie zoznamu požiadaviek s vysokou prioritou
  - **PATCH /tickets/12/messages/5** - čiastočný update message #5 na požiadavke #12



PUT versus PATCH versus POST

- PUT/PATCH – idempotent – update zdroja
  - Môže byť zopakovaný viac krát
  - Request obsahuje ID zdroja
- POST – vytvorenie zdroja
- Pri vytvorení http code 201
- Odpoveď môže obsahovať zdroj

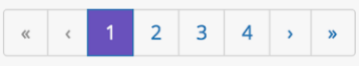
55

SOFTEC

GET URL - Stránkovanie a filtrovanie

Stránkovanie

- Dôležité ak chceme budovať štandardizované komponenty
- Request obsahuje query parameters s informáciou o stránke
  - offset
  - Limit
- ../tickets?offset=50&limit=25
- Response
  - by mal obsahovať metadata - aktuálna stránka, celkový počet, aktuálna veľkosť stránky, prvá, predošlá, nasledujúca, posledná
  - Viacero možností
    - V http hlavičke (hlavička Link - RFC 5988, X-Total-Count)
    - V obálke kolekcie



Filtrovanie

- Najčastejšie ako URL parametre na kolekcii resources
  - GET /tickets?status=open
  - GET /tickets?sort=status,dateCreated
- Aliasy na často využívané dotazy
  - GET /tickets/recently_closed

Expandovanie

- GET /tickets/12?embed=messages,createdBy.name
- Môže viesť ku komplexnej implementácii

Problémy:

- Nie je štandardizované (len na úrovni best practise)
- Každý zrealizuje ináč
- Náročné na implementáciu
- Odpoveď

GraphQL

SOFTEC

JSON – formát dát pre internet

```
[ {
  "href" : "http://localhost:9080/nemesis-rest/subjectcases/21964",
  "id" : 21964,
  "strategy" : "ACCEPTANCE",
  "state" : null,
  "office" : null,
  "currentCount" : 0,
  "noAnswerCallCount" : 0,
  "invalidCallCount" : 0,
  "dates" : {
    "skluc1" : "2015-04-07T10:10:51.202Z"
  },
  "texts" : {
    "skluc2" : "shodnota2",
    "skluc1" : "shodnota1"
  },
  "actions" : [ {
    "id" : 382280,
```

Data types:

- a string
- a number
- an object (JSON object)
- an array
- a boolean
- *null*

57

SOFTEC

XML versus JSON

XML

```
<empinfo>
   <employees>
      <employee>
         <name>James Kirk</name>
         <age>40></age>
      </employee>
      <employee>
         <name>Jean-Luc Picard</name>
         <age>45</age>
      </employee>
      <employee>
         <name>Wesley Crusher</name>
         <age>27</age>
      </employee>
   </employees>
</empinfo>
```

JSON

```
{  "empinfo" :
     {
          "employees" :  [
          {
               "name" : "James Kirk",
               "age" : 40,
          },
          {
               "name" : "Jean-Luc Picard",
               "age" : 45,
          },
          {
               "name" : "Wesley Crusher",
               "age" : 27,
          }
                              ]
     }
}
```

Jednoduchosť použitia JSON

```
.service('pcSrv', ['$http', '$log', function ($http, $log) {
    var basket = {};
    var pcSrv = this;
    $log.debug('Creating service ...');


    this.refreshBasket = function (clbk) {
        $log.debug('Basket to refresh ' + angular.toJson(basket));
        $http.post("http://localhost:8088/pc/offer/compute", basket).success(function (data) {
            basket = data;
            saveState();
            clbk(basket);
        });
    };
```

Content-Type negotiation

- Client
  - **Accept**: application/json
- Server
  - **Content-Type**: application/json
- Media Type
  - Špecifikácia formátu + parsovacie pravidlá (detailnejší popis formátu)
    - application/json;v=1
    - application/problem-json
    - application/hal-json
- Iné spôsoby
  - http://..../subjectcases/21964**.json**

60

Linkovanie

1. Súvisí so stavom na klientovi
   1. HATEOAS = Hypermedia as the Engine of Application State
   2. Na začiatku by mi mala „teoreticky“ stačiť jedna URI linka (bookmark)

2. Prechádzanie referencií
   1. Zdroje sú navzájom previazené

```
{
  "href" : "http://localhost:8080/rest-api-examples/tickets/2",
  "id" : "2",
  "name" : "Porusenie bezpecnosti",
  "status" : "N",
  "dateCreated" : "2015-03-16T11:16:45.85Z",
  "createdBy" : {
    "href" : "http://localhost:8080/rest-api-examples/users/pgr"
  }
}
```

- Expandovanie liniek
  - V query parametri povieme, ktoré referencie chceme expandovať (vyhneme sa tak nadbytočnému HTTP dotazu)
  - Napr. http://localhost:8080/rest-api-examples/tickets/2?expand=createdBy

61

SOFTEC

Štandardizácia linkovania

- HAL - Hypertext Application Language (application/hal+json)
  - Stav (JSON data)
  - Linky
  - Embedded Resources
- Iné štandardy
  - collection-json
  - Siren
  - hm-json

62

SOFTEC

Verziovanie API

- REST API je odolné voči „kompatibilným“ zmenám
  - Pridanie fieldu, referencie,
  - Pridanie encodingu, ...
- Verzia by mala byť súčasťou URL
  - Používaný prístup
    - URL major version (štrukturálne zmeny)
    - HTTP header – minor verzia (kompatibilná zmena)
  - GET https://api.sandbox.paypal.com/**v1**/payments/payment
- Iné prístupy
  - Verzia v hlavičke : Content-Type : application/json;v=1.0
- Dokumentácia zmeny (je treba dať vedieť dopredu aká zmena a kedy sa chystá)

63

SOFTEC

Chyby

- Dva účely – ladenie aplikácie, riadenie UI (zobrazenie hlášky, ...)
- Developer API Experience
  - Samo popisné, s dostatkom informácii o príčine chyby
    - Typ – štruktúra chyby
    - Unikátne číslo chyby/timestamp – väzba na aplikačný log
    - Kód chyby – väzba na detailné info a lokalizovaný message
    - Správa pre používateľa – ak nie je kód chyby
    - Technický popis chyby – informácia pre developera
    - Detailnejšie parametre chyby– napr. pre potrebu zahrnutia info do lokalizovaného message
    - URL na detailný popis chyby


```
{
    "timestamp": "2021-11-05T13:44:30.011",
    "type": "/problems/cws-error",
    "status": 500,
    "message": "Server error",
    "path": "/api/v1/card-designs/121",
    "errorCode": 50000000,
    "errorCodeString": "SERVER_ERROR",
    "action": "NONE",
    "errorParams": []
}
```



```
{
  "error": {
    "code": 401,
    "message": "Request is missing required authentication c
    https://developers.google.com/identity/sign-in/web/devco
    "errors": [
      {
        "message": "Login Required.",
        "domain": "global",
        "reason": "required",
        "location": "Authorization",
        "locationType": "header"
      }
    ],
    "status": "UNAUTHENTICATED",
    "details": [
      {
        "@type": "type.googleapis.com/google.rpc.ErrorInf
        "reason": "CREDENTIALS_MISSING",
        "domain": "googleapis.com",
        "metadata": {
          "service": "admin.googleapis.com",
          "method": "ccc.hosted.frontend.directory.v1.I
        }
      }
    ]
  }
}
```


https://google.github.io/styleguide/jsoncstyleguide.xml

RFC 7807: Problem Details for HTTP APIs - **application/problem+json**

SOFTEC

Chyby v HTTP

- V zhode s HTTP kódmi
  - 4xx – klientské/business chyby
    - nezotaviteľné – vždy vrátia chybu, napr. validačné chyby, klient musí upraviť request
    - Mali by mať vždy JSON payload s popisom chyby
  - 5xx – serverovské chyby
    - zotaviteľné – po oprave na strane serveru je možné request zopakovať

- Príklady chýb
  - 400 Bad Request - The request is malformed, such as if the body does not parse
  - 404 Not Found - When a non-existent resource is requested
  - 405 Method Not Allowed - When an HTTP method is being requested that isn't allowed for the authenticated user
  - 410 Gone - Indicates that the resource at this end point is no longer available. Useful as a blanket response for old API versions
  - 415 Unsupported Media Type - If incorrect content type was provided as part of the request
  - 422 Unprocessable Entity - Used for validation errors

65

Caching

- Použitie hlavičiek
  - cache-control
    - private, public
    - no-store – neukladať
    - no-cache – revalidate resource (napr. za pomoci ETag)
    - max-age – dĺžka platnosti cache (v sekundách)
    - s-maxage – max age pre „shared cache" – napr. CDNs
  - cache-control má prioritu pred expires, pragma
- Entity tag (ETag)
  - Server (iniciálna odpoveď)
    - ETag:"a090bca78978a788d"
  - Client (následný dotaz)
    - If-None-Match:"a090bca78978a788d"
  - Server (odpoveď)
    - 304 Not Modified
- Last modified
  - Server (iniciálna odpoveď)
    - Last-Modified: Tue, 15 Nov 1994 12:45:26 GMT
  - Client (následný dotaz)
    - If-Modified-Since:Tue, 15 Nov 1994 12:45:26 GMT
  - Server (odpoveď)
    - 304 Not Modified

cache-control directive	may I cache locally?	may I cache anywhere?	should revalidate, even being fresh?
no-store	no	no	n/a
private	yes	no	no
no-cache	yes	yes	yes
public	yes	yes	no

66

Cross site requests

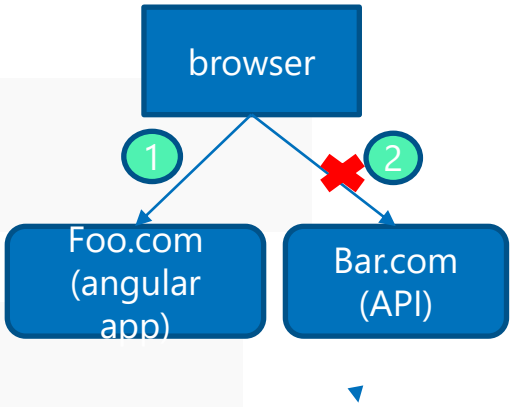
- Potrebujeme cross site request ak chceme robiť mashup, browser však obsahuje security obmedzenia – dotazy môžu ísť len na tú istú doménu
- Cross-origin resource sharing (CORS)
  - Riadenie security v browser (security implementovaná v browser)
  - Odolnosť voči niektorým security útokom

2a Request

```
OPTIONS /
Host: bar.com
Origin: http://foo.com
```

Response

```
Access-Control-Allow-Origin: http://foo.com
Access-Control-Allow-Methods: PUT, DELETE
```



67

SOFTEC

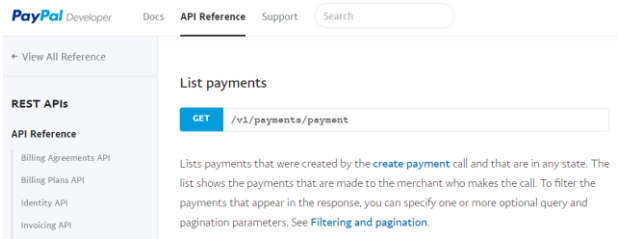
Popisovanie rozhrani

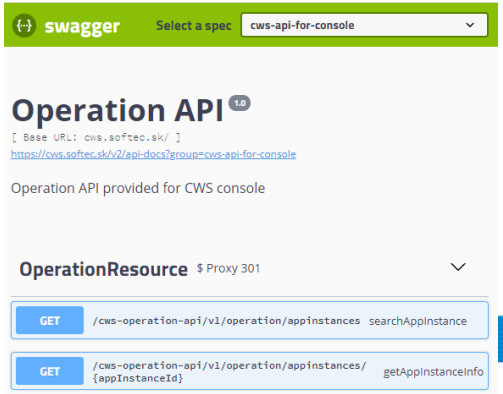
Open API (https://www.openapis.org/)

```yaml
openapi: 3.0.3
info:
  title: CWS wallet API
  version: 1.3.0
  description: |
    `Overview`

    Definition of operations for interacting with wallet (and re

    `Operations`:
    * externalActivation
    * walletTokens

tags:
  - name: wallets
    description: Wallet resources and operations
paths:
  /cws-internal-api/v1/app-instance/external-activation:
    post:
      tags:
        - wallets
      operationId: externalWalletActivation
      summary: External wallet activation
      requestBody:
        $ref: '#/components/requestBodies/ExternalActivation'
      responses:
        "200":
          description: Wallet has been activated
        "400":
          $ref: '#/components/responses/ValidationResponse'
```

```yaml
    ExternalActivation:
      description: Information for external wallet activation
      properties:
        bankId:
          $ref: '#/components/schemas/NetworkBankCode'
        applicationInstanceId:
          description: Device identifier
          type: string
          maxLength: 32
        mbaCustomerId:
          description: Mobile banking customer identifier
          type: string
          maxLength: 64
        mbaCustomerIdType:
          description: Mobile banking customer id type
          type: string
          maxLength: 64
```

- SWAGGER
- Custom
  - Popis, príklady
  - RestDoc





Testovanie API

- AT/UT - FW support
  - Postman
  - SOAP UI (využitie Open API)
  - Spring RestTemplate
  - **Rest-Assured**

```
given().
        param("key1", "value1").
        param("key2", "value2").
when().
        post("/somewhere").
then().
        body(containsString("OK"));
```

```
given().auth().basic(username, password).when().get("/secured").then().statusCode(200);
```


69

JAX-RS

- Java API for RESTfull Web Service
- Podobne ako JAX WS
- Implementácie
  - Apache CXF
  - Jersey

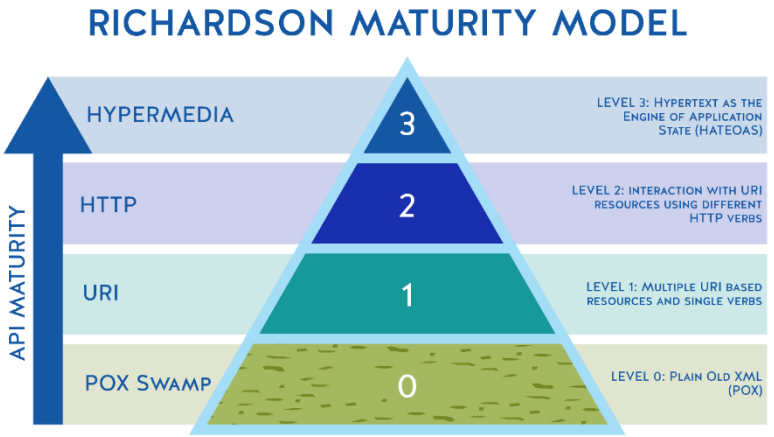
- Osobitná implementácia v Spring Web

Ukážka anotácii

```
package my.bank.testrs;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Response;

@Path("/hello")
public class HelloWorld {

    @GET
    @Path("/echo/{input}")
    @Produces("text/plain")
    public String ping(@PathParam("input") String input) {
        return "Echo ...." + input;
    }

    @POST
    @Produces("application/json")
    @Consumes("application/json")
    @Path("/jsonBean")
    public Response modifyJson(JsonBean input) {
                        input.setVal2(input.getVal1());
                        return Response.ok().entity(input).build();
    }
}
```

Ako dobre navrhnúť API

- Best practicies, examples
- Učenie od druhých
- Využitie HTTP
- Pozor na komplexnosť
  - riadenie API môže obsahovať veľa možností a kombinácii (triedenie, stránkovanie, expandovanie, podpora jsonp, formátovanie, ...)
    - Riešenie : napr. schovať zložitosť za ? (ako súčasť query string-u)
    - zvážiť použitie graphQL



SOFTEC

Prečo sa REST presadil

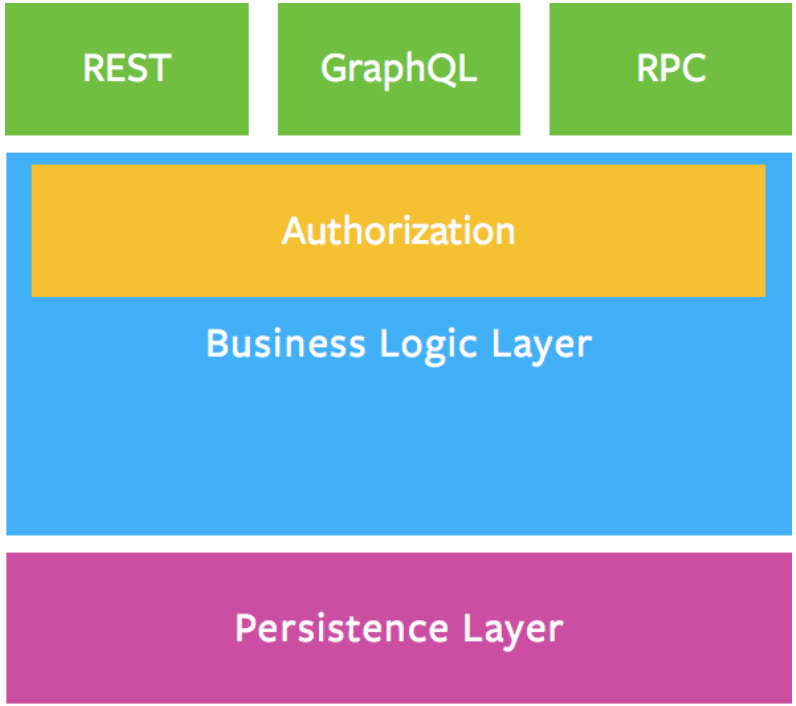
- Vlastnosti
  - Jednoduchosť
  - Pochopiteľnosť
  - Dostupnosť
  - Vyspelosť (HTTP)
  - Podporuje JS

- Prirodzený pre internet
  - Rozvoj s použitím SPA aplikácii (nar. Angular)
- Využiteľný v podnikoch
- Základ pre distribuovanú architektúru Mikroservisov

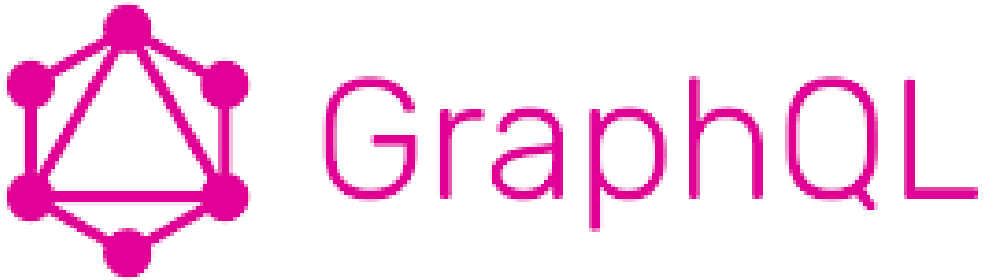
73

Alternatívy k REST API

- REST – klasika
- GraphQL – alternatíva k REST
- gRPC – optimalizovaný prenos



SOFTEC


GraphQL

GraphQL - princípy

- Server – čo viem poskytnúť
- Client – čo z toho potrebuje
- Striktne typový
- Zloženie popisu - schémy
  - Typový systém a rozšírenia
  - Operácie
    - Query, Mutation, Subscription
- Popis v Schema Definition Language (SDL)
  - Dokážeme ju dotazovať cez queryAPI
- Abstrakcia od HTTP mapovania

```
# post author
type Author {
    id: ID!
    name: String!
    thumbnail: String
    posts: [Post]!
}

# user
type User {
    id: ID!
    name: String!
    login: String
}


# The Root Query for the application
type Query {
    recentPosts(page: Int, size: Int): [Post]!
    filteredPosts(title: String, page: Int, size: Int): [Post!]!
    authors:[Author!]!
    post(id: ID!): Post!
}
```

```
query FILTERED_POSTS {
  filteredPosts(title: "ti", page: 0, size: 2){
    id, title, commentsCount,  commentsCount, author{name,thumbnail}
  }
}
```

```
{
  "data": {
    "filteredPosts": [
      {
        "id": "1",
        "title": "title0",
        "commentsCount": 2,
        "author": {
          "name": "Author 0",
          "thumbnail": "thumbnail0"
        }
      },
      {
        "id": "2",
        "title": "title1",
        "commentsCount": 0,
```

GraphQL

GraphQL

Umožňuje zjednodušenie budovania komplexného API

Štandard:

- **Stránkovanie, filtrovanie, expandovanie, selekcia, ...**
- Viac resources v jednom dotaze
- Možný aj update resources
- ...

```
{
  user(id: 4802170) {
    id
    name
    isViewerFriend
    profilePicture(size: 50)  {
      uri
      width
      height
    }
    friendConnection(first: 5) {
      totalCount
      friends {
        id
        name
      }
    }
  }
}
```

```
{
  "data": {
    "user": {
      "id": "4802170",
      "name": "Lee Byron",
      "isViewerFriend": true,
      "profilePicture": {
        "uri": "cdn://pic/4802170/50",
        "width": 50,
        "height": 50
      },
      "friendConnection": {
        "totalCount": 13,
        "friends": [
          {
            "id": "305249",
            "name": "Stephen Schwink"
          },
          {
            "id": "3108935",
            "name": "Nathaniel Roman"
```

GraphQL – výhody oproti REST

Výhody oproti REST

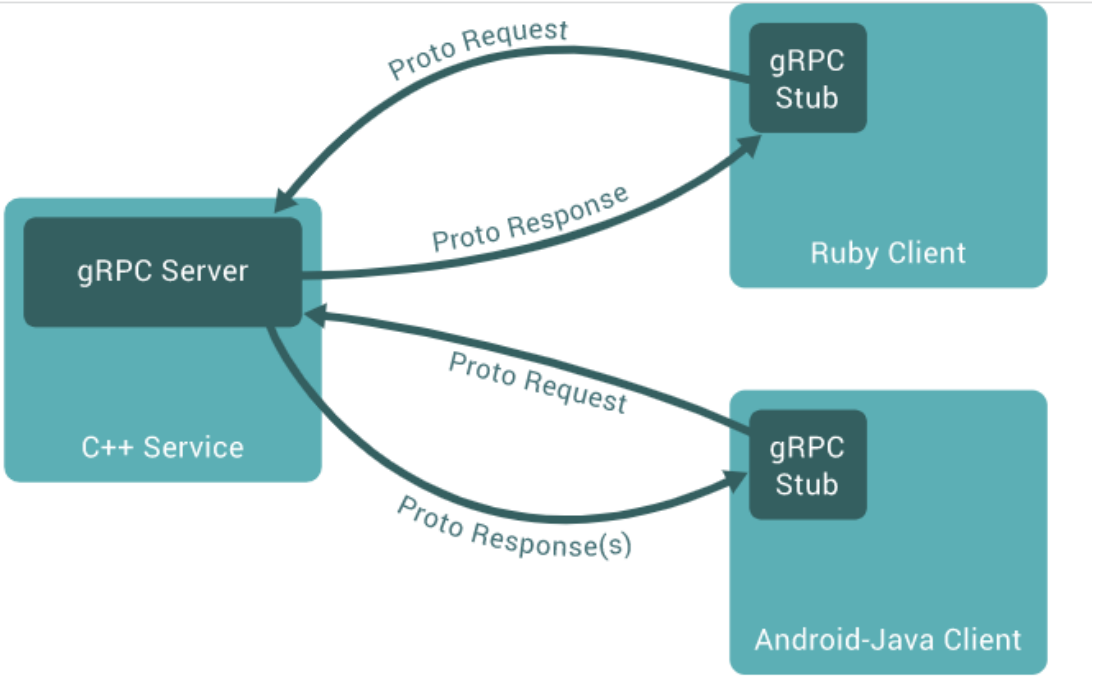
- Jednoznačnosť mapovania na HTTP
  - V REST 5 ľudí jednu požiadavku vie namapovať 6-timi spôsobmi
- Jednoznačnosť a jednoduchosť schémy
  - Jednoduchšia ako OpenAPI
  - Možnosť generovania
- Prenášam len dáta čo potrebujem
  - Neprenášame celé entity
  - Out of the box query - ak chcem expandovať linky, robiť projekciu, ...
- Eliminácia viacnásobného dotazovania
  - Query eliminuje linkovanie
  - Viac query v jednom dotaze
- Podpora streaming-u dát
- Podpora CQRS - Command and Query Responsibility Segregation
  - Rozdelenie na query a mutations


gRPC

gRPC

Klasické RPC

- Vzdialený objekt cez lokálne API
- Podpora vo viacerých jazykoch
- Zabezpečená interoperabilita



SOFTEC

gRPC

- **Založený na Protocol Buffers (Protobuf)**
  - **I**nterface **D**escription **L**anguage
  - Serializácie/Deserializácia
  - Generátory a implementácie v jazykoch
- **gRPC pridáva vlastnosti**
  - Podobné ako „binding“ pri SOA
  - Full protocol stack (http(s) 2.x + protobuf)
  - Podporuje streaming, autentizáciu, timeouty, blocking-not blocking volanie, ...
  - Nevhodný pre browser (slabá podpora)
  - Vhodný na komunikáciu medzi súžbami, mob. aplikácia, IoT

```
// The greeter service definition.
service Greeter {
  // Sends a greeting
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
```

```
// polyline.proto
syntax = "proto2";

message Point {
  required int32 x = 1;
  required int32 y = 2;
  optional string label = 3;
}

message Line {
  required Point start = 1;
  required Point end = 2;
  optional string label = 3;
}

message Polyline {
  repeated Point point = 1;
  optional string label = 2;
}
```

Protobuff

- Field Unique number - jeho ID
- Field Rules
  - singular – povinné pole
  - optional - nepovinné
  - repeated - zoznam
  - map – kľúč/hodnota
- Základné typy (a ich mapovanie do java)
  - int – int32, uint32, sint32, fixed32, sfixed34
  - long – int64, uint64, sint64, fixed64, sfixed64
  - bolean - boolean
  - double - double
  - float - float
  - String – string
  - bytes

- Default values
- Enumerations
  - Default
  - Reserved values
- any
- oneof
- Nested Types
- Packages
- Importing Definitions
- Services
- Options
- JSON Mapping
- Commenty
  - /*....*/

```
syntax = "proto3";

option java_multiple_files = true;
option java_package = "com.example.grpc.stock.api";

import "google/protobuf/timestamp.proto";

enum StockCode {
  ALI = 0;
  WIG = 1;
  CGE = 2;
  ALFA = 3;
  FCBK = 4;
}

enum ActionCode {
  SELL = 0;
  BUY = 1;
}

message StockTradeActionRequest{
  string action_id = 1;
  string userCode = 2;
  StockCode stock =3;
  int64 amount = 4;
  ActionCode action = 5;
}
```

Zdroj

Typy volaní

- Unary RPCs

```
rpc SayHello(HelloRequest) returns (HelloResponse);
```

- Server streaming

```
rpc LotsOfReplies(HelloRequest) returns (stream HelloResponse);
```

- Client streaming

```
rpc LotsOfGreetings(stream HelloRequest) returns (HelloResponse);
```

- Bidirectional streaming

```
rpc BidiHello(stream HelloRequest) returns (stream HelloResponse);
```

Synch/Asynch
Metadata
- Podobné ako headers v http

Channels
- Niečo ako session

Podpora v jazykoch

- Generuje
  - Client stub
    - Synch (blocking)
    - Asynch (not blocking - Future stub / listenable future
    - Channel
  - Server skeleton

```
HelloReply response;
try {
  response = blockingStub.sayHello(request);
} catch (StatusRuntimeException e) {
  logger.log(Level.WARNING, "RPC failed: {0}", e.getStatus());
  return;
}
```

```
private class GreeterImpl extends GreeterGrpc.GreeterImplBase {

  @Override
  public void sayHello(HelloRequest req, StreamObserver<HelloReply> responseObserver) {
    HelloReply reply = HelloReply.newBuilder().setMessage("Hello " + req.getName()).build();
    responseObserver.onNext(reply);
    responseObserver.onCompleted();
  }
```

Java

```
protoc --plugin=protoc-gen-grpc-java \
    --grpc-java_out="$OUTPUT_FILE" --proto_path="$DIR_OF_PROTO_FILE" "$PROTO_FILE"
```

https://grpc.io/docs/languages/java/basics/

Vyhody gRPC

Výhody

- Lepší výkon
- Podpora streaming-u
- Skrytie implementácie (serializácia, využitie HTTP)
- Výhody IDL
  - Silne typový formát
  - Ľahko zrozumiteľné špecifikácie API
  - Polyglot: generujte server/klient v akomkoľvek jazyku
  - Evolúcia schém dozadu/dopredu kompatibilná
- Dobrá dokumentácia
  - https://grpc.io/docs

Nevýhody

- Nie je podpora v prehliadači
- Veľa generovaného kódu
- Spojenie nie je plne state less
  - http 2 - jedno TCP pripojenie s dlhou životnosťou, multiplex
  - Potrebný špeciálny load balancing
- Obmedzené nástroje

gRPC versus Rest

Feature	gRPC	HTTP APIs with JSON
Contract	Required (`.proto`)	Optional (OpenAPI)
Protocol	HTTP/2	HTTP
Payload	Protobuf (small, binary)	JSON (large, human readable)
Prescriptiveness	Strict specification	Loose. Any HTTP is valid.
Streaming	Client, server, bi-directional	Client, server
Browser support	No (requires grpc-web)	Yes
Security	Transport (TLS)	Transport (TLS)
Client code-generation	Yes	OpenAPI + third-party tooling

https://learn.microsoft.com/en-us/aspnet/core/grpc/comparison?view=aspnetcore-7.0

ROA versus SOA

- V konečnom dôsledku robia to isté – sprostredkúvajú dáta a služby – ale na iných princípoch (Services vs. Resources)
- SOA (WS*)
  - je heavy weight (štandardy, nástroje, metodiky, implementácie) – plusy aj mínusy,
  - Komplikovaná
    - Pridáva extra vrstvy abstrakcie - soap, niektoré WS*
    - Duplikuje funkcionalitu SOAP versus HTTP, funcionalitu HTTP versus niektore WS*
  - menej kompatibilná s „internetom“ (rôzny klienti, rôzne technológie, rôzne požiadavky),
  - nevhodná pre súčasné FE technológie (SPA – napr. Angular)
  - na realizáciu potrebujeme podporu knižníc
  - riešenie komplexných problémov – validácia cez schému, podpora dedičnosti, ...
- ROA (REST)
  - je jednoduchá,
  - efektívna,
  - viac kompatibilná s „internetom“
    - JSON, podpora JS, HTTP klient, využitie HTTP štandardu...,
  - jednoducho realizovateľná
  - akceptovateľná aj pre IoT
  - podpora podnikovej integrácie a aj FE aplikácii súčasne

SOFTEC

Trendy v servisnej architektúre

SOFTEC

Trendy v servisnej architektúre

- **Mikroslužby**
  - **Agilita**, otvorenosť (technologie, jazyky, ...), škálovateľnost, odolnosť voči výpadkom, návrh pre zmenu, kontinuálne delivery, jednoduchá prevádzka - **DevOps**
- **Cloud ready aplikácie**

  12 faktorová aplikácia - https://12factor.net/
- **Mikroslužby a Streaming platform**
  - Dáta získavajú na dôležitosti - agilita pri práci s dátami (prechod od ETL, spoločných DB, datových služieb s centrálnym uložiskom k distribuovanej dátovej architektúre, near online dostupnosť dát)

Podrobnejšie v predmete: Vývoj natívnych aplikácií pre cloud

Questions