Architektúra orientovaná na služby a jej technická realizácia pomocou zbernice podnikových služieb

Matej Duháček

Diplomová práca
2017

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2016/2017

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Matej Duháček**
Osobní číslo:	**A15161**
Studijní program:	**N3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Forma studia:	**prezenční**
Téma práce:	**Architektura orientovaná na služby a její technická realizace pomocí Sběrnice podnikových služeb**
Téma anglicky:	**An Architecture Oriented on Services and its Technical Realisation using Company Hub Services**

Zásady pro vypracování:

1. **Zpracujte literární studii zaměřenou na základní principy architektury orientované na služby (SOA) a na integrační standard sběrnice orientovaný na služby (ESB).**
2. **Seznamte se s integrační platformou Red Hat JBoss Fuse s její komponentami a s vývojovým prostředím JBoss developer studio.**
3. **Proveďte instalaci a nastavení virtuálního prostředí pro realizaci podnikové sběrnice.**
4. **Vytvořte návrh podnikové sběrnice pomocí integrační platformy JBoss Fuse a následně ji implementujte do vytvořeného virtuálního prostředí.**
5. **Otestujte funkčnost a integraci podnikové sběrnice pomocí vzorově vytvořených aplikací.**

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. **Meeraj KUNNUMPURATH. JBoss 3.2 Deployment and Administration. 1. Apress, 2004. ISBN 978-1-4302-0819-8.**
2. **JBoss Fuse tutorials point: simply easy learning [online]. (1) [cit. 2017-01-23]. Dostupné z: https://www.tutorialspoint.com/jboss_fuse/jboss_fuse_tutorial.pdf**
3. **ERL, Thomas. SOA: Servisně orientovaná architektura. Computer Press, 2009. ISBN 9788025118863.**
4. **JBoss fuse documentation. Redhat.com [online]. [cit. 2017-01-23]. Dostupné z: https://access.redhat.com/search/#/documentation?q=&documentation_product=Red%20Hat%20JBoss%20Fuse&language=en**
5. **IBSEN, Claus a Jonathan ANSTEY. Camel in action. 2010. ISBN 9781935182368.**
6. **SMYDER, Bruce, Dejan BOSANAC a Rob DAVIES. ActiveMQ in Action. 2011. ISBN 9781933988948.**
7. **SMYDER, Bruce, Dejan BOSANAC a Rob DAVIES. Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions. 1. ISBN 978-0321200686.**

Vedoucí diplomové práce:	**Ing. Milan Navrátil, Ph.D.** Ústav elektroniky a měření
Konzultant:	**Ing. Filip Kovář** Ústav informatiky a umělé inteligence
Datum zadání diplomové práce:	**3. února 2017**
Termín odevzdání diplomové práce:	**16. května 2017**

Ve Zlíně dne 3. února 2017

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Mgr. Roman Jašek, Ph.D.
ředitel ústavu

Matej, Duháček:

Architektura orientovaná na služby a její technická realizace pomocí Sběrnice podnikových služeb

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové/bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 15. 5 2017

.....
podpis diplomanta

ABSTRAKT

Diplomová práca je zameraná na návrh podnikovej zbernici služieb ako implementáciu architektúry orientovanej na služby. Práca sa skladá z 6 častí. V prvej časti je popísaná servisne orientovaná architektúra čo tento pojem predstavuje a aký má význam v integračných systémoch. V druhej časti je popísaná podniková zbernica ako implementácia servisne orientovanej architektúry a jej uplatnenie v porovnaní s inými integračnými systémami. V tretej časti sa venujem integračnej platforme JBoss Fuse a jej komponentom ktorú využívam aj s komponentami pri navrhovaný podnikovej zbernici služieb. V ďalšej časti popisujem postup pri vytváraní virtuálne prostredia, na ktoré bola nasadená podniková zbernica. V piatej časti sa venujem návrhu integračného riešenia podnikovej zbernice pričom vysvetľujem funkcionalitu riešenia. V poslednej časti diplomovej práce popisujem testovaciu aplikáciu a následne testujem navrhnutú podnikovú zbernicu.

Kľúčové slová : Služba, Podniková Zbernica služieb, JBoss Fuse, ActiveMQ,

ABSTRACT

This diploma thesis proposes enterprise service bus as implamention of architecture focused on services. Thesis contains of 6 chapters. First part presents theory of architecture focused on service and its meaning within integrated systems. Second chapter analyse enterprise bus as an implementation of architecture focused on services and compare its usage with another types of integrated systems. In the third part I target the Jboss Fuse as integration platform and its components as I use this platform in my proposal of enterprise service bus. Next chapter describes the process of creating virtual environmnet on which the enterprise bus has been implemented. Fifth chapter provides the proposal of integrated solution for enterprise bus and describes funcionalities of the solution. Last part of this thesis analyses the tested application and tests the enterprise bus itself.

Key words: Service, Enterprise bus, Jboss Fuse, ActiveMQ

Týmto by som rád poďakoval vedúcemu mojej diplomovej práce, Ing. Milan Navrátil Ph.D a tiež odbornému konzultantovi Ing. Filip Kovář za pomoc pri spracovaní diplomovej práce v podobe odborného vedenia, cenných rád, pripomienok a konzultácií počas spracovávania mojej práce.

Prehlasujem, že odovzdaná verzia diplomovej práce a verzia elektronická nahraná do IS/STAGE sú totožné.

OBSAH

ÚVOD

Pri vývoje stolových či webových aplikácií bolo zvykom vyvíjať pre určitú platformu, pričom vývojári boli špecializovaní na svoj programovací jazyk, v ktorom bola aplikácia realizovaná. V súčasnej dobe je však situácia iná. Zmenili sa požiadavky na hranice vyvíjaných systémov. Pokiaľ chceme, aby sa aplikácia uplatnila na trhu, je potrebné, aby bola schopná komunikovať medzi rôznymi verziami či platformami. Takisto sa zmenili tiež požiadavky na vývojárov, pričom už nestačí, aby sa vývojár špecializoval na jednu technológiu. Dôvod, prečo tieto zmeny nastali, je snaha o kompletné pokrytie business procesov informačnými systémami, čo je možné dosiahnuť len kombináciou viacerých technológií. Mimo iné aj tieto aspekty zapríčinili prudký rozvoj servisne orientovaného prístupu.

Pre zrealizovanie servisne orientovanej architektúry v prostredí informačných technológií, existujú rôzne softwarové prostriedky. Tieto produkty vytvárajú niečo ako zbernicu, kde sú nasadené a ovládané služby smerujúce z rôznych aplikácií, pričom tieto aplikácie môžu byť zrealizované na rozličných platformách. Jedným z takýchto produktov je práve použitý JBoss Fuse.

Cieľom mojej diplomovej práce je zoznámiť sa s princípmi servisne orientovanej architektúry a navrhnúť jej realizáciu pomocou podnikovej zbernici služieb JBoss Fuse. Pričom návrh zahrňuje prepojenie rôznych platforiem a externých API. Na záver práce bude funkčnosť návrhu otestovaná pomocou vzorovej aplikácie, ktorá bude predstavovať pre testovacie účely jednoduchý internetový obchod.

I. TEORETICKÁ ČASŤ

1 SERVISNE ORIENTOVANEJ ARCHITEKTÚRA

Termín servisne orientovaný je všeobecne známy a používaný. Je spomínaný v rôznych konceptoch pre rôzne účely. Základná myšlienka je však taká, že logiku nevyhnutnú pre riešenie daného problému môžeme efektívnejšie vytvoriť, uskutočniť a riadiť, pokiaľ ju rozdelíme na kolekciu menších vzájomne súvisiacich častí problému.

1.1 Servisne orientovaná architektúra - definícia

Servisne orientovaná architektúra (alebo tiež SOA) predstavuje architektonický štýl a spôsob, ktorým môže byť informačný systém založený. Tento systém je založený na službách.

Konkrétna implementácia SOA predstavuje univerzálne a jednotné prostredie, ktoré je schopné prepojiť rôzne informačné systémy a IT služby. Výhoda tohto systému spočíva v zabezpečení prostredia, ktoré sa vyznačuje neustálymi zmenami. V takomto prostredí dokáže zabezpečiť riadenie daného systému a jeho stabilitu.

1.2 Služby servisne orientovanej architektúry

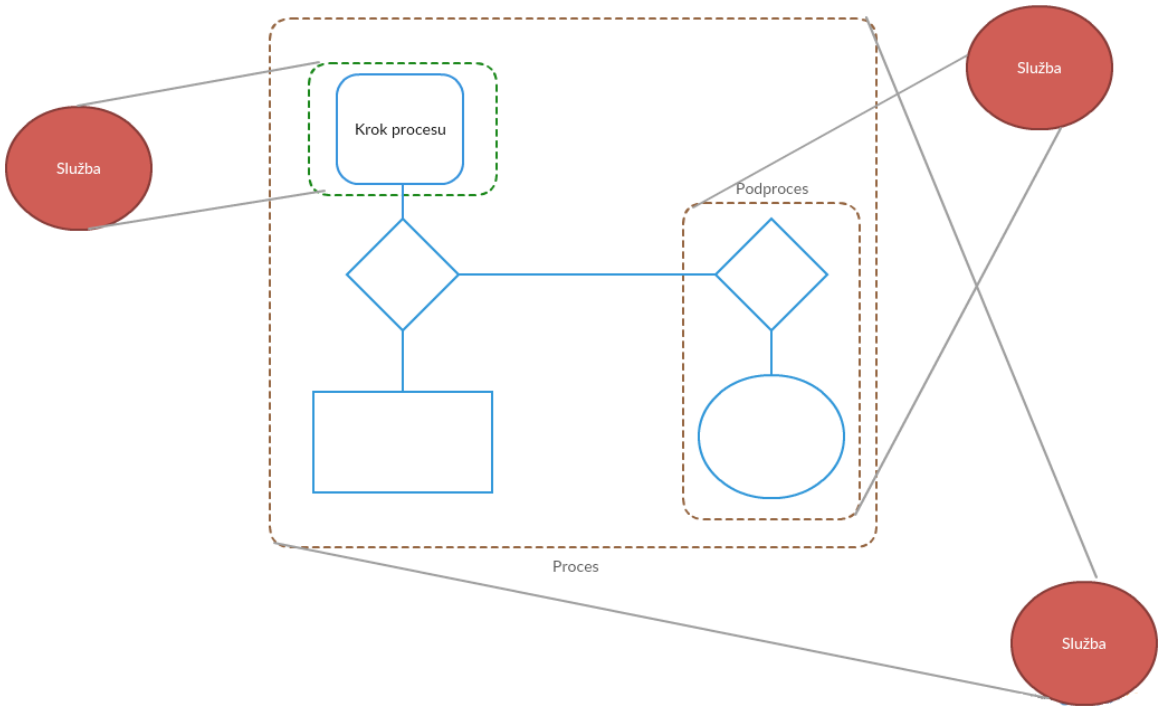
Služba servisne orientovanej architektúry je softwarový prvok, ktorý pomocou svojho definovaného rozhrania ponúka jednu alebo viac funkcionalít. Služby podporujú vzájomnú spoluprácu, sú rozmanité a autonómne. Rozhranie služby popisuje všetky možné druhy požiadaviek zo svojho okolia, ktoré je daná služba schopná spracovať a tiež definuje to ako je služba schopná odpovedať na tieto požiadavky. [2]

1.2.1 Služby – zapuzdrenie

Aby služby boli schopné udržať si svoju nezávislosť, je potrebné, aby zapuzdrovali svoju logiku v rôznych kontextoch. Každý kontext je odlišný v závislosti k riadiacej úlohe, entite riadenia alebo logickému zoskupeniu.

Každá služba môže zapuzdrovať úlohu, ktorá je vykonávaná v jednom kroku alebo v podprocesu zloženého z skupiny krokov. Tieto kroky sa spúšťajú v pred definovanom poradí a to na základe riadiacich pravidiel a podmienok spustenia.

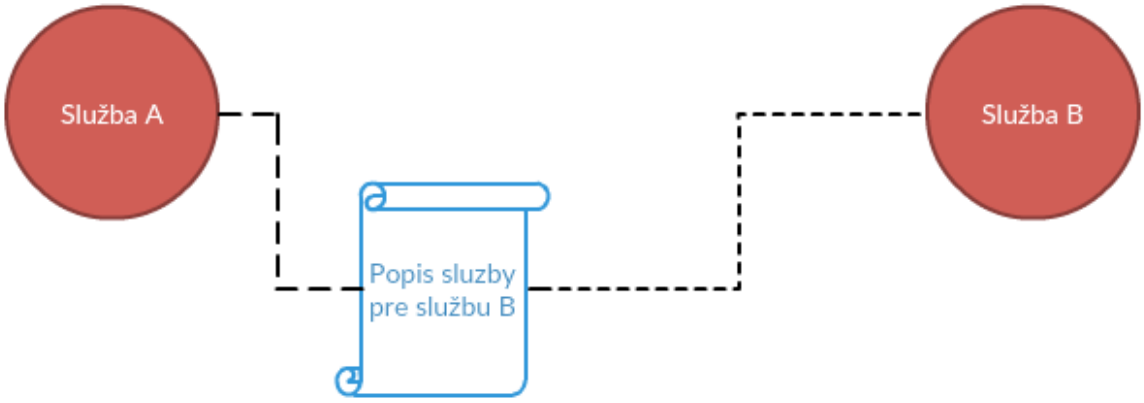
Na Obr.1 je graficky znázornené, že pokiaľ vytvárame riešenie založené na službách, služba nemusí zapuzdrovať iba krok procesu či podproces, ale môže zapuzdrovať celú logiku procesu, pričom pri podprocesoch a pri logike celého procesu môže oblasť reprezentovaná službami zahrňovať logiku, ktorú zapuzdrujú ďalšie služby. K tomu, aby jednotlivé služby boli schopné použiť nimi zapuzdrenú logiku, je nutné, aby sa zúčastnili spúšťania riadiacich aktivít. K tomu je potrebné vytvoriť rôzne vzťahy s tými, ktoré by chceli tieto služby použiť.



Obr. 1 Zapuzdrenie služieb

1.2.2 Vzťahy medzi službami

Ako už bolo vyššie spomínané, služby v servisne orientovanej architektúre môžu používať iné služby, či už pre ich správnu činnosť alebo pre riadenie daného systému. Preto je potrebné, aby tieto služby navzájom o sebe vedeli a tiež aby bola umožnená ich komunikácia. K tomuto účelu slúži *popis* služieb.



Obr. 2 Vzťahy medzi službami

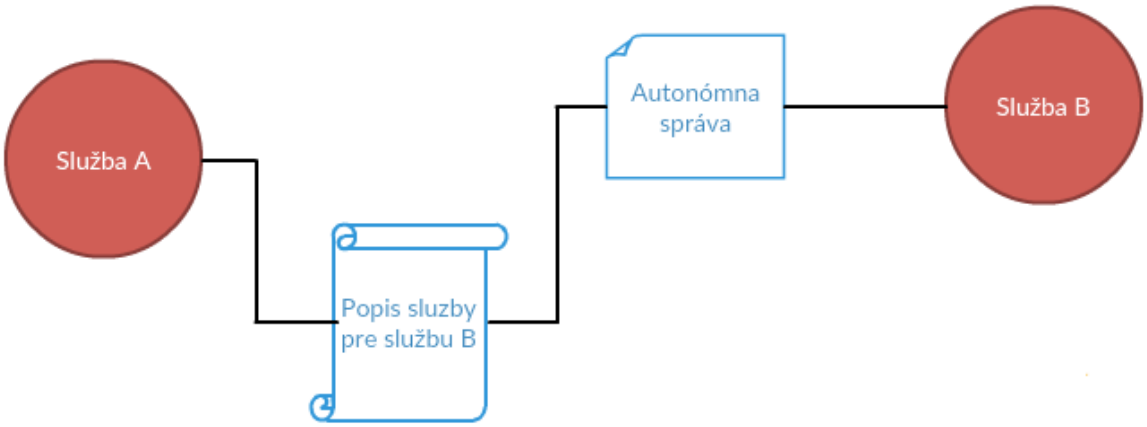
Na Obr. 2 je možné si všimnúť, že vďaka popisu služby má teraz služba B všetky potrebné informácie pre komunikáciu zo službou A.

Hlavná funkcia popisu služby je stanoviť jej názov umiestnenie a tiež požiadavky, ktoré sú potrebné pre výmenu dát. Takáto väzba, ktorá medzi týmito službami vďaka popisu služieb vzniká je klasifikovaná ako voľná väzba. Na Obr. 2 je vidno, že služba A si je vedomá existencie služby B a to prostredníctvom popisu služby k službe B , ktorý služba A vlastní.

1.2.3 Komunikácia medzi službami

Služby medzi sebou vymieňajú informácie a to prostredníctvom komunikačného systému, ktorý by mal byť schopný zachovať medzi nimi ich vzťah voľnej väzby, ktorá je popísaná nižšie.

Akonáhle služba odošle správu, stratí kontrolu nad danou správou, to znamená, že nemá prehlaď, čo sa so správou počas jej cesty stane. Na základe tohto faktu je potrebné, aby správa sama o sebe sa stala nezávislou jednotkou komunikácie, a teda rovnako ako služba, správa sa stane autonómnou. Aby bolo možné toto dosiahnuť, správa je vybavená dočasnou „inteligenciou“. Čo znamená, že sú schopné spravovať svoju časť procesnej logiky.



Obr. 3 Autonómna správa

Služby, ktoré sú schopné komunikovať prostredníctvom správ, tak ako sme si vyššie popísali, tvoria základnú architektúru servisne orientovaného riešenia. Táto architektúra sa môže javiť ako podobná už používaným architektúram, ktoré rovnako podporujú výmenu správ a oddeľujú rozhranie od procesnej logiky. Čo však odlišuje túto architektúru, je to ako sú navrhnuté tri komponenty jadra a to služby popisy a správy.

1.3 Webové služby

V predošlej kapitole bola rozvinutá téma zahrňujúca pojem servisných služieb, a tiež to, ako sa zapuzdrujú rôzne časti logiky. V reálnom svete, pokiaľ chceme zaviesť služby do automatizačných riešení, je potrebné zabezpečiť technológiu podporujúcu základnú servisnú orientáciu, ktorá je dôležitou súčasťou pri implementácií funkcií riadenia.

Najbližšie k splneniu vyššie spomínaných požiadaviek za predpokladu správneho navrhnutia majú práve webové služby. A to prevažne z dôvodu flexibilných a adaptívnych vlastností týchto služieb.

Webovú službu môžeme definovať ako základný stavebný blok pre prechod distribuovanej výpočtovej techniky na internete. Tvoria prostredie, ktoré umožňuje komunikáciu medzi aplikáciami a tým sprostredkovávajú integráciu aplikácií.[8]

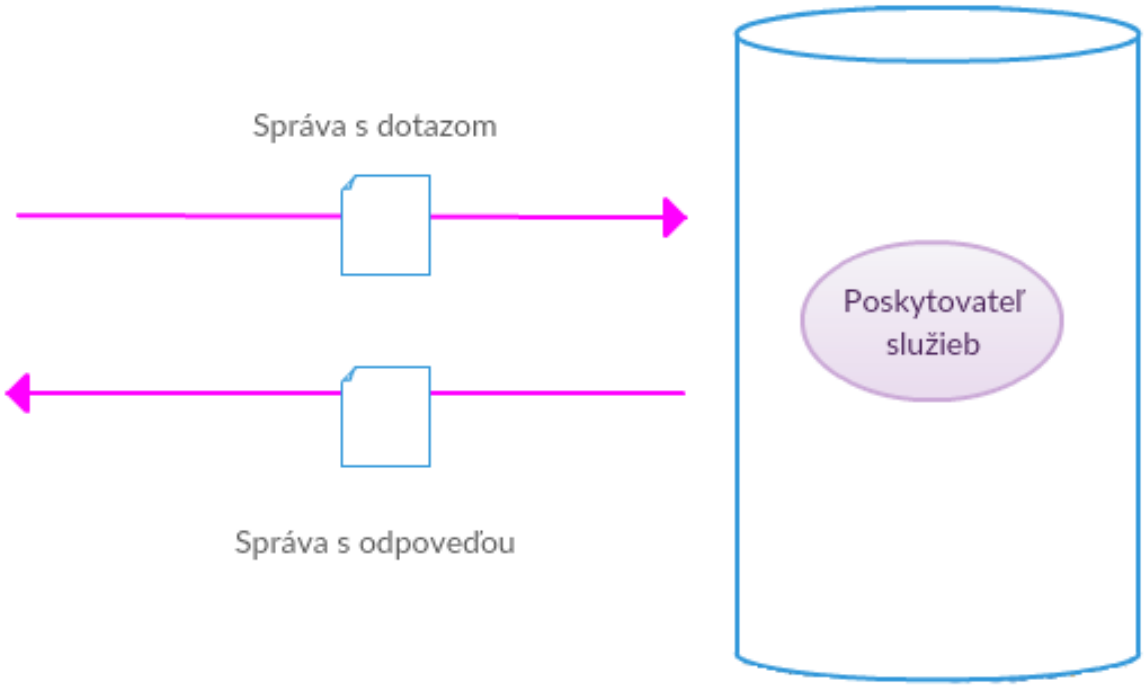
1.3.1 Funkcie webových služieb

Ako už bolo spomínané, webová služba disponuje flexibilnými a adaptívnymi vlastnosťami. To dokazuje napríklad fakt, že webová služba je schopná spĺňať rôzne funkcie a meniť svoju rolu v rámci daného systému. Môže napríklad vystupovať ako iniciátor požiadavku, prenášač správy alebo príjemca správy. Webovú službu teda môžeme označiť ako jednotku

softwaru, schopnú zmeniť svoju funkciu v systéme vzhľadom na potrebu systému v danej situácií.[14]

1.3.1.1 Poskytovateľ služieb

Pokiaľ je služba volaná externým zdrojom, ako je napríklad žiadateľ služieb a webová služba poskytuje popis služby, vďaka ktorým informuje o svojich funkcionalitách a o svojom chovaní môžeme hovoriť, že služba spĺňa funkciu poskytovateľa. Túto funkciu môžeme zrovnať s funkciou serveru pri klasickej architektúre klient – server. Webová služba, ktorá pôsobí ako poskytovateľ služieb, býva využívaná k identifikácií spoločností zodpovednej za poskytnutie danej webovej služby.

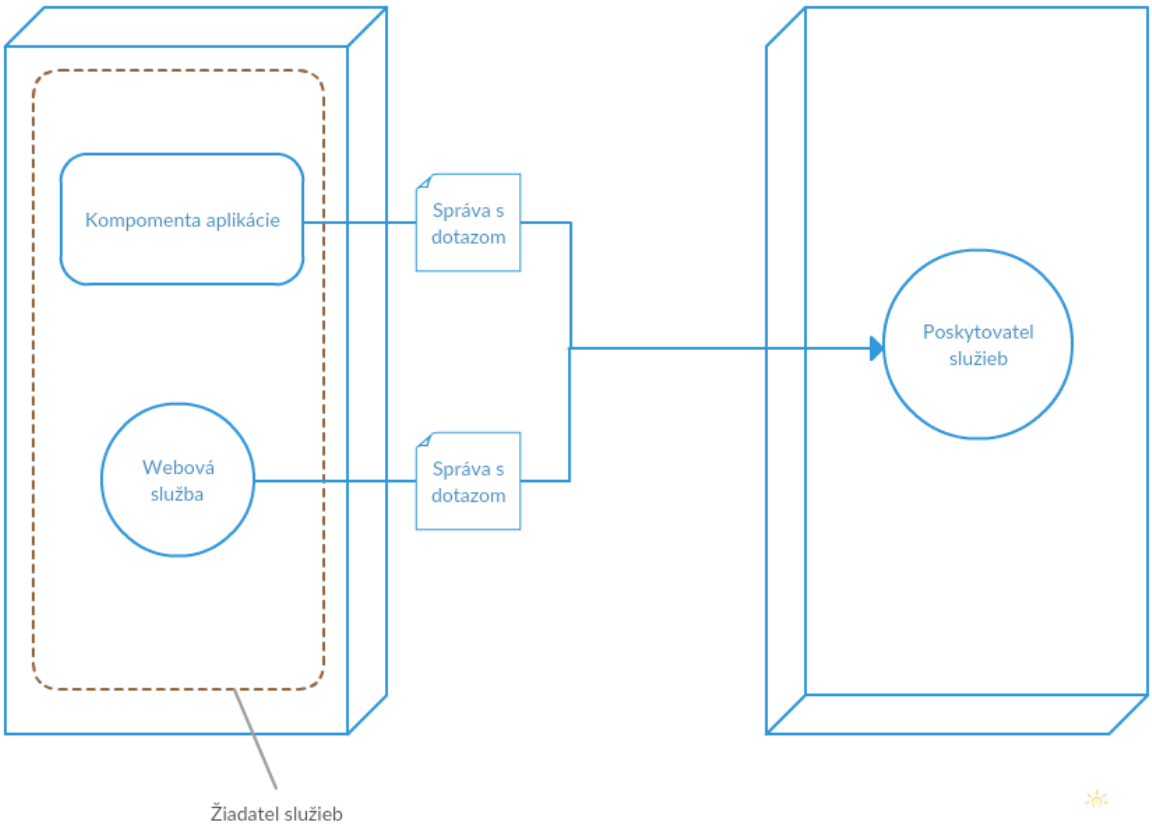


Obr. 4 Webová služba ako poskytovateľ služieb

1.3.1.2 Žiadateľ služieb

Webová služba, ktorá naplňuje funkciu žiadateľa služieb, je vlastne akákoľvek jednotka, ktorá dokáže logicky spracovať správu obsahujúcu dotaz a poskytovateľ služieb je schopný tejto správe porozumieť. Funkciu poskytovateľa služieb, ktorá je opísaná v predošlej kapitole, vykonáva svojím spôsobom každá webová služba, avšak niektoré z nich môžu tiež vystupovať aj ako žiadatelia služieb.

V predchádzajúcej kapitole je taktiež spomenutý fakt, že poskytovateľ služieb odosiela správu s odpoveďou, čo sa môže javiť ako žiadateľ služieb. Avšak poskytovateľ služieb nevystupuje ako žiadateľ, keď odosiela správu ako reakciu na správu s požiadavkami. Môžeme sa na žiadateľa služieb dívať ako na program, ktorý zahajuje komunikáciu s poskytovateľom služieb viac Obr. 5.



Obr. 5 žiadatel služieb

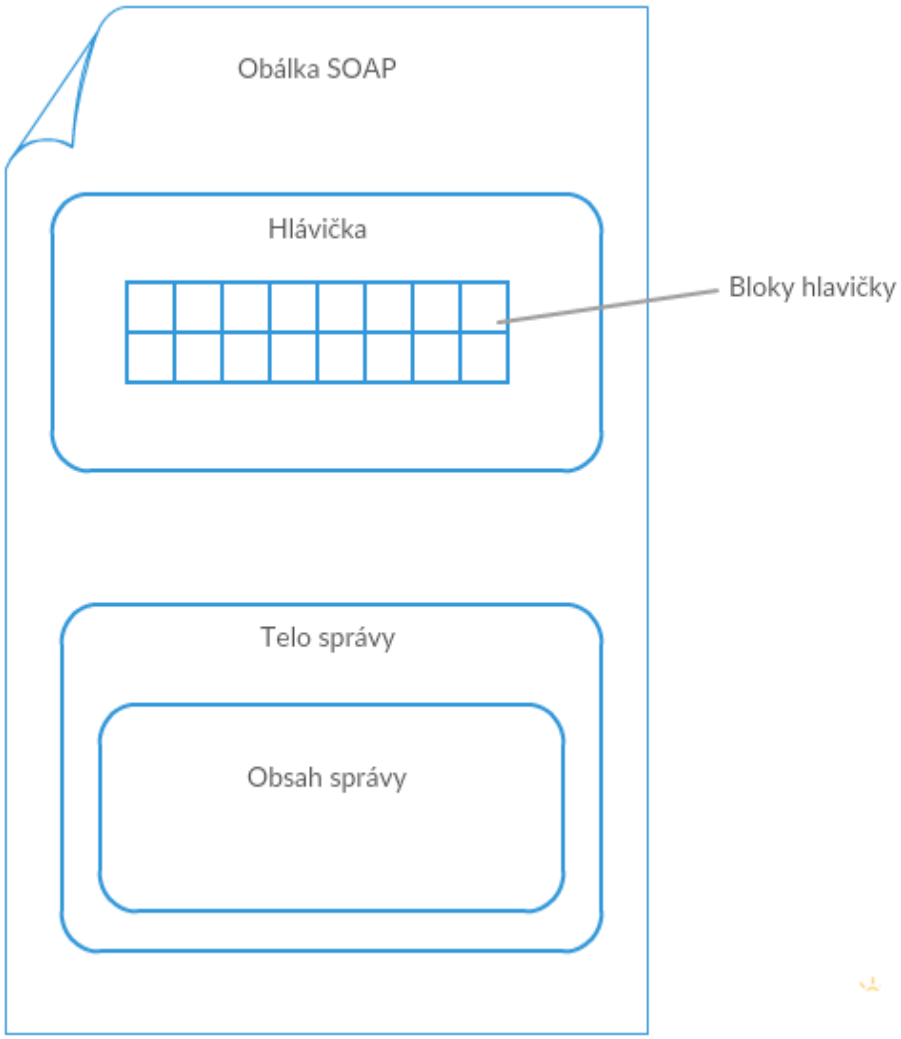
1.3.2 Základy protokolu SOAP

Protokol SOAP (Simple Object Acces Protocol) je protokol, ktorý slúži pre štandardizáciu procesu, výmeny správ medzi jednotlivými službami bez ohľadu na pôvod, používaný formát a protokol prenosu danej správy. V servisne orientovanej architektúre je prijatie služby základnou akciou. Táto akcia vyžaduje aby bol systém výmeny správ extrémne flexibilný a taktiež rozšíriteľný. Protokol SOAP sa stal obecne prijímaným štandardným protokolom prenosu správ práve preto, že je schopný tieto podmienky splniť. Od svojho prvého vydania je protokol neustále upravovaný, aby sa prispôsobil prepracovanejším štruktúram správ. V preklade Simple Object Acces Protocol síce znamená jednoduchý prístupový

objektový protokol avšak jeho hlavná funkcia spočíva v definícií štandardných formátov správ. Štruktúra tohto protokolu je síce jednoduchá, ale fakt že ju dokážeme rozšíriť a upraviť, robí z tohto protokolu hnaciu silu servisne orientovanej architektúry.

1.3.2.1 Štruktúra správy protokolu SOAP

Každá správa tvorená štandardom SOAP je zabalená do takzvanej obálky. Obálku si môžeme predstaviť ako pomyselné zapuzdrenie vnútorných častí správ. Takáto obálka potom obsahuje hlavičku správy a telo správy. Štruktúra správy definovaná pomocou protokolu SOAP je zobrazená na Obr. 6.



Obr. 6 Štruktúra správy protokolu SOAP

1.3.2.2 Hlavička

Každá správa definovaná pomocou protokolu SOAP môže obsahovať hlavičku, v ktorej sú uchovávané metadáta[1]. Hlavička nie je povinná súčasť správy, avšak vo väčšine servisne orientovaných riešení býva súčasťou každej správy pretože vďaka hlavičke sme schopný implementovať rôzne rozšírenia.

Pomocou hlavičiek správy sme schopný dosiahnuť nezávislosť správ a rozšíriteľnosť správ pričom bloky hlavičiek danej správy ukladajú informácie potrebné pre služby, ktoré so správou prídu do kontaktu a to tak aby boli schopné správu spracovať a smerovať v súlade s pravidlami, inštrukciami a vlastnosťami. Webové služby môžeme teda navrhnúť tak, aby boli schopné riadiť svoje funkcie pre spracovanie správy na základe hlavičiek správy.

1.3.3 Popis služieb pomocou jazyku WSDL

Ako už bolo povedané servisne orientovanú architektúru tvoria aj služby, pričom každá služba je popísaná a to práve pomocou jazyku WSDL (Web Services Description Language). Je to neoddeliteľná súčasť servisne orientovanej architektúry pre zavedenie konzistentnej voľnej viazanej formy komunikácie medzi službami implementovanými ako webové služby.

1.3.3.1 Koncový bod

Jazyk WSDL je schopný popísať takzvaný koncový bod služby, ktorý slúži ako kontaktný bod pre poskytovateľa služieb. Na základe toho teda dostávame formálnu definíciu koncového rozhrania a tiež zavádzame fyzické umiestenie alebo adresu služby. Popis služieb pomocou jazyku WSDL môžeme rozdeliť do dvoch kategórií ktoré ďalej obsahujú :

- Abstraktný popis
    - Typ portu
    - Operácia
    - Správa
- Konkrétny popis
    - Väzba
    - Port alebo koncový bod
    - Služba

[1] Štruktúrované dáta nesúce informácie o primárnych dátach.

1.3.4 Jazyk XML a jeho definícia pomocou XSD

XML(eXtensible Markup Language) je univerzálny textový formát, ktorý využíva k prezentácií štruktúrované dáta a predovšetkým sa používa za účelom výmeny informácií.

Jazyk XSD (XML Schema Definition Language) slúži ako prostriedok pre definíciu XML schémy. Tento jazyk sa stal bežnou súčasťou architektúr postavených na jazyku XML a webových službách. XML dokument môžeme tiež chápať ako instanciu odpovedajúcej XSD schéme. XSD schéma tiež definuje sériu pravidiel a obmedzení, ktoré musí instancia XML dokumentu splňovať.

Základné pravidlá reprezentácie dát poskytovaných XSD schémou súvisia s reprezentáciou dát podľa typu. Podobne ako dátové typy používané v iných programovacích jazykoch poskytuje aj XSD schéma sadu dátových typoch používanú pre reprezentáciu informácií v XML dokumentoch.

XSD schémy môžu tiež existovať ako samostatné dokumenty typicky s príponou .xsd alebo sú súčasťou iných typov dokumentov, ako napríklad instancie XML dokumentov alebo WSDL definície pričom instancie XML dokumentov sa často odkazujú na samotný súbor XSD. Na základe toho je možné teda použiť viacero instancií XML dokumentov na jedinú XSD schému. Ukážka Kód 1 predstavuje instanciu XSD schémy, ktorá je ďalej uvedená na ukážke Kód. 2 pričom sa jedná o vzorový príklad zákazníka zo základnými údajmi ako zákazníkovo špecifické číslo, meno, vek a tak ďalej. Na ukážke Kód.2 je vidieť definované dátové typy jednotlivých položiek XML dokumentu[1].

```
<?xml version="1.0" encoding="UTF-8"?>
<CustInfo infotype="EshopCustomer">
  <custInfoId>1M</custInfoId>
  <firstName>Meno</firstName>
  <lastName>Priezvisko</lastName>
  <age>29</age>
  <address>Adresa zakaznika</address>
  <orderId>1</orderId>
</CustInfo>
```

Kód. 1 Ukážka XML dokumentu

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="CustInfo">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="custInfoId"/>
        <xs:element type="xs:string" name="firstName"/>
        <xs:element type="xs:string" name="lastName"/>
        <xs:element type="xs:byte" name="age"/>
        <xs:element type="xs:string" name="address"/>
        <xs:element type="xs:string" name="orderId"/>
      </xs:sequence>
      <xs:attribute type="xs:string" name="infotype"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Kód. 2 Ukážka schémy XSD

1.4 Moderná SOA

Základy servisne orientovanej architektúry ostávajú stále rovnaké, avšak nedávne početné priemyselné a vývojové trendy poskytli príležitosť, ktorá zapríčinila veľké rozšírenie servisne orientovanej architektúry.

Významní distribútori softwaru neustále formulujú a vylepšujú nové špecifikácie webových služieb a poskytujú stále lepšiu podporu XML a webových služieb do aktuálnych technologických platforiem. Výsledkom je rozšírená verzia servisne orientovanej architektúry, označená ako moderná (alebo aj súčasná) servisne orientovaná architektúra.

1.4.1 Zlepšenie kvalít služieb

Pokiaľ chceme priviesť SOA do takého stavu kedy je možné implementovať funkcionalitu na úrovni podniku, je nutné zabezpečiť bezpečnosť a spoľahlivosť tak ako to robí stále viac zavedených architektúr.

Z toho vyplýva teda potreba splniť určité požiadavky na kvalitu služieb v takejto architektúre. Tieto požiadavky môžu byť napríklad nasledovné :

- Možnosť uskutočňovať úlohy bezpečným spôsobom tak, aby bola zaručená ochrana obsahu správ a prístup k jednotlivým službám.
- Možnosť uskutočňovať úlohy spoľahlivo, tak aby bolo možné zaručiť doručenie správy a taktiež oznámenie o prípadnom neúspechu pri odosielaní správy.

- Potreba zabezpečiť aby réžia spojená zo správou SOAP[2] a spracovaním XML[3]nebola prekážkou pri uskutočňovaní úlohy.
- Transakčná schopnosť pre ochranu integrity špecifických riadiacich úloh zo zárukou kedy by úloha nebola vykonaná, bude spustená výnimková logika.

1.4.2 Vždy autonómna

Servisne orientovaný princíp je založený na tom aby jednotlivé služby boli v čo najväčšej miere sebestačné a nezávislé tak, ako to len bude možné s ohľadom na logiku v pozadí. Samostatnosť v SOA je dosiahnutá pomocou správ, pričom správy, ktoré si služby vymieňajú sú dostatočne inteligentné nato, aby kontrolovali spôsob akým budú spracované službami.

SOA stavia na tomto princípu a rozširuje jej propagáciu návrhu samosprávy cez celé prostredie daného riešenia a podniku. Aplikácie zložené z autonómnych služieb môžu teda napríklad samy seba vidieť ako zložené sebestačné služby, ktoré vykonávajú svoju samosprávu v servisne orientovaných zjednotených prostrediach.

1.4.3 Otvorené štandardy

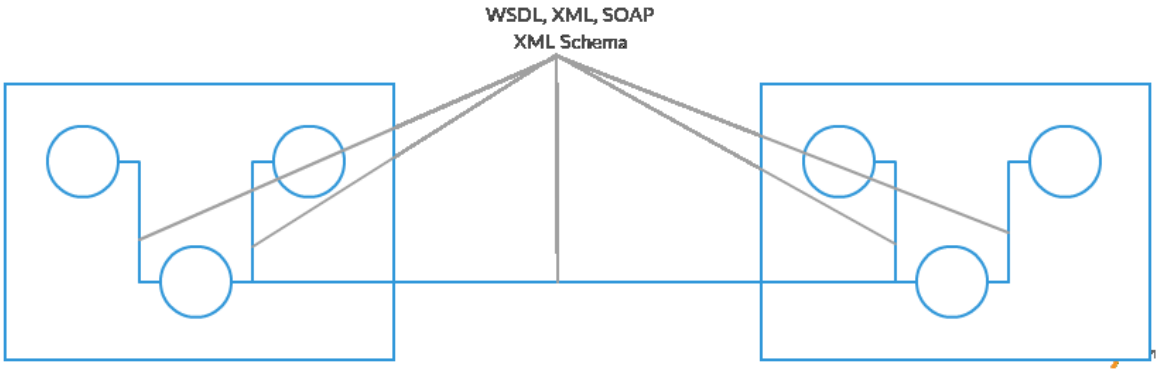
Základnou a pomerne najdôležitejšou vlastnosťou webových služieb je fakt, že výmena dát ktorá medzi nimi prebieha je realizovaná pomocou otvorených štandardov. Tieto štandardy sú schopné zabezpečiť už vyššie spomínanú autonómnosť správ. V momente kedy je správa zo služby odoslaná putuje napríklad k druhej službe prostredníctvom sady protokolov, ktoré sú obecne štandardizované a prijímané.

Tak isto ako výmena dát je aj samotná správa štandardizovaná či už vo formáte v ktorom je posielaná alebo v spôsobe akým popisuje svoj obsah. To aby boli správy sebestačné a to aby podporovali fakt, že pre komunikáciu službám stačí iba znalosť popisu ostatných služieb je zabezpečené pomocou štandardov SOAP, WSDL, XML, a XML Schema. Vďaka týmto štandardom sa vylučuje požiadavka, aby v pozadí ležiaca logika služieb zdieľala typový systém.

Moderná SOA teda plne podporuje otvorené štandardy a vďaka tomu je teda vždy možné zvoliť komunikáciu medzi službami.

---

[2] Protokol pre výmenu správ založený na jazyku XML
[3] Rozšíriteľný značkovací jazyk



Obr. 7 Grafické znázornenie otvorených štandardov

1.4.4 Podpora viacerých platforiem

Táto vlastnosť servisne orientovanej architektúry úzko súvisí už so sekciou o otvorených štandardoch. Otvorené štandardy nie len že umožňujú správam správať sa autonómne a sebestačne, ale taktiež umožňujú firmám neobmedzovať s používaním rôznych platforiem, vývojových prostredí a podobne.

Bez ohľadu na to kto vlastní vývojové prostredie v ktorom firma vyvíja alebo ak firma pracuje s rôznymi platformami stačí, aby dané vývojové prostredie podporovalo tvorbu štandardných webových služieb a môže byť použité pre tvorbu otvorenej vrstvy rozhrania služieb. To dovoľuje následnú komunikáciu medzi inými aplikáciami, ktoré sú založené na službách a tie môžu byť vytvorené v iných prostrediach. Názorná ukážka Obr. 8. Samozrejme firmy môžu naďalej pracovať na platformách na akých sú zvyknuté, avšak pomocou otvorenej technológie poskytovanej systémom webových služieb a princípov, ktoré ponúka servisne orientovaná architektúra, majú možnosť preskúmať iné možnosti bez toho aby sa báli možného pádu fungujúceho systému.

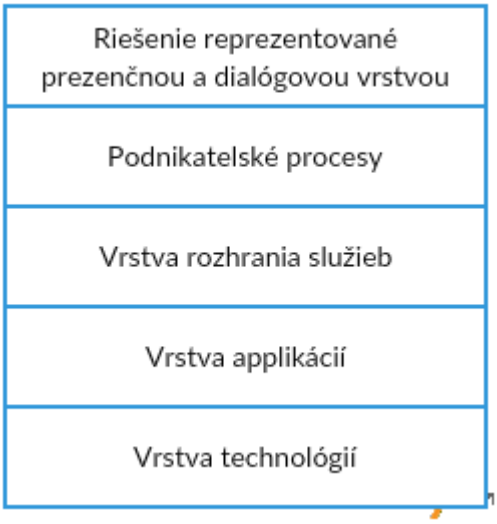
Obr. 8 Multiplatformová komunikácia

Implementovanie SOA do podniku však nemusí nutne znamenať to, že sa nahradí celý fungujúci systém novým, ktorý využíva webové služby. Avšak je možné zaviesť jednotnosť. Pomocou SOA dokážeme zapuzdriť zastaralú a nezastaralú aplikačnú logiku a používať ju pomocou bežného, otvoreného a štandardizovaného systému komunikácie.

Hlavnou výhodou začlenenie SOA do fungujúcich platforiem je, že servisne orientovaným zjednotením vzniknú komunikačné kanály, ktoré sú štandardizované.

1.5 Vrstvová architektúra v SOA

Princípy založené na servisne orientovanej architektúre sú pojímané pomocou referenčnej vrstvenej architektúry, ktorý je znázornený na Obr. 9



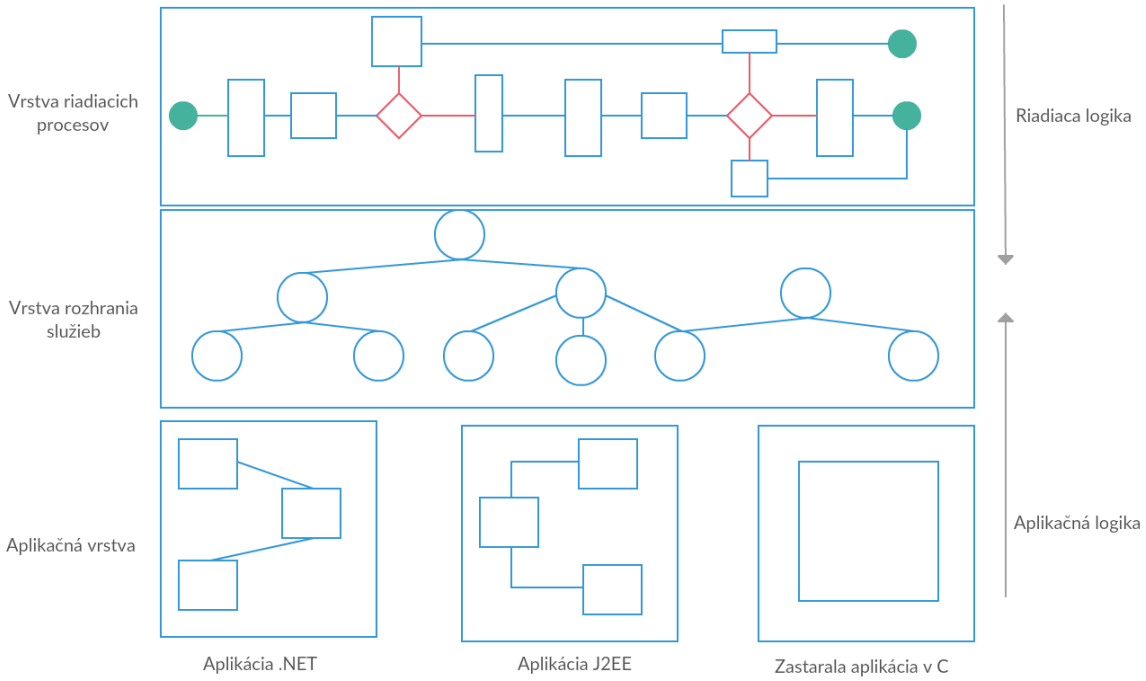
Obr. 9 Referenčný vrstevný model SOA

U uvedenom modeli však predstavuje vrstva podnikateľských procesov iba konceptuálnu vrstvu, to znamená že v modeli je zaradená len z dôvodu aby bola znázornená tiež väzba medzi podnikateľskými procesmi a službami, ktoré môžu byť určené pre podporu alebo k vykonávaniu týchto procesov. Znázornené vrstvy služieb, aplikácií a technológií sú oddelené fyzické vrstvy, reprezentované skutočnými aplikačnými programami, programovým vybavením či hardwarom. Na vrchole modelu sa nachádza vrstva riešenia. Táto vrstva je naplnená klientskymi aplikáciami, ktoré služby využívajú. To umožňuje, aby služba mohla byť užitočná skrz rôzne riešenia napríklad webová aplikácia, aplikácia v mobilnom telefóne, aplikácia úradníka na pokladni.[3]

1.5.1 Aplikačná a riadiaca logika

V časti aplikačná a riadiaca logika sú viac rozobraté vrstvy: podnikateľské procesy, rozhranie služieb. S referenčného vrstevného modelu SOA. Tieto vrstvy reprezentujú totiž neobytnú súčasť organizačnej štruktúry. Vrstva podnikateľské procesy (alebo aj riadiace procesy) je vlastne implementáciou požiadavky na riadenie, ktoré pochádzajú od vedenia podniku.

Aplikačná logika je teda implementáciou riadiacej logiky. To v jednoduchosti znamená, že plní riadiace pokyny od vrstvy podnikateľské procesy. A to za pomoci vlastných alebo zakúpených systémov v rámci IT infraštruktúry.



Obr. 10 Vrstva rozhrania služieb

Na Obr. 10 je zobrazené, kde sa služby nachádzajú v celkovej štruktúre automatizovanej spoločnosti. Z obrázku vyplýva, že služby zavádzajú abstrakciu medzi tradičnou riadiacou a aplikačnou vrstvou, pričom na vrstve rozhrania služieb môžu zapuzdrovať aplikačnú logiku.

Vďaka tomuto konceptu služby rozdeľujú logiku podniku na moduly a vytvárajú tak samostatné jednotky. Z obrázku tiež vyplýva že vrstva služieb sa môže skladať z niekoľko vrstiev abstrakcie.

Na Obr. 10 je možné vidieť aplikačnú vrstvu, v ktorej sa nachádzajú tri aplikácie, ktoré sú symbolicky ohraničené svojou platformou. Avšak vďaka službám, ktoré sú v jednej kontinuálnej vrstve je zabezpečená otvorená spojitosť medzi rozhraniami služieb. Keďže medzi službami funguje otvorená väzba, služby môžu voľne komunikovať medzi sebou a to za pomoci otvorených protokolov.

2 PODNIKOVÁ ZBERNICA SLUŽIEB

2.1 Definícia

Podniková zbernica tvorí jeden zo základných stavebných prvkov servisne orientovanej architektúry. V podstate ide o zoskupenie rôznych pravidiel a zásad určených k integrácií aplikácií či služieb. Podniková zbernica spojuje a sprostredkováva všetky komunikácie a interakcie medzi službami. Zároveň však dovoľuje služby a procesy rýchle meniť a tiež ich ľahko pripojovať, zviditeľniť či riadiť.

Hlavnou myšlienkou je integrácia odlišných a typovo rôznych aplikácií. A to takým spôsobom, že umiestnime medzi tieto aplikácie spomínanú podnikovú zbernicu, ktorá vždy príjme dáta od aplikácií za podmienky, že je na zbernicu napojená. Tento fakt navzájom oddeľuje systémy od seba, čo im umožní komunikovať bez toho aby boli závislé na vedomosti ostatných služieb alebo aplikácií.[4]

Podniková zbernica je schopná prijať dáta od rôznych aplikácií či služieb fungujúcich na rozličných platformách. Tieto dáta je schopná spracovať rôznymi spôsobmi, ako napríklad transformovať do podoby akú potrebujeme, vykonať požadované výpočty, uložiť dáta do požadovanej databázy alebo len preposlať ďalej do inej webovej služby[5].

2.2 Podniková zbernica služieb a jej pozícia v servisne orientovanej architektúre

Pre možnosť implementácie servisne orientovanej architektúry je potrebné, aby aplikácie a infraštruktúra podporovala jej princípy a zásady. Podpora princípov a zásad servisne orientovanej architektúry zahŕňa vytvorenie servisných rozhraní a to buď k existujúcim funkcionalitám alebo k novým. To môžeme dosiahnuť priamo alebo pomocou adaptérov. Využitie infraštruktúry na najzákladnejšej úrovni zahŕňa poskytnutie schopnosti smerovať a prepravovať požiadavky na služby, správnemu poskytovateľovi služieb. Úlohou podnikovo orientovanej zbernici v takejto architektúre je takúto infraštruktúru zrealizovať.

Avšak skutočnou hodnotou konceptu podnikovej zbernice služieb je umožniť infraštruktúru pre servisne orientovanú architektúru spôsobom, ktorý odráža potreby súčasného podniku. Tieto potreby spočívajú vm poskytovaní vhodnej úrovne služieb, spravovateľnosti, funkčnosti a možnosti integrovať sa do heterogénneho prostredia. Podniková zbernica služieb by mala byť schopná nahradiť implementáciu jednej služby

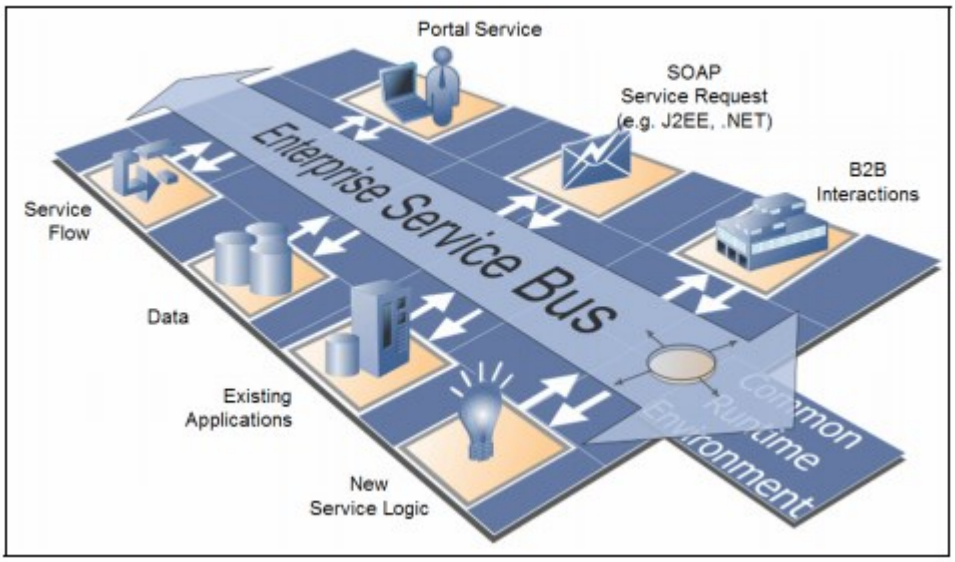
druhou bez nežiadúceho efektu, ktorý by mohol ovplyvniť klienta danej služby. To však vyžaduje servisné rozhranie špecifické pre servisne orientovanú architektúru ale aj podnikovú zbernicu služieb, ktorá dovolí klientovi vyvolávať služby spôsobom, ktorý je nezávislý od miesta vyskytovania služby a tiež od komunikačného protokolu.

2.2.1 Integračné paradigmy podnikovej zbernice služieb

Za účelom plnej podpory rôznych možných interakcií, ktorú sú potrebné v komplexnej servisne orientovanej architektúre ako napríklad interakcie typu : (request / response, publish / subscribe, events) je potrebné, aby podniková zbernica služieb podporovala v jednej infraštruktúre až tri typy podnikovej integrácie ktoré sú :

- **Servisne orientovaná architektúra,** kde aplikácie komunikujú prostredníctvom znovu použiteľných služieb s jasne definovanými explicitnými rozhraniami. Servisne orientované interakcie využívajú komunikačné modely ako správy a udalosti.
- **Architektúra riadená správami,** kde aplikácie posielajú správy pomocou podnikovej zbernice služieb.
- **Architektúra založená na udalostiach**, v ktorej aplikácie vytvárajú a prijímajú správy nezávisle od seba.

Podniková zbernica služieb vykonáva činnosť pri poskytovaní dodatočných spôsobilostí na sprostredkovanie alebo transformáciu správ a interakciu medzi službami, čo umožňuje širokú škálu správania a podporu rôznych modelov spájania.

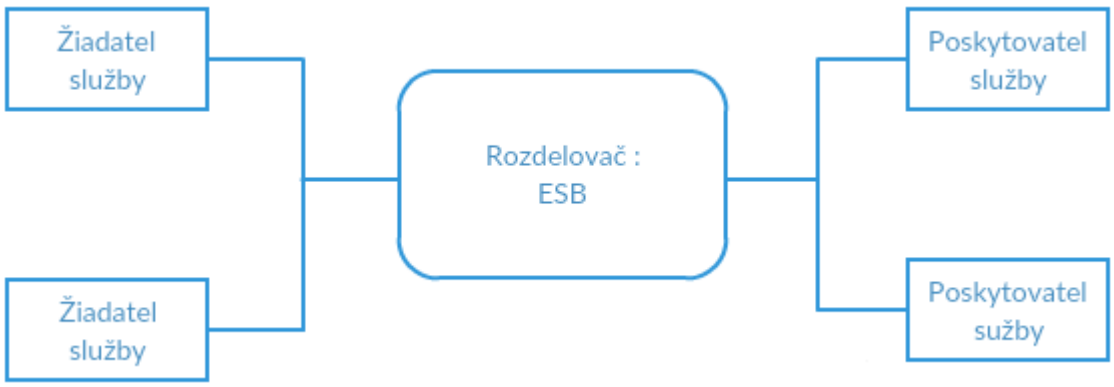


Obr. 11 Integrácia pomocou podnikovej zbernici služieb

2.2.2 Podniková zbernica ako prostriedok pre kontrolu a distribúciu správ

Podniková zbernica často býva znázorňovaná ako čiara do ktorej a s ktorej smerujú služby, aplikácie či rôzne iné komponenty tak ako je to znázornené na Obr. 11. Avšak čo sa fyzicky skrýva za touto líniou býva často nejasné.

Podniková zbernica býva tiež popísaná ako distribuovaná infraštruktúra, ktorá môže byť v kontraste riešením „hub-and-spoke“ [4] viac Obr. 12.

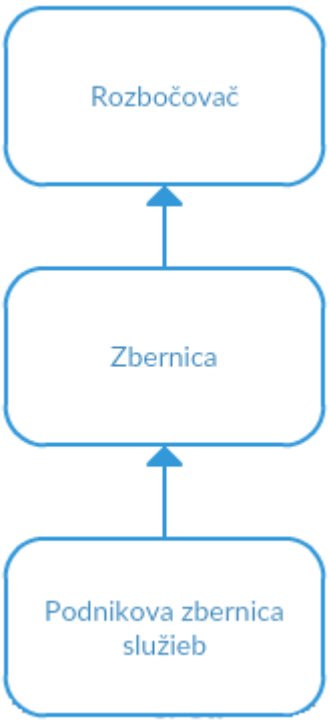


Obr. 12 Integrácia hub-and-spoke

V podnikových vzoroch môže byť podniková zbernica jednoducho klasifikovaná ako určitý typ zbernice, pričom tento typ je definovaný ako typ rozbočovača.

[4] Rozdeľovacia paradigma, ktorá sa bežne používa v priemysle, najme v oblasti dopravy, telekomunikácií, nákladnej dopravy a distribuovaných informačných systémov

Rozbočovač môže byť fyzicky rozdelený na zoskupenie menších federovaných[5] rozbočovačov. Tento prípad je používaný spoločné s federovanými adaptérmi vďaka ktorým je umožnené prepojenie klientov s protokolmi a zo systémom posielania správ s ktorým podniková zbernica služieb disponuje.



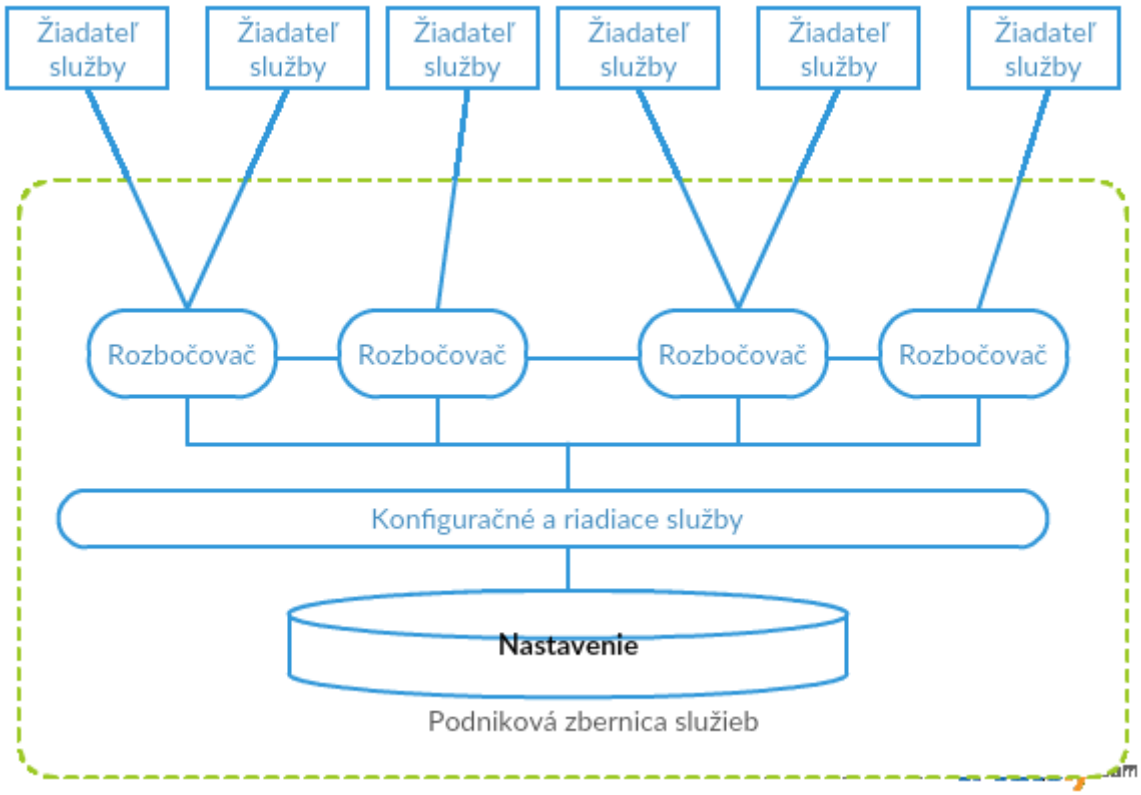
Obr. 13 Vzťah medzi rozbočovačom, zbernicou a podnikovou zbernicou služieb

Na Obr. 10 je zobrazený vzťah medzi jednotlivými integračnými riešeniami tento vzťah si bližšie popíšme na základe jeho jednotlivých zložiek.

- Rozbočovač poskytuje logicky centralizovanú integračnú službu ako napríklad : integračný server, databáza.
- Zbernica slúži ako rozbočovač, ktorý podporuje distribuovanú alebo federatívnu implementáciu
- Podniková zbernica služieb slúži ako zbernica, ktorá okrem iného predstavuje virtuálny meniaci sa priestor ktorý je možné spravovať ako jeden celok.

[5] Prepojenie niekoľkých samostatných prostriedkov takým spôsobom, že vystupujú ako jeden väčší prostriedok.

Hovoriť teda o kontrastne medzi distribuovaným riešením podnikovej zbernice a centralizovaným riešením *„hub-and-spoke“* nie je úplne správne a to s dôvodu že sa jedná o rôzne problémy a to centralizáciu kontroly a distribúciu infraštruktúry. V prvotných implementáciách integračných riešení je fyzická infraštruktúra pravdepodobne sústredená na jednom rozbočovači. Keďže sa však implementácia vyvíja, infraštruktúra sa môže stať viac fyzicky distribuovaná ako zbernica. Pričom sa zachová logická kontrola nad konfiguráciou.



Obr. 14 Podniková zbernica ako distribučná infraštruktúra s centralizovanou kontrolou

Obr. 14 znázorňuje výslednú implementáciu podnikovej zbernici služieb ako distribučnú infraštruktúru s centralizovanou kontrolou pričom časť určená pre konfiguráciu a riadenie služieb je zobrazená prerušovaným ohraničením.

Avšak implementácia takejto infraštruktúry závisí od možnosti špecifických technológií na podporu týchto distribučných modelov. Rovnako dôležité z hľadiska prírastkovej im-

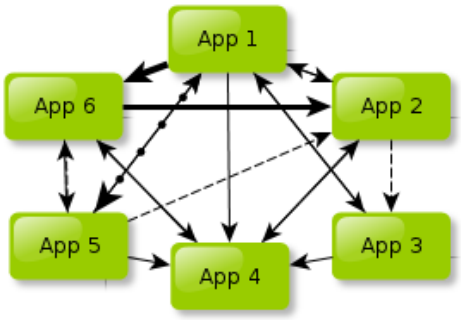
plementácie je schopnosť rozšíriť existujúce nasadenia pridaním ďalšej distribuovanej kapacity bez ovplyvnenia existujúcej.[7]

2.3 Point to point integrácia a podniková zbernica

Koncept integrácie pomocou podnikovej zbernice vznikol práve kvôli nedostatkom integračného konceptu point-to-point (bod k bodu), ktorý pri väčšom informačnom systéme nie je dostačujúci.

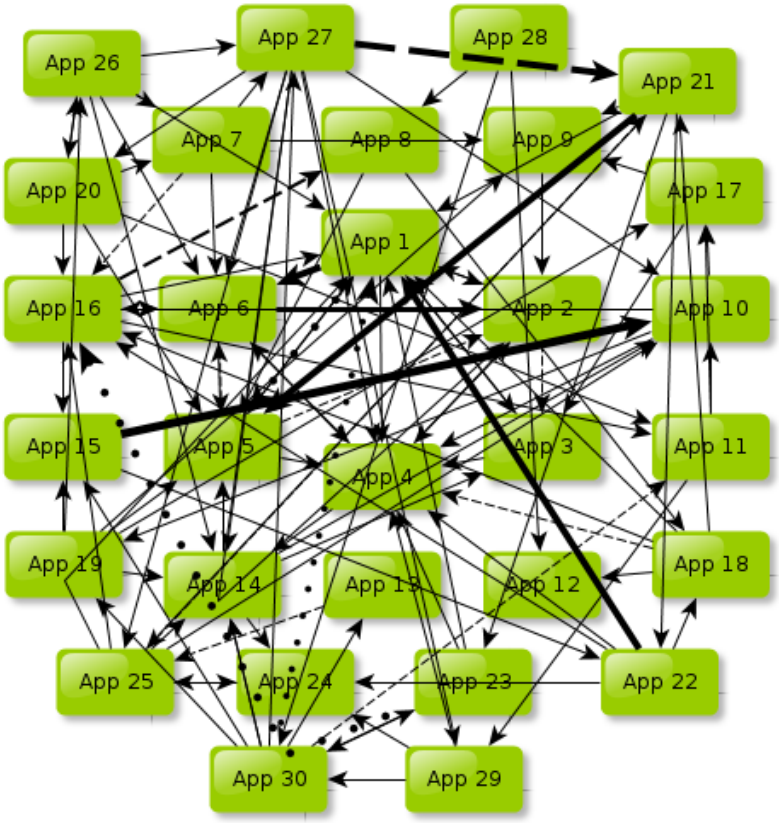
Point to point integrácia alebo tiež priama väzba je typ integrácie, ktorý je založený na spojený dvoch aplikácií pričom musíme zaistiť aby aplikácia dokázala zabezpečiť transformáciu a tok dát, do podoby ktorej rozumie cieľová aplikácia, pričom transformácia môže byť tiež vykonaná cieľovou aplikáciou.

Takéto riešenie je naozaj výhodné avšak pri jednoduchších informačných systémoch kedy sa tento informačný systém skladá iba s niekoľkých aplikácii či služieb. Príklad takého systému riešeného point to point integráciou je zobrazený na Obr. 15.



Obr. 15 Point to point integrácia [6]

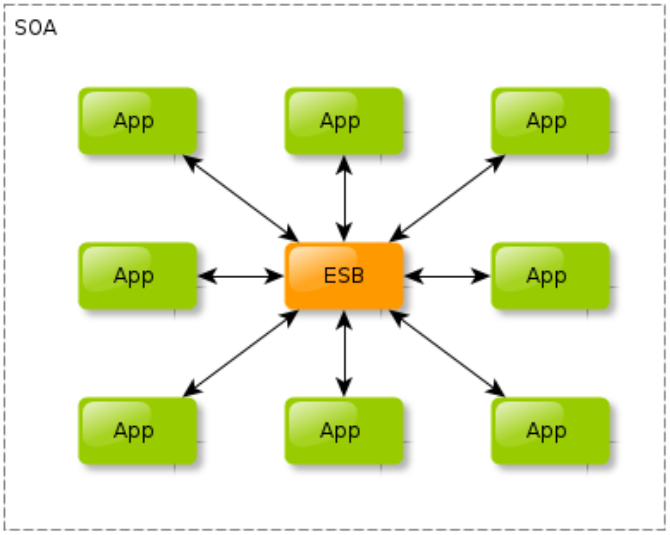
V taktom prípade teda môže byť systém zvládnuteľný. Avšak v dôsledku priamych väzieb, ktoré vznikajú pri point to point integrácií môže vzniknúť pri väčšom informačnom systém takzvaná špagetová integrácia, ktorú názorne zobrazuje Obr. 16.



Obr. 16 Špagetová integrácia [6]

S Obr. 16 je teda zrejmé že pri rozsiahlejších informačných systémoch je tento druh integrácie neprehľadný a tým pádom ťažko ovládateľný. Riešenie takéhoto problému môže byť práve v použití podnikovej zbernici služieb.

Ako vidíme s Obr.15 aj pri menšom informačnom systéme a použití integrácie point to point je potrebné zabezpečiť komunikáciu medzi jednotlivými aplikáciami a to tak aby každá aplikácia mohla komunikovať s každou aplikáciou. Vytvára to tak veľký počet väzieb a to znamená vytvorenie množstva prepojení. Avšak pokiaľ používame podnikovú zbernicu ktorá sprostredkuje a rozhoduje o komunikácií medzi službami môžeme počet týchto väzieb a teda aj vytvorených prepojený pomerne značne redukovať viac Obr. 17.



Obr. 17 Integrácia pomocou podnikovej zbernici služieb [6]

Na Obr. 17 je teda vidno že počet vytvorených prepojený zmenšil a to vďaka riešeniu pomocou podnikovej zbernice služieb ktorá zabezpečuje komunikáciu medzi jednotlivými službami.

2.4 Možnosti podnikovej zbernici služieb

V tabuľke 1 sú zhrnuté a rozdelené podľa kategórií, niektoré z funkcií podnikovej zbernice služieb. Tieto služby sú prevzaté s existujúcej voľne dostupnej literatúry. Tieto možnosti sa používajú na posúdenie vhodnosti rôznych dostupných riešení pre implementáciu podnikovej zbernici služieb. Mnohé s týchto možností môžu byť implementované pomocou vlastných technológií alebo pomocou otvorených štandardov. Rozhodovanie o tom, či by sa niektorá z nich mala implementovať prostredníctvom vlastných technológií alebo pomocou otvorených štandardov, patrí medzi kľúčové rozhodnutia pri navrhovaný servisne orientovanej architektúry[6].

Komunikácia	**Interakcia so službami**
Smerovanie Adresovanie Štandardy a protokoly (HTTP, HTTPS) Publish/Subscribe Request/Response Fire & Forget, udalosti[6] Synchrónne a asynchrónne posielanie správ	WSDL rozhranie pre prácu so službami Substitúcia implementácie služieb Moduly služieb potrebné pre komunikáciu a integráciu (SOAP, XML) Adresár služieb
Integrácia	**Kvalita služieb**
Databáza Aplikačné adaptéry Pripojenie k middleware [7]pre integráciu podnikových aplikácií Mapovanie služieb Transformácia protokolov Podpora prostredia aplikačného serveru (J2EE, .NET) Jazykové rozhrania pre vyvolanie služieb (Java, C#, C/C++)	Transakcie (atomické transakcie, kompenzácie, WS - Transakcie) Rôzne zabezpečené paradigmy doručovania (WS - ReliableMessaging [8]alebo podpora pre integráciu podnikových aplikácií)
Bezpečnosť	**Úroveň služieb**
Autentizácia Autorizácia Non-repudiation[9] Dôvernosť Bezpečnostné štandardy (Kerberos, WS-Security)	Predstavenie Priepustnosť Dostupnosť Ďalšie nepretržité opatrenia, ktoré môžu byť stanovené na základe zmlúv alebo dohody
Spracovanie správ	**Manažment a autonómia**
Kódovacia logika Logika založená na obsahu Transformácia správ a dát Agregácia, korelácia služieb a správ Validácia Sprostredkovanie Mapovanie identity objektu Store and forward[10]	Administračné možnosti Poskytovanie a registrácia služieb Logovanie Zaznamenávanie metrík Monitorovanie Integrácia do riadenia systémov a nástrojov pre správu Samokontrola a samo riadenie

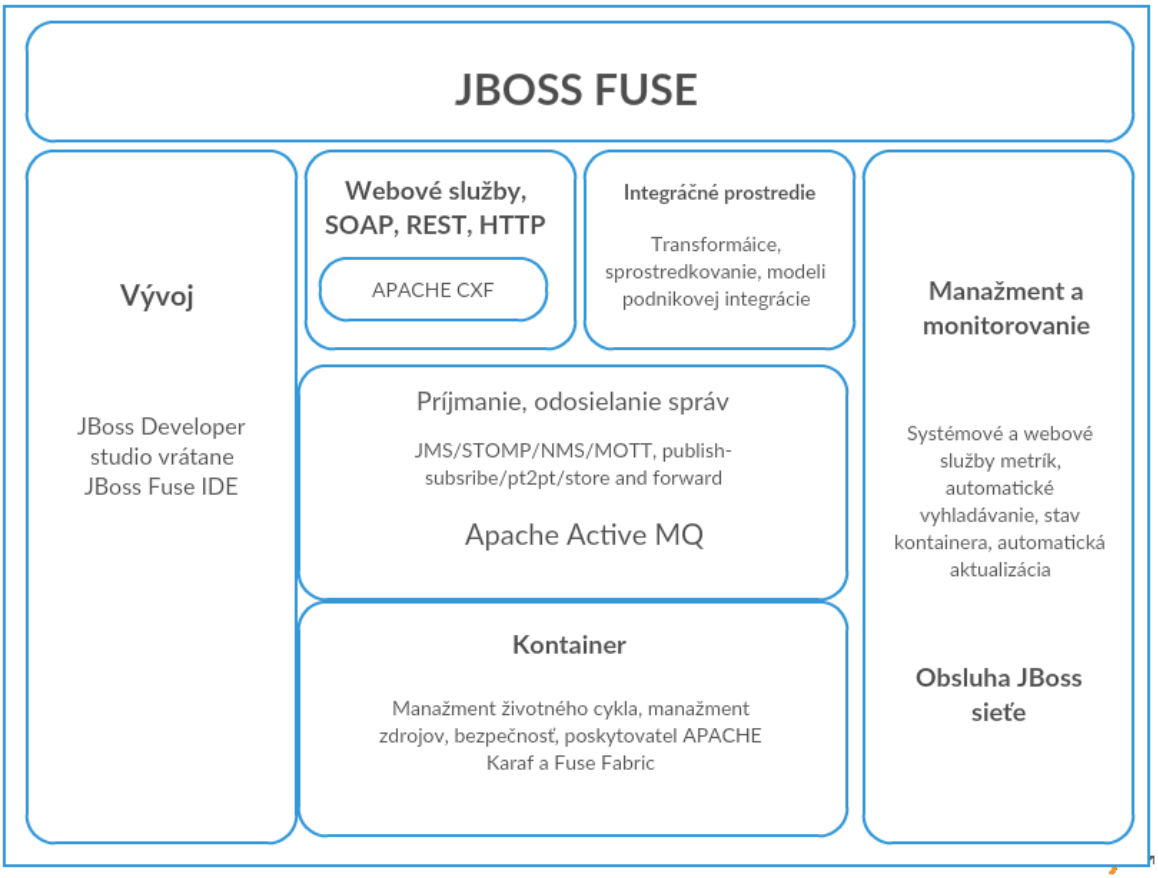
Tabuľka 1 Rozdelené možnosti podnikovej zbernice služieb [7]

[6] Návrhové vzory webových služieb pre zasielanie a príjem správ
[7] Software slúžiaci na prepojenie iných softwarových komponentov
[8] Služba opisujúca protokol pre spoľahlivé dodávanie SOAP správ
[9] Forma zabezpečenia dát pri ich smerovaní
[10] Metóda smerovania správ

3 INTEGRAČNÁ PLATFORMA RED HAT JBOSS FUSE

„Rad Hat Jboss Fuse“ je voľne dostupná podniková zbernica, ktorá dokáže zjednodušiť spojenie rôznych aplikácií, služieb a zariadený pre komplexné a efektívne riešenia. Táto integračná platforma je špecifická tým že obsahuje populárne a univerzálne integračne rozhranie apache camel o ktorom bude viac povedané v kapitole 3.1.2 . Taktiež umožňuje implementáciu najbežnejšie používaných modelov podnikovej integrácie. Disponuje s rôznymi integračnými modelmi a s viac ako 150 komponentami ktoré je možné na podnikovú zbernicu pripojiť a nasledovne ich používať.

Podniková zbernica Jboss Fuse podporuje integráciu v rámci rozšíriteľných podnikových aplikácií, na mobilných aplikácií alebo v cloude. Niektoré integračné riešenia vyžadujú komplexnejšie integračné schopnosti, zatiaľ čo iné zas môžu vyžadovať jednoduchšie a ľahko spracovateľné a samozrejme sú tu také ktoré vyžadujú oboje. Podniková zbernica Rad Hat JBoss Fuse môže byť nasadená a ľahko spracovateľná v každej konfigurácií. Taktiež je možné zvoliť pre rôzne koncové body rôzne konfigurácie.[9]. Integračná platforma JBoss Fuse je zložená z viacerých komponentov ktoré sú zobrazené na Obr. 18



Obr. 18 Komponenty integračnej platformy JBoss Fuse

3.1 Komponenty integračnej platformy JBoss Fuse

3.1.1 Kontajner

Schopnosť podnikovej zbernice zjednodušiť spojenie rôznych aplikácií je vo veľkej miere spôsobené znakmi kontajnera. Kontajner je vo svojej podstate fyzickým prejavom abstraktného koncového bodu a poskytuje implementáciu rozhrania služby. S tohto dôvodu býva často kontajner označovaný v odbornej literatúre ako službový kontajner. Môžeme ho tiež definovať ako vzdialený proces, ktorý je schopný hostiť komponenty používaného integrovaného softvéru.[10]

Službový kontajner spravuje teda životný cyklus danej služby a hlavne poskytuje službe rozhranie, pomocou ktorého sa daná služba dokáže pripojiť na komunikačnú zbernicu.

Jedným zo základných stavebných prvkov podnikovej zbernice JBoss Fuse je práve službový kontajner. Zbernica využíva kontajner typu Apache Karaf pričom je rozšírený funkcionalitou Fuse Fabric ktorá poskytuje rozhranie na jednoduchú správu kontajnerov.

3.1.2 Integračné rozhranie Apache Camel

Integračné rozhranie je navrhnuté tak aby sme boli schopný používať štandardnú metódu zápisu a jazyk na vysokej úrovni, pričom je možné prejsť s vizuálneho diagramu na implementáciu s minimálnou potrebou písania zdrojového kódu. To nám zabezpečuje integračné rozhranie Apache Camel v kombinácií s vývojovým prostredím JBoss Developer Studio, ktoré však bude detailnejšie popísane v kapitole č. 4.

Apache Camel predstavuje nástroj pre definícií vlastných pravidiel smerovanie správ dát služieb vo webovom rozhraní. Pričom je vďaka tomuto nástroji možné napríklad určiť od akých zdrojov prijímať správy, určiť ako ich ďalej spracovať a odosielať ďalej do iných destinácií. Jedným zo základných princípov Apache Camel je že nemá určené žiadne predpoklady ohľadne toho aké typy dát môže spracovať. Tento fakt dáva vývojárovi príležitosť integrovať akýkoľvek systém bez potreby premeny dát do kanonického tvaru.

Camel tiež ponúka abstrakcie na vyššej úrovni, ktoré umožňujú komunikovať s rôznymi systémami používajúce rovnaké rozhranie bez ohľadu na protokol alebo dátové typy, ktoré systémy používajú. Camel umožňuje podporu viac ako 80 protokolov a dátových typov. Rozšíriteľná a modulárna architektúra umožňuje tiež implementovať a bez problémov za-

pojiť podporu pre vlastné protokoly. Na základe týchto vlastností je eliminovaná potreba zbytočných konverzií čo robí integračné rozhranie Apache Camel rýchlejším.

Avšak je potrebné spomenúť čím Camel nie je. Camel nie je podniková zbernica služieb, aj keď v niektorých odborných článkoch vďaka svojej podpore smerovania, transformácie a monitorovania je spomínaná ako ľahká verzia podnikovej zbernice. Camel nedisponuje s kontajnerom a zbernicou na prijímanie a odosielanie správ avšak môže byť na takýto kontajner umiestený.[11]

3.1.3 Štandardy webových služieb

Nástroje ktoré slúžia na premenu ľubovoľnej aplikácie alebo systému na webovú službu. A to tým spôsobom aby bolo možné ju zahrnúť do servisne orientovanej architektúry. Funkcia aktivácie služieb je založená na technológií Apache CXF.

3.1.3.1 Apache CXF

Apache CXF voľne dostupné rozhranie, ktoré slúži pre vývoj webových služieb napríklad SOAP alebo RestFull, XML/HTTP. Pomáha pri budovaný a vývoji služieb, ktoré používajú API pre programovanie front-end. Apache CXF je najrozšírenejšie rozhranie pre vývoj webových služieb. Medzi najnovšie. V porovnaný s inými rozhraniami vyžaduje menej programovania a tiež je plne oddeľuje front-end od JAX-WS[11] kódu.

3.1.3.2 Rest webové služby

Rest (Representational State Transfer) je architektonický štýl v ktorom sa údaje a funkcie považujú za zdroje. Tieto webové služby sú ľahko škálovateľné a udržiavateľné a často používajú API pre webové aplikácie. Architektonický štýl Rest obmedzuje architektúru na architektúru klient / server a je navrhnutý tak, aby používal komunikačný protokol bez štátnej príslušnosti ako napríklad http protokol.

3.1.4 Spoľahlivé zasielanie správ

Spoľahlivé zasielanie správ v integračnej platforme JBoss Fuse je zabezpečené pomocou štandardného sprostredkovateľa ktorý je v odbornej literatúre označovaný ako message broker založeného na technológie Apache ActiveMQ.[17]

[11] Programovací jazyk Java pre programovanie webových služieb

3.1.4.1 Apache ActiveMQ

Apache ActiveMQ je teda voľne dostupný message broker písaný v programovacom jazyku Java, podporujúci JMS teda Java Message Service. Poskytuje podnikové funkcie, čo znamená že podporuje komunikácie od viac ako jedného klienta alebo servera.

ActiveMQ využíva niekoľko režimov pre vysokú dostupnosť a to v rátane mechanizmu uzamykania na úrovni súborov a databáz na úrovni riadku, zdieľanie úložiska prostredníctvom zdieľaného súborového systému alebo skutočnú replikáciu pomocou aplikácie Apache ZooKeeper. Robustný horizontálny mechanizmus škálovania nazývaní tiež „*Network of brokers*“ alebo teda sieť sprostredkovateľov.[16]

3.1.5 Integračný nástroj JBoss Developer Studio

JBoss Developer Studio je zoskupenie vývojových nástrojov založených na platforme eclipse, ktoré sú vopred nakonfigurované. Vďaka vopred nakonfigurovaným nástrojom je JBoss Developer Studio navrhnuté tak aby zvýšil produktivitu vývojára pri vývoji aplikácií. Môže sa tak sústrediť na vytváranie, testovanie a nasadzovanie aplikácií. Taktiež JBoss Developer studio disponuje množstvom funkcií, ktoré dokážu značne pomôcť pri vývoji aplikácií.

3.1.5.1 Funkcie integračného nástroja JBoss Developer Studio

- Vývoj nových aplikácií pomocou sprievodcov a projektových príkladov spoločnosti JBoss Central
- Možnosť jednoduchého pridania výkonných funkcií aplikáciám pomocou nástroja Forge.
- Jednoduché *vytvárania* webového rozhrania pomocou nástrojov vizuálne úpravy *drag-and-drop*
- Automatické aktualizovanie prehliadačov pomocou nástroja LiveReload
- Náhľad a testovanie mobilných webových aplikácií na rôznych simulačných mobilných zariadeniach pomocou programu BrowserSim
- Nástroje JBoss Server Tools a OpenShift, ktoré umožňujú nasadzovať aplikácie na JBoss servery a cloud.

3.1.5.2 Využitie integračného nástroja JBoss Developer Studio

JBoss Developer Studio pomáha vývojárom, ktorý vyvíjajú na platforme JEE s integráciou technológie JBoss a API do jedného vývojového prostredia. Nasledovný text je venovaný tomu ako tento integračný nástroj dokáže uľahčiť prácu vývojára. Bližšie ukázané a prakticky popísané bude toto prostredie neskôr v praktickej časti práce.

- Webové aplikácie

JBoss Central poskytuje sprievodcov, ktorý vytvárajú základnú kostru a vzorové projekty. Umožňujú tým vývojárom zamerať sa na vývoj funkčnosti aplikácií. Sprievodcovia vytvárajú webové aplikácie založené na rôznych API a technológiách, ktoré ukazujú využitie a výhody každého z nich. JBoss Developer Studio ponúka šablóny projektov v populárnych programovacích jazykoch ako HTML, XHTML, JSF a tak ďalej.

Rôzne palety aplikácií umožňujú prístup k základným prvkom rozhrania JSF, Richfaces a Seam API, ktoré sa používajú pri vývoji užívateľských rozhraní aplikácií. Prvky týchto rozhraní je možné presunúť a zahrnúť ich priamo do nami vytvoreného projektu. Visual Web Tools ponúka grafické a zdrojové prezeranie súborov a predvolené nastavenia špeciálnych editorov pre rôzne typy súborov. Tiež podporuje špecifikáciu jazyka Java EE 6 a poskytuje tým nástroje pre API rozhrania JAX-RS, Hibernate a CDI, takže je možné bez väčšej námahy vyvíjať komponenty aplikácie na strane servera.

- Webové aplikácie optimalizované pre mobilné zariadenia

Mobilné nástroje ktorú sú obsiahnuté v vývojovým prostredí JBoss Developer studio poskytujú podoporu pre programovací jazyky HTML5 a JQuery Mobile, ktoré umožňujú vytvárať optimalizované webové aplikácie pre klientov s osobnými počítačmi či mobilnými zariadeniami. Sprievodca projektom HTML5 v spoločnosti JBoss Central generuje vzorovú aplikáciu s využitím technológií HTML5 a JQuery Mobile. Pomocou dostupných šablón umožňuje efektívne a rýchle zrealizovanie navrhovanej aplikácie.

Disponuje tiež programom BrowserSim, ktorý umožňuje odkášať si vyvíjané aplikácie na rôznych simulovaných mobilných zariadeniach. Tento program je v kombinácií s funkciami ako Firebug Lite a Wiener silný prostriedok pre kontrolovanie a odlaďovanie zdrojových kódov.

- Aplikácie pre cloudové riešenia

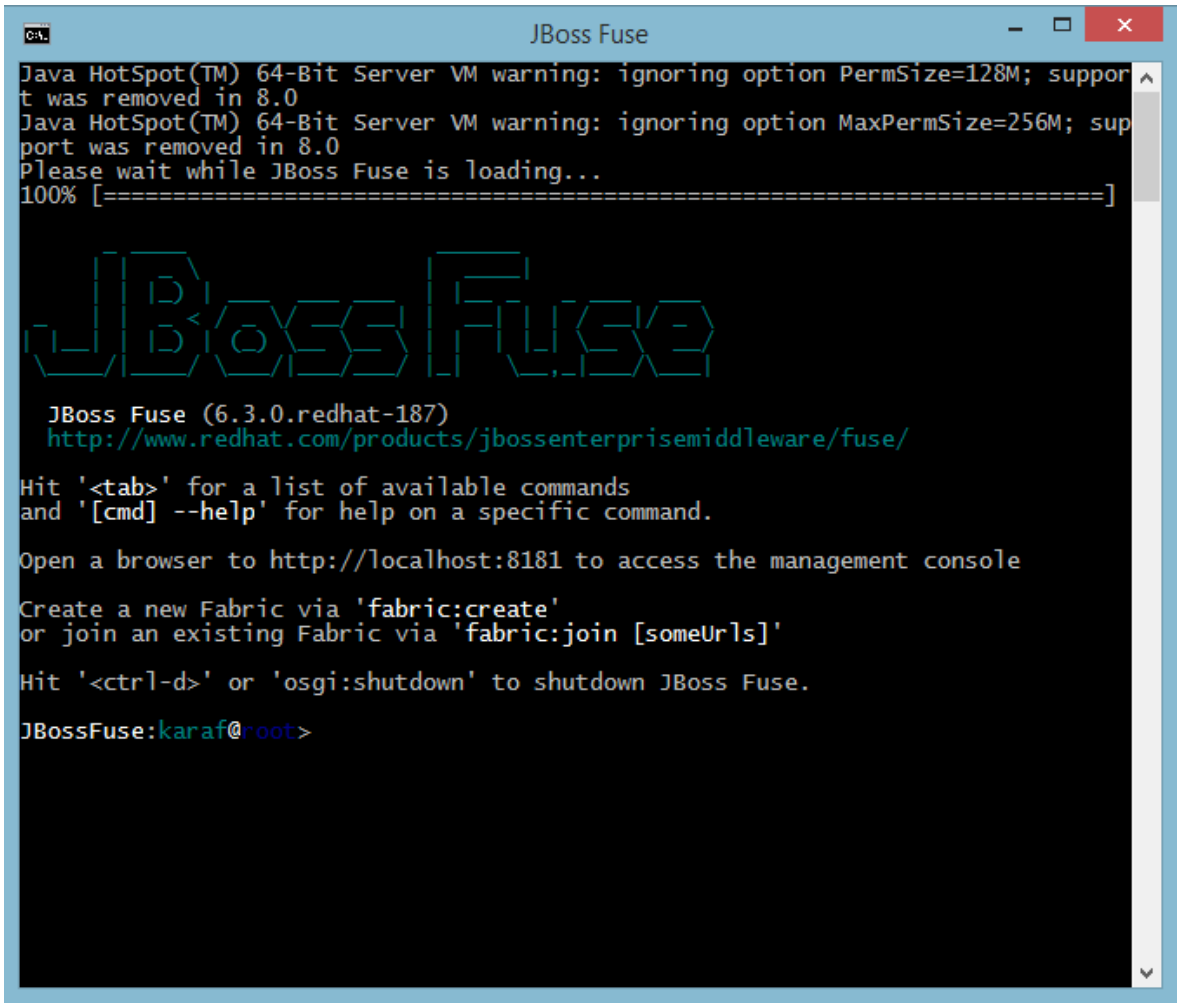
Prostredníctvom nástroja OpenShift dokážeme priamo nasadiť na cloud aplikácie vyvíjané na platforme Red Hat . K dispozícií je možno vytvorenia si vlastného účtu OpenShift čím sa vyskytuje možnosť spravovať aplikácie. Tak ako aj v predchádzajúcich prípadoch je tu možnosť používania sprievodcu aplikáciou OpenShift na vytvorenie a nasadenie nových aplikácií. OpenShift Tools nám však dokážu tiež importovať už nasadené aplikácie v takom stave aby bolo možné ďalej rozvíjať a spravovať ich nasedenie.[12]

II. PRAKTICKÁ ČASŤ

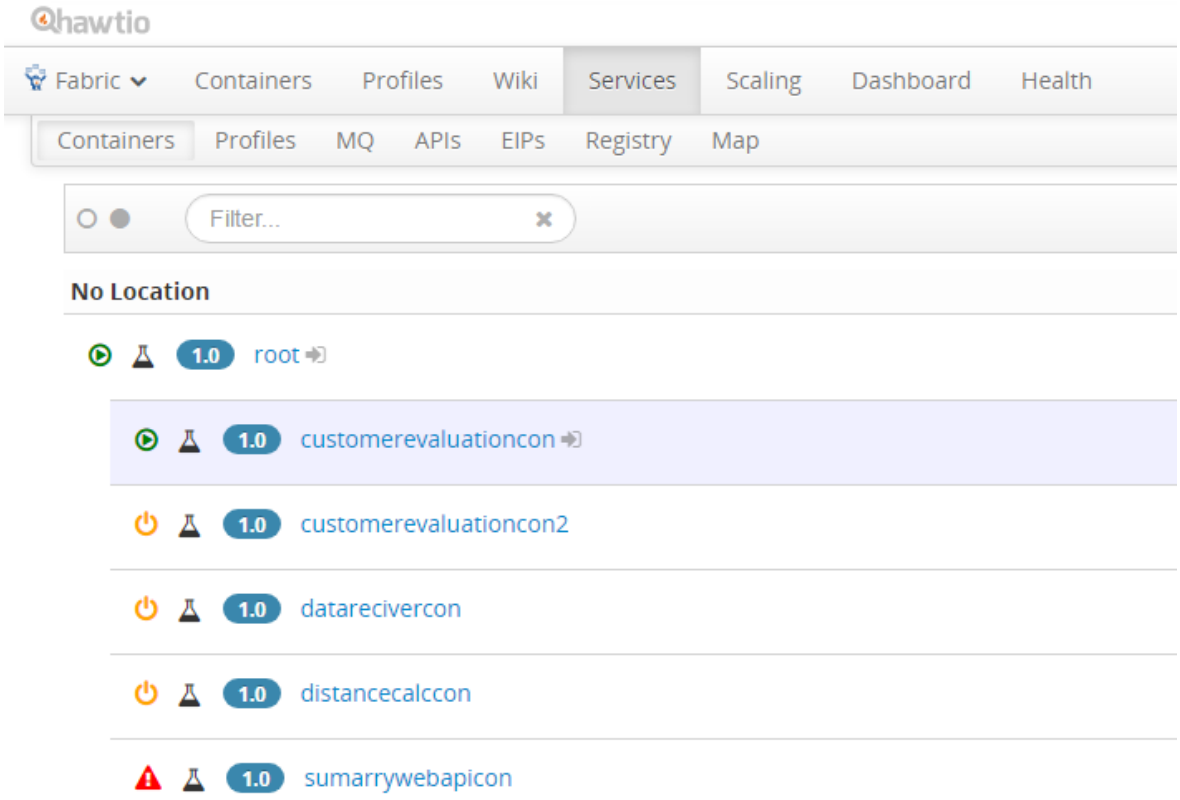
4 INŠTALÁCIA A NASTAVENIE VIRTUÁLNEHO PROSTREDIA PRE REALIZÁCIU PODNIKOVEJ ZBERNICE SLUŽIEB

Pod pojmom virtuálne prostredie si môžeme prestaviť lokálne prostredie do ktorého sme schopný umiestňovať naše integračné riešenia vytvárať či spravovať kontajneri a tiež ich monitorovať. V praktickej časti je používaná konzolová aplikácia pre správu JBoss Fuse, tá poskytuje centrálne rozhranie pre správu a konfiguráciu objektov umiestnených na JBoss Fuse. Aplikácia je tiež spôsobilá ku konfigurovaniu a nasadzovaniu kontajnerov. Sme schopný tiež monitorovať procesy na podnikovej zbernici JBoss Fuse, vykonávať aktualizácie a ovládať jednotlivé služby.

Aplikáciu pre správu JBoss Fuse môžeme ovládať pomocou konzolovej aplikácie, ktorá je zobrazená na Obr. č 19 alebo pomocou grafického rozhrania hawtio ktorého ukážke je zobrazená na Obr. 20



Obr. 19 Ukážka konzolového prostredia



Obr. 20 Ukážka grafického rozhrania hawtio

4.1 Inštalácia JBoss Fuse

Ako už bolo spomínané JBoss Fuse je voľne dostupný integračný nástroj. Tento integračný nástroj je možné stiahnuť s oficiálnych stránok Red Hat. Po stiahnutí zložku .zip súbor rozbalíme. Ako prvý krok je potrebné otvoriť súbor ktorý sa nachádza v zložke etc s názvom „users.properties“. V súbore od komentujeme posledný riadok teda s pôvodného kódu Kód. 3 dostaneme Kód. 4.

```
#admin=admin,admin,manager,viewer,Monitor,Operator,Maintainer,Deployer,Auditor,
Administrator,SuperUser
```

Kód. 3 Súbor users.properties pred zmenou

```
admin=admin,admin,manager,viewer,Monitor,Operator,Maintainer,Deployer,Auditor,
Administrator,SuperUser
```

Kód. 4 Súbor users.properties po zmene

4.1.1 Vytvorenie a nastavenie fabriky

Jedným s komponentov integračnej platformy JBoss Fuse je takzvaná JBoss Fuse Fabric8, ktorá slúži pre správu a kontajnerov v ktorých sú umiestené jednotlivé služby. Vytvoríme ju spustením konzolovej aplikácie umiestenej v zložke bin s názvom fuse.bat. Po spustený sa pustí konzolová aplikácia, ktorej ukážka je zobrazená na Obr. 19. Do aplikácie napíšme nasledovný príkaz :

```
fabric:fabric:create --zookeeper-password admin
```

Kód. 5 Vytvorenie fabriky

Tým bola teda vytvorená fabrika v ktorej budú neskôr vytvárané jednotlivé kontajnery obsahujúce potrebné služby.

Posledným krokom k úspešnému vytvoreniu a nastaveniu virtuálne prostredia pre podnikovú zbernicu služieb je nastavenie JBoss Fuse tak aby bolo možné s integračného nástroju JBoss Developer Studio umiestniť do virtuálneho prostredia fabriky vyvíjané služby. Toto nastavenie vykonáme v súbore settings.xml, ktorý je umiestení v ceste c:\users\{uživatel}\.m2\settings.xml. Do daného súboru pridáme Kód.6.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<settings xmlns="http://maven.apache.org/SETTINGS/1.1.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/SETTINGS/1.1.0
http://maven.apache.org/xsd/settings-1.1.0.xsd">
  <servers>
    <server>
      <id>fabric8.upload.repo</id>
      <username>admin</username>
      <password>admin</password>
    </server>
  </servers>
</settings>
<plugins>
  <plugin>
    <groupId>io.fabric8</groupId>
    <artifactId>fabric8-maven-plugin</artifactId>
  </plugin>
</plugins>
```

Kód. 6 Nastavenie súboru seetings.xml

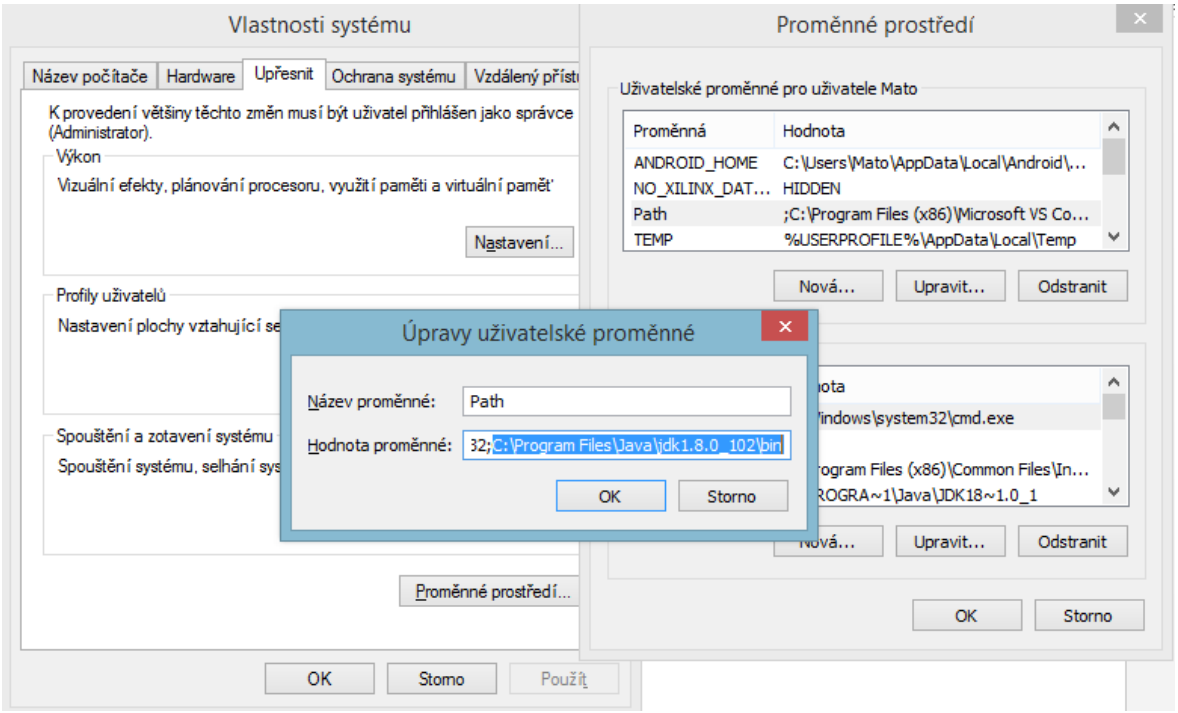
Pri inštalácií virtuálne prostredia do fáze aby bolo možné navrhnúť podnikovú zbernicu služieb je potrebné tiež nastaviť systémové premenné a nainštalovať vývojové prostredie JBoss Developer studio aj s potrebnými komponentami.

4.2 Konfigurácia systémových premenných

Pred nastaveným systémových premenných je potrebné stiahnuť súpravu pre vývoj v platforme Java teda Java Development Kit (JDK). Je to implementácia platformy Java vydaná spoločnosťou Oracle.

Produkt je voľne dostupný priamo s oficiálnych stránok výrobcu Oracle. Po stiahnutí je potreba Oracle JDK nainštalovať do štandardnej zložky C:\Program Files:\Java\jdk…

Po inštalácií nasleduje nastavenie už spomínaných systémových premenných prostredia operačného systému windows. Premenné sa definujú v okne s názvom „*Premenné Prostredia*“ nájdeme ho vo vlastnostiach systému pod zložkou upresniť. Nasledujúci krok spočíva v pridaný novej premennej s názvom „*Path*“. Nastavíme hodnotu tejto premennej na súborovú cestu nainštalovanej Java Development Kit. Pokiaľ však takáto premenná je vytvorená stačí k jej hodnote pridať už spomínanú súborovú cestu a oddeliť ju bodkočiarkou viac Obr. 22



Obr. 21 Nastavenie systémových premenných

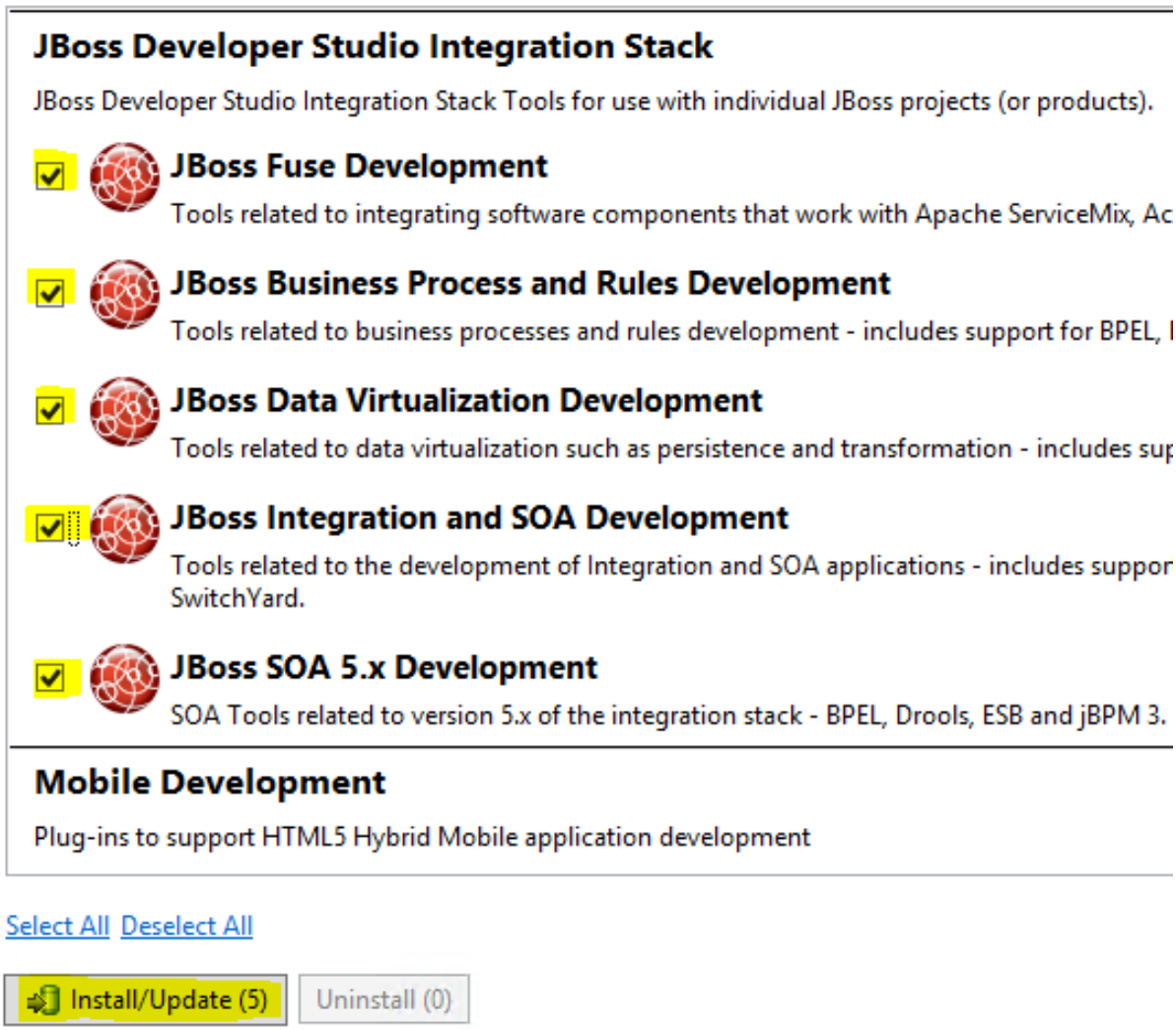
Rovnakým spôsobom je potrebné pridať systémovú premennú s názvom „JAVA_HOME“, ktorej hodnota bude predstavovať súborovú cestu JDK ako v predchádzajúcom prípade avšak bez zložky bin.

4.3 Inštalácia vývojového prostredia JBoss Developer Studio

Ako už bolo spomínane v teoretickej časti JBoss Developer studio je vývojové prostredie určenie tiež k vývoji integračných prostriedkov v integračnej platforme JBoss Fuse bez ktorého by nebolo možné vytvorenie jednotlivých služieb ich umiestenie na kontajneri a ich konfigurácia čo znamená teda že je neoddeliteľnou súčasťou prostredia pre vývoj podnikovej zbernice služieb. JBoss Developer Studio je voľne dostupné vývojové prostredie, ktoré je dostupné na jeho oficiálnych stránkach. [12].

Po stiahnutý dvojklikom na devstudio.jar pre štart inštalácie. Pri kroku 4 inštalácie „Select Java VM“ je potrebné zmeniť Java VM na nainštalované Java Development Kit ďalej pokračujeme s predvoleným nastavením.

Po nainštalovaný a úspešnom prvom spustení je potrebné stiahnuť a doinštalovať niektoré komponenty potrebné pre vytváranie JBoss Fuse projektov. Po spustený sa teda preklikneme na „Sotware/Update“ ikonu umiestnenú v základnom okne. Zaškrkneme v pravom dolnom rohu „Enable Early Access“ a následne zaškrkneme všetky položky „JBoss Developer studio Integration stack“. Položky ktoré je potrebné stiahnuť a nainštalovať sú zobrazené na Obr. 23



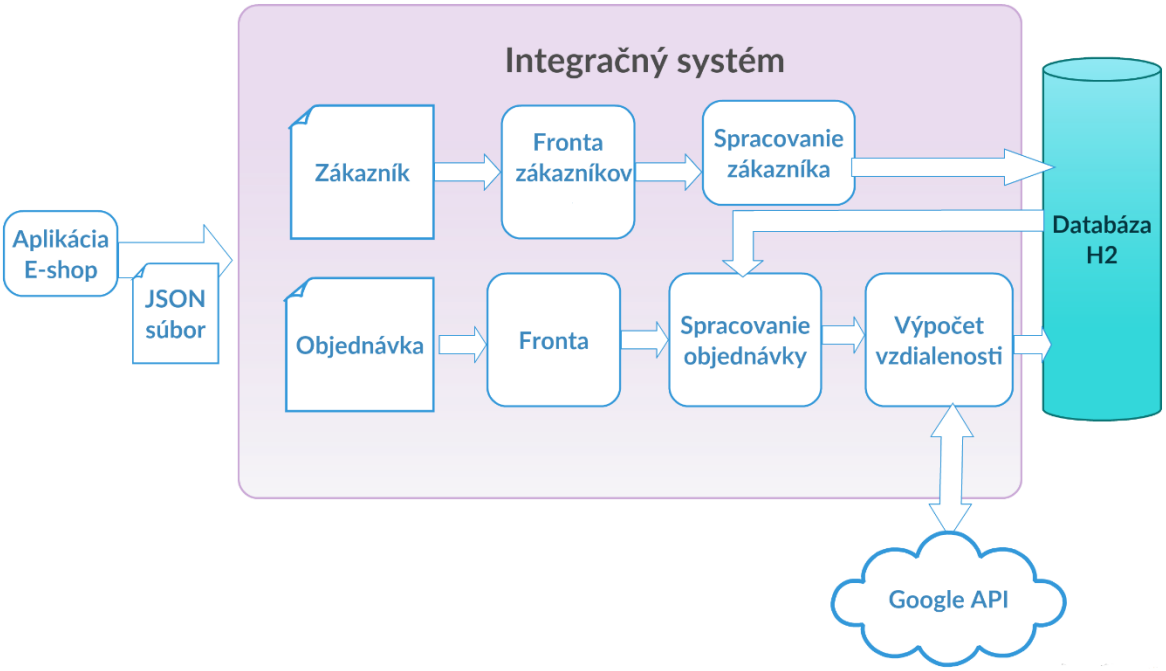
Obr. 22 JBoss Developer Studio Integration stack

Vybrané komponenty nainštalujeme jednoducho kliknutím na vyznačenú ikonu a následne preklikáme sprievodcu inštaláciou. Posledným krokom k správnemu fungovaniu vývojového prostredia je JBoss Developer Studio je reštartovanie aplikácie. Tým pádom je pripravené k používaniu na vývoj podnikovej zbernice služieb integračnou platformou JBoss Fuse.

5 NÁVRH PODNIKOVEJ ZBERNICE SLUŽIEB

5.1 Stručný popis navrhovaného riešenia

Integračné riešenie pracuje s aplikáciou, ktorá je navrhnutá ako internetový obchod ktorý posiela na podnikovú zbernicu rôzne údaje vo formáte JSON. Konkrétne sú to údaje pri registrovaní nového zákazníka, kedy posiela na komponent JBoss Fuse ActiveMQ správu s údajmi o zákazníkovi ako sú meno, priezvisko, vek, adresu a id zákazníka. Taktiež posiela správu pri vykonaní novej objednávky kedy sú poslané základné údaje o danej objednávke ako je čas vykonania objednávky, typ objednávaného produktu, základne informácie o produkte, meno produktu, špecifické id zákazníka, ktorý túto objednávku vykonal a id danej objednávky. Podniková zbernica rozozná typ správy teda či ide o správu o objednávky alebo o registráciu nového zákazníka a na základe toho ďalej odloží správu do fronty určenej pre zákazníka alebo objednávku. Nasleduje krok kedy podniková zbernica zoberie danú správu s fronty a spracuje ju. Pri registrácií zákazníka ju spracuje do vhodného tvaru a uloží nového zákazníka do externej databázy. U objednávky taktiež spracuje správu do vhodného tvaru, na základe id zákazníka vyberie s externej databázy adresu zákazníka. Pomocou adresy a pomocou externých google API zistí súradnice zákazníka. V čase keď už je známa poloha zákazníka je možné, znova pomocou externých google API zistiť polohu najbližšej kamenej predajne v okolí kde si zákazník môže vyzdvihnúť objednaný tovar. Následne sa vypočíta vzdialenosť od bydliska zákazníka. Daná vzdialenosť je pridaná k objednávke a uložená do externej databázy. Integračný návrh je zobrazený na Obr.23 pričom jednotlivé časti budú bližšie popísanie v ďalšej časti práce. V konečnom dôsledku je prepojená aplikácia, ktorá založená na frameworku .NET, MySQL s databázou typu H2 a s troma rozličnými google API a to všetko je konfigurované a transformované vo vývojovom prostredí JAVA.



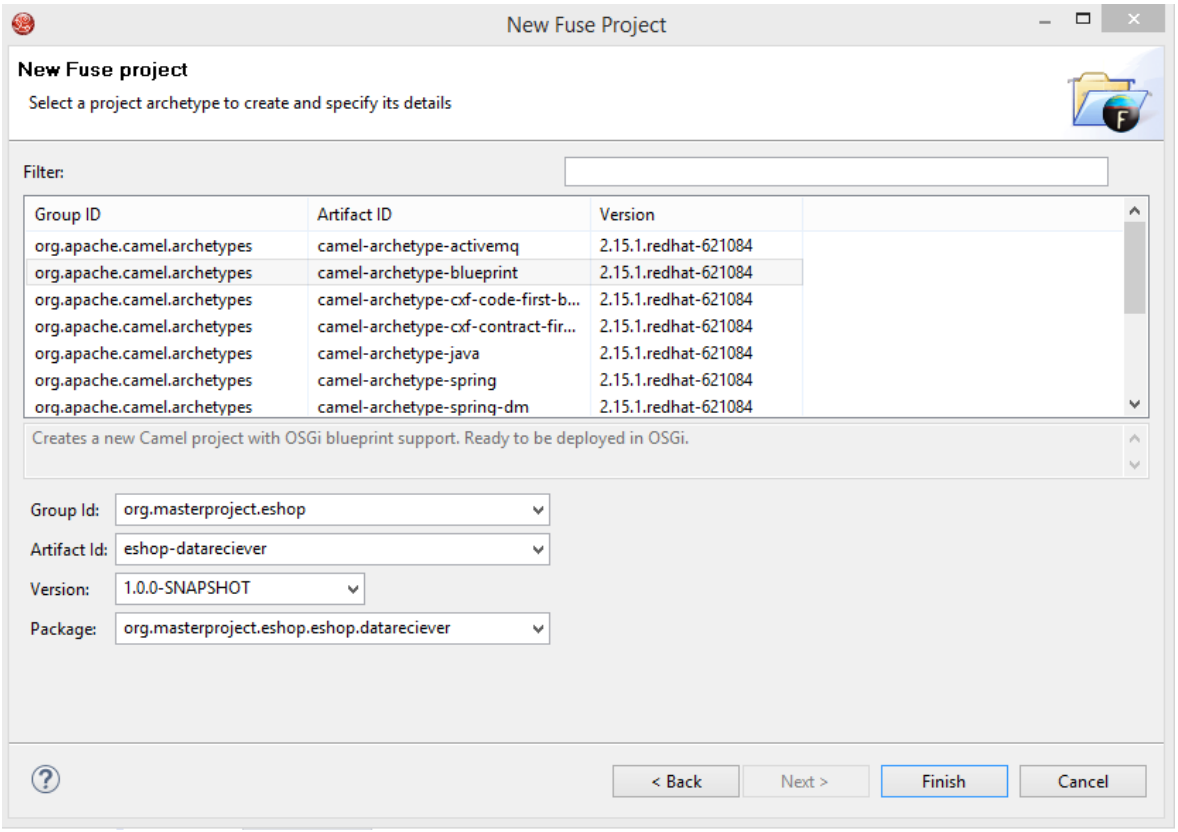
Obr. 23 Navrh integračného riešenia

5.2 Popis jednotlivých služieb integračného systému

Celý integračný systém je rozdelený do viacerých projektov (alebo tiež služieb) pričom každá vykonáva inú činnosť. Tieto služby sú následne implementované do vlastných kontajnerov v JBoss Fuse fabrike. V nasledujúcej časti budú jednotlivé projekty opísané.

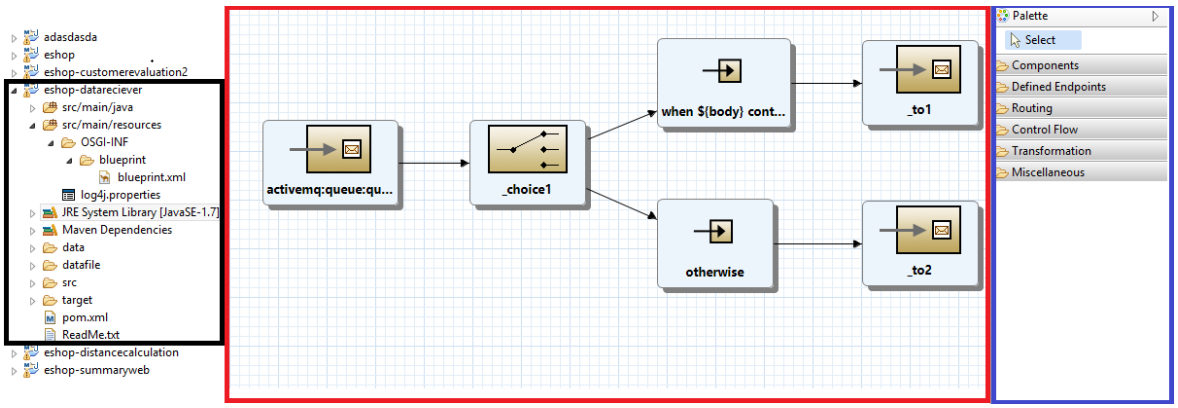
5.2.1 Prijímanie dát

Ako prvé bolo potrebné vytvoriť nový Fuse projekt vo vývojovom prostredí JBoss Developer Studio. Pri vytváraní nového projektu je potrebné zvoliť Group Id, Artifact Id, verziu a balík. Pri tvorbe viacerých projektov ktoré spolu budú spolupracovať je potrebné zvoliť rovnaké Group Id, verziu a názov balíku pre všetky súvisiace projekty. Artifact Id tvorí jedinečné Id projektu, ktoré môže byť rôzne. Na Obr. 24 je znázornená použitá konfigurácia pri vytváraní jednotlivých projektov.



Obr. 24 Ukážka vytvárania projektu datareciver

Po úspešnom vytvorený projektu ktorý slúži ako služba podnikovej zbernice sa nám vygeneruje súborová štruktúra tak ako je zobrazená modrým ohraničením na Obr. 25. Vygeneruje sa tiež súbor s názvom pom.xml. Je to konfigurační súbor v ktorom definujeme používané komponenty ich verzie a tiež repozitáre odkiaľ budú komponenty sťahované. Obsah tohto súboru závisí od škály použitých komponentov. Modrým ohraničením je vyznačená paleta kde sú umiestnené smerovacie, transformačné prvky, a užívateľom definované koncové body. Tieto prvky ktoré sú k dispozícií v palete sú následne umiestňované do priestoru vyznačeného červeným ohraničením. V tomto priestore sú jednotlivé komponenty prepájané a vizualizujú tým tok dát integračným systémom. Jednotlivé komponenty tvoria ako celok zdrojový kód písaný v jazyku XML. Vývojové prostredie dovoľuje tvoriť integrační systém priamo v programovacom jazyku XML alebo pomocou grafického rozhrania aké je zobrazené na Obr.25



Obr. 25 Ukážka služby prijímač dát

Na Obr. 25 môžeme vidieť že tok dát sa začína prijatím správy ktorá má nasledovnú štruktúru:

```
{
        "CustInfo":"EshopCustomer",
        "custInfoId":"1",
        "firstName":"meno",
        "lastName":"priezvisko",
        "age":"45",
        "address":"adresa",
        "orderId":"4"
}
```

Kód. 7 Obsah prijatej správy

Štruktúru ktorá je zobrazená vyššie príjme komponent ActiveMQ ktorého definícia v jazyku XML je nasledovná:

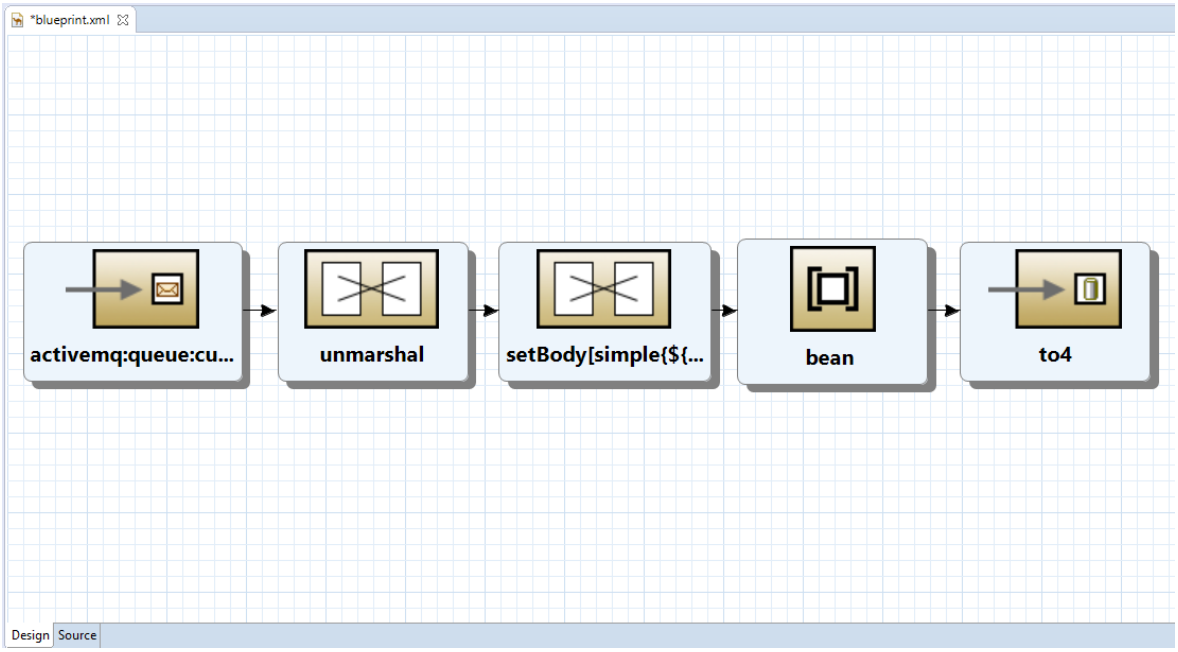
```
  <bean id="activemq"
class="org.apache.activemq.camel.component.ActiveMQComponent">
      <property name="brokerURL" value="tcp://localhost:61617"/>
      <property name="userName" value="admin"/>
      <property name="password" value="admin"/>
  </bean>
```

Kód. 8 Komponent príjamnie dát s ActiveMQ v jazyku XML

Prijaté dáta ďalej smerujú do komponentu, ktorý ich rozdelí na základe toho či obsahujú reťazec EshopCustomer. Ak áno smeruje správa do fronty určenej pre zákazníka a naopak ak tento reťazec správa neobsahuje smeruje do fronty určenej pre objednávku. Tým máme zaobstarané prijatie a roztriedenie správ.

5.3 Spracovanie zákazníka

Ďalšou službou ktorá je súčasťou navrhovaného integračného mechanizmu je služba spracovanie zákazníka. V podstate ide o spracovanie dát týkajúcich sa registrácií zákazníka tak aby bolo možné s nimi pracovať a následne ich uložiť do externej databázy viac Obr. 26.



Obr. 26 Ukážka služby spracovanie zákazníka

Na Obr. 26 je vidieť že služba znova načúva fronte ActiveMQ avšak teraz je to fronta do ktorej sme v predchádzajúcej službe posielali dáta teda fronta zákazníka. Ako už bolo spomenuté dáta sa posielajú vo formáte JSON viac Kód.7. Avšak takýto formát správy je potrebné transformovať tak aby sme s daným súborom mohli ďalej pracovať na to nám slúži komponent JBoss Fuse umarshal. Ten poskytuje transformáciu dát zo širokej škály dostupných formátov. V tomto prípade bola použitá transformácia s formátu JSON na objekty typu Java. Definícia komponentu unmarshall v jazyku XML je nasledovná:

```
<unmarshal>
      <json library="Jackson"/>
 </unmarshal>
```

Kód. 9 Definiiícna komponentu unmarshall

V ďalšom kroku bolo potrebné naplniť telo správy získanými údajmi o zákazníkovi a to s preto aby bolo možné s údajmi pracovať ako s reťazcom dátového typu string čo je neskôr potrebné pre funkčnosť ďalšieho komponentu „*bean*“. Tento komponent predstavuje odkaz na vytvorenú metódu v programovacom jazyku Java viac Kód. 11. V metóde sa da-

ný reťazec spracuje tak a aby bolo možné dané dáta ukladať do externej databázy ukážka spomínanej metódy je uvedená v Kód.12. Posledný komponent na obrázku Obr.26 , ktorý je označení ako „to4“ je komponent vďaka ktorému je možné pripojiť sa do externej databázy v tomto prípade je to databáza typu H2. Tento komponent je definovaný v jazyku XML a uvedený v Kód. 13. Integračná platforma JBoss Fuse poskytuje možnosť prepojenia s viacerými typmi databáz.[15]

```
<bean method="toCustomer"
beanType=
"org.masterproject.eshop.eshop.customerevaluation2.ToCustomerClass"
/>
```

Kód. 10 Komponenta bean odkaz na Java metódu

```
package org.masterproject.eshop.eshop.customerevaluation2;

public class ToCustomerClass {

	public Customer toCustomer(String customer) {

		Customer customer_ = new Customer();
		String[] parsedCustomer = customer.split(",");
	customer_.setCustInfoId(Integer.parseInt(parsedCustomer[0]));
		customer_.setFirstName(parsedCustomer[1]);
		customer_.setLastName(parsedCustomer[2]);
		customer_.setAge(Integer.parseInt(parsedCustomer[3]));
		customer_.setAddress(parsedCustomer[4]);
		return customer_;
	}

}
```

Kód. 11 Formátovanie správy pomocou Java metódy

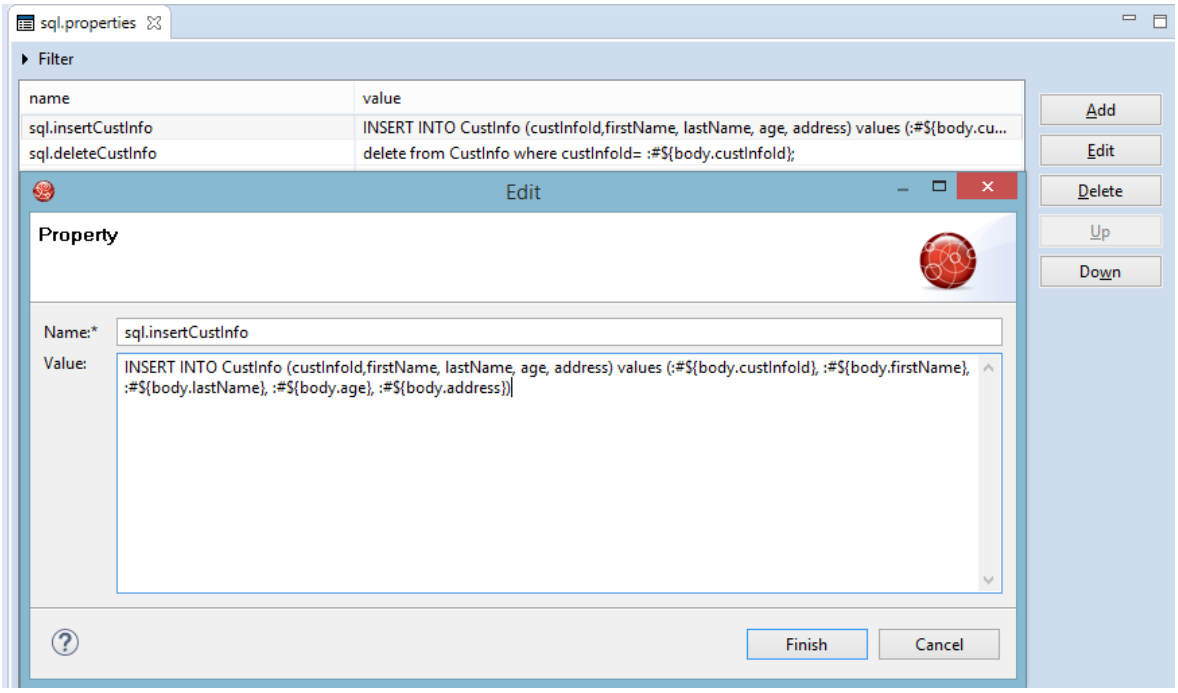
```
	<bean id="dataSource"
class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
	  <property name="driverClassName" value="org.h2.Driver"/>
<property name="url" value="jdbc:h2:file:~/h2/eshop;AUTO_SERVER=TRUE"/>
	  <property name="username" value="sa"/>
	  <property name="password" value=""/>
	</bean>

<!-- configure the Camel SQL component to use the JDBC data source -->
<bean id="sql" class="org.apache.camel.component.sql.SqlComponent">
	  <property name="dataSource" ref="dataSource"/>
	</bean>
```

Kód. 12 Komponent prepojenie s externou H2 databázou

Kód.11 a Kód.12 definuje danú komponentu avšak neuvádza aká činnosť bude s databázou uskutočňovaná. Preto bolo potrebné definovať súbor sql.properties. Tento súbor obsahuje definované príkazy pre prácu s databázou písane v jazyku sql. Príkazy je možné písať priamo do XML zdrojového kódu avšak vytvorením súboru sql.properties

a definovaným v ňom sql príkazov je docielené sprehľadnenie a jednoduchosť zdrojového kódu. Na Obr.27 je zobrazený obsah súboru sql.properties vo vývojovom prostredí JBoss Developer Studio.



Obr. 27 Obsah súboru sql.properties v službe spracovanie zákzníka

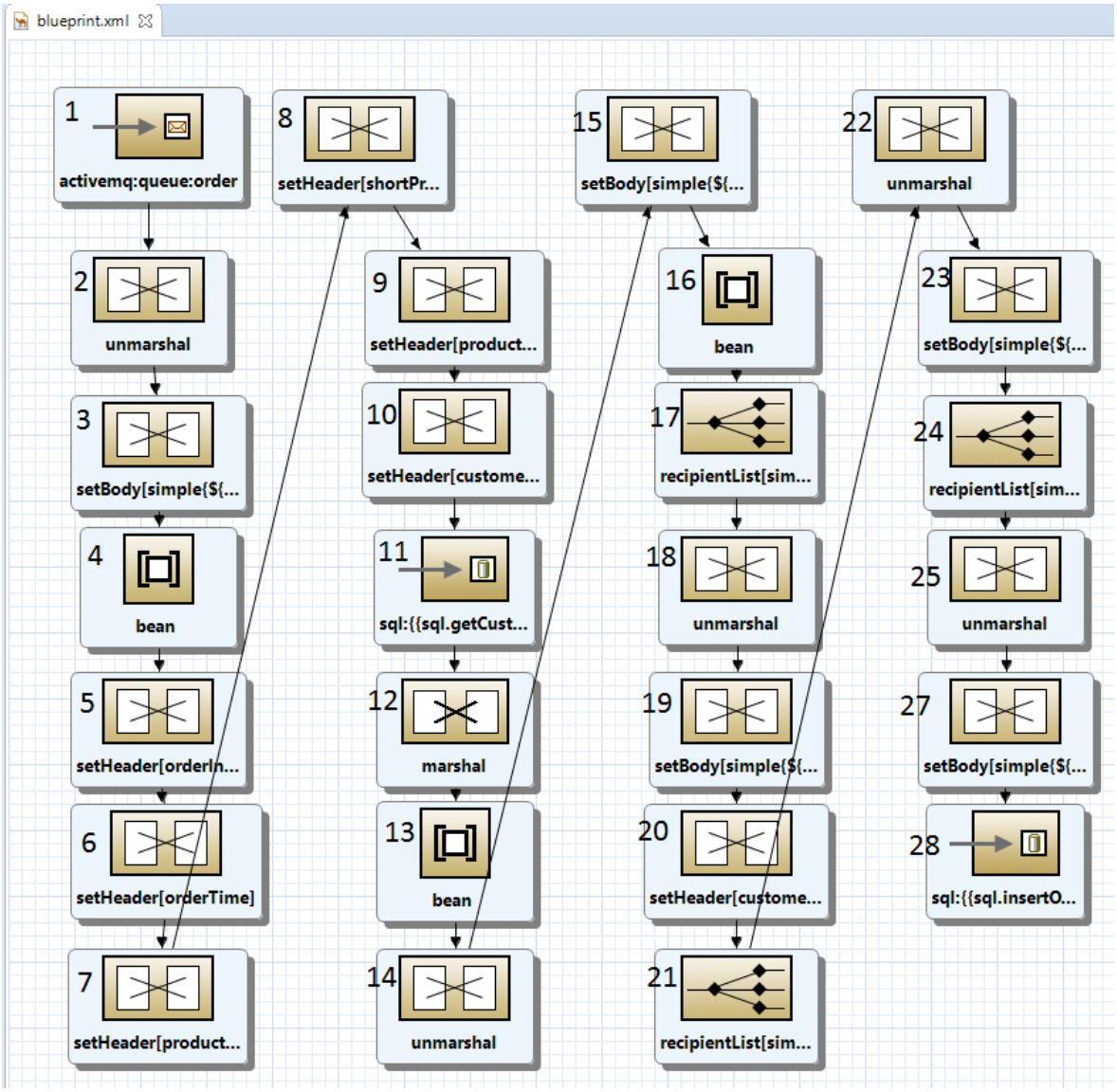
Na Obr. 27 je možné vidieť že hodnoty súboru sql.properties môžeme priamo vo vývojovom prostredí meniť upravovať, odstraňovať alebo pridávať. Tieto zadefinované sql príkazy je potrebné vykonať na definovanom pripojený do databáze ktoré je ukázané v Kód.12. Vykonanie týchto príkazov je definované v jazyku XML nasledovne :

```
<to
uri="sql:{{sql.insertCustInfo}}?alwaysPopulateStatement=true" id="to4"
/>
```

Kód. 13 Odkaz na vykonanie SQL príkazov v jazyku XML

5.4 Výpočet vzdialenosti

Poslednou službou obsiahnutou v navrhovanom integračnou systéme je služba, ktorá slúži na výpočet vzdialenosti bydliska zákazníka od najbližšej predajne internetového obchodu kde si môže objednávku zákazník vyzdvihnúť. Táto služba je zo všetkých najobsiahlejšia pretože sa v nej vykonávajú viaceré funkcie a integruje viac systémov.



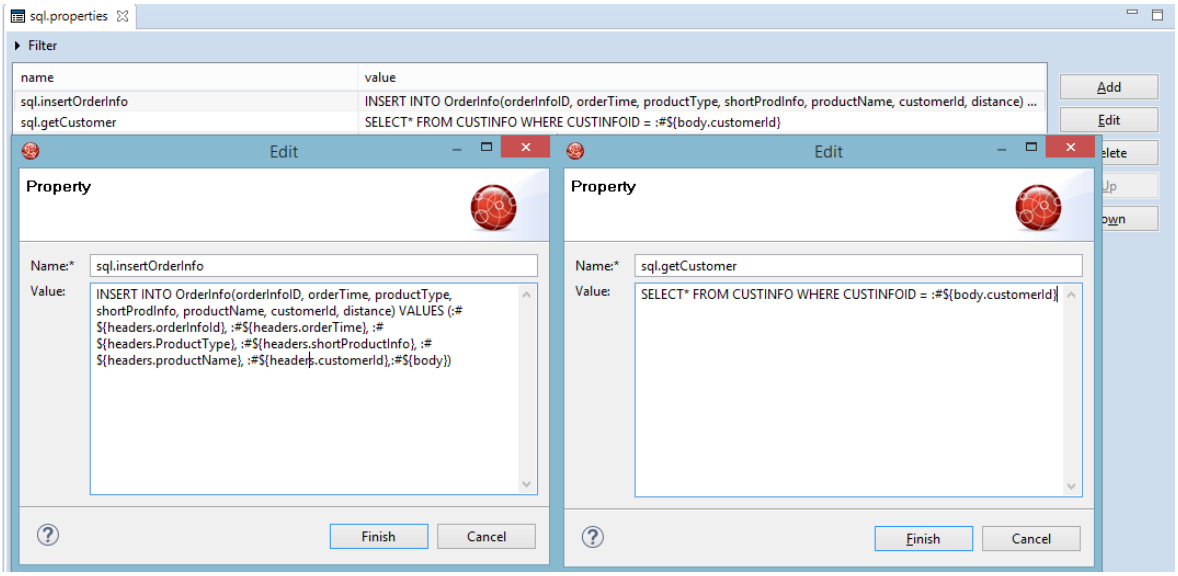
Obr. 28 Ukážka služby výpočet vzdialenosti

Na Obr. 28 je zobrazená štruktúra služby ktorá sa stará o výpočet vzdialenosti medzi bydliskom zákazníka a najbližšej predajni kde si zákazník môže vyzdvihnúť svoju objednávku. Jednotlivé komponenty sú očíslované pre lepší prehlaď pri vysvetľovaný ich činnosti v integračnom systéme. Služba, tak ako aj v predchádzajúcich prípadoch načúva fronte ActiveMQ. V tomto prípade načúva fronte objednávok. Začiatok služby je podobný službe pre spracovanie zákazníka, správa sa transformuje z formátu JSON spracuje vo vytvorenej Java metóde. Nasleduje komponent „setHeader“. Tento komponent slúži pre nastavenie hlavičky správy v danom prípade je použití pre uchovanie informácií o objednávke a to s dôvodu aby bolo možné s danými informáciami pracovať aj neskôr. Definícia tohto komponentu vyzerá nasledovne:

```
<setHeader headerName="productType">
      <simple>${body.productType}</simple>
</setHeader>
```

Kód. 14 Definícia komponentu setHeader v jazyku XML

Ďalším krokom je prístup do databáze za účelom získania dát o zákazníkovi, ktorý vytvoril objednávku. Dáta o zákazníkovi získame na základe zákazníkovho Id, ktoré je zahrnuté v správe o objednávke. Pripojenie k databáze je riešené rovnakým spôsobom ako tomu bolo pri službe spracovanie zákazníka. Zmenili sa však sql príkazy, ktorú sú definované znova v súbore sql.properties. Na Obr. 29 sú zobrazené sql príkazy ktoré boli využívané pri službe výpočet vzdialenosti.



Obr. 29 Ukážka súboru sql.properties v službe vypoceť vzdialenosti

Získané informácie o zákazníkovi v ďalšom kroku transformujeme do formátu JSON pomocou komponentu marshall jeho definícia v jazyku XML je nasledujúca :

```
<marshal>
      <json prettyPrint="true" library="Jackson"/>
</marshal>
```

Obr. 30 Definícia kompontu marhall v jayzku XML

Informácie ktoré sme dostali je nutné spracovať tak aby sme boli schopní pracovať s adresou zákazníka. To dosiahneme pomocou rôznych transformácií ktoré sú obsiahnuté v komponentoch 13 až 15. V komponente 16 s názvom „*bean*“ , prebieha parsovanie správy do nami požadovanej podoby teda do podoby kedy sme schopný dostať adresu. Metóda na ktorú komponent číslo 16 odkazuje vyzerá nasledovne :

```
public String GetAdrress(String order){
      String[] parsedOrder = order.split(",");
      String address = parsedOrder[4];
      address = address.replace(' ', '+');
      return address;
}
```

Kód. 15 Metóda pre získanie adresy zákazníka

Adresu ďalej posielame do komponenty s názvom „*recipentList*“. Prostredníctvom rercipientListu je zabezpečená komunikácia s externou API geocode, ktorú poskytuje spoločnosť Google a je voľne dostupná. API geocode poskytneme adresu zákazníka a tá nám vráti dáta vo formáte JSON. V týchto dátach je uvedená poloha zákazníka v zemepisných súradniciach.[18]

```
<recipientList>
 <simple>
 https4://maps.googleapis.com/maps/api/geocode/json?address=${body}&amp;
  sensor=false&amp;key={{googlekey}}
 </simple>
</recipientList>
```

Kód. 16 Komponta na získanie dát od Google API geocode

Nasledujúcich krokoch sú získané súradnice spracované tak aby ich bolo možné použiť pri ďalšej komunikácií s Google API ktoré je obsiahnuté v komponente číslo 21. Tentokrát je využitá API s názvom „*nearbysearch*“, ktorá poskytne zemepisné súradnice najbližšieho elektro obchodu na základe zadaného kľúčového slova, ktoré predstavuje názov obchodu a tiež na základe nastaveného dosahu hľadania.

```
<recipientList delimiter="|">
 <simple>
 https4://maps.googleapis.com/maps/api/place/nearbysearch/json?location=
 ${headers.customerLocation}&amp;radius=5000&amp;type=electronics_store
 &amp;keyword=alza&amp;key={{googlekey}}
 </simple>
</recipientList>
```

Kód. 17 Komponta na získanie dát od Google API geocode

Tak ako v predchádzajúcom prípade je potrebné získané súradnice upraviť tak aby bolo možné ich použiť pri poslednom komponente slúžiacom na komunikáciu s Google API. Posledná API má názov „*distancematrix*“. Slúži pre získanie vzdialenosti medzi dvoma alebo viacerými súradnicami. Súradnice o polohe zákazníka sú ukladané v komponente číslo 20 ako hlavička správy. Druhé súradnice sme získali v krokoch, ktoré sú popísanie vyššie.

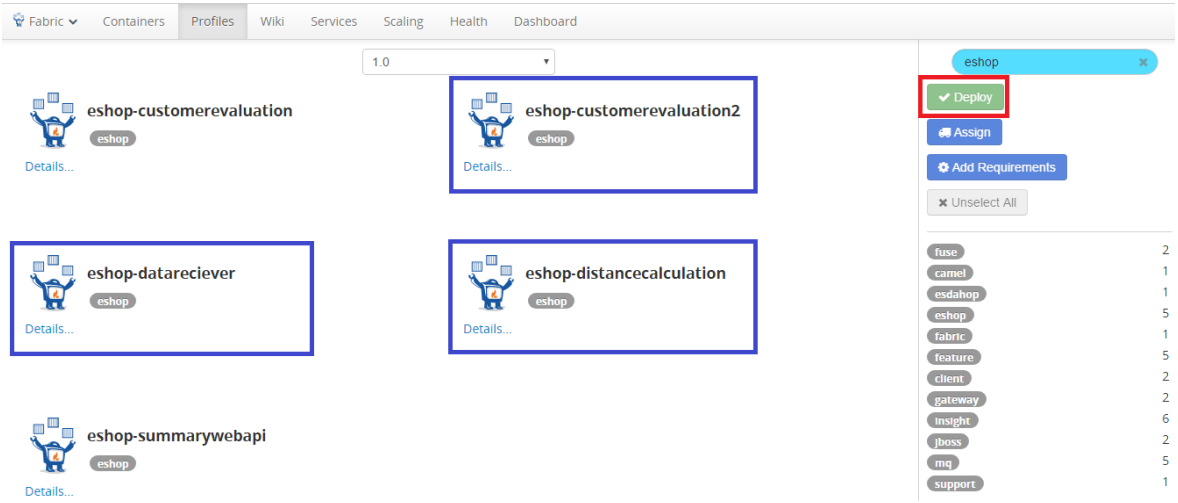
Získali sme teda vzdialenosť bydliska zákazníka od najbližšieho elektro obchodu kde si môže objednávku vyzdvihnúť v komponentoch 25 a 26 je údaj o vzdialenosť spracovaný a v komponente číslo 27 uložený do externej databázy.

5.5 Implementácia podnikovej zbernice do virtuálneho prostredia

V kapitole číslo 4 bolo nastavené virtuálne prostredie tak aby bolo možné integračné riešenie jednoducho a priamo z vývojového prostredia nasadiť. Jednotlivé služby je potrebné nasadiť do virtuálneho prostredia fabriky samostatne jednu po druhej. Spočiatku je potrebné dostať sa v JBoss Developer Studio do štruktúry projektov a pravým tlačidlom myši kliknúť na projekt ktorý bude nasadzovaný. V menu zvoliť „*Run As*“ a „*Maven build*“ viac Obr. 28. Otvorí sa okno ktoré je zobrazené na Obr. 29 kde do políčka „*goal*“ je nutné zadať reťazec „*fabric8:deploy*“ a potvrdiť tlačidlom „*Run*“. Tento proces je potrebné aplikovať na každú s vytváraných služieb.

5.5.1 Vytvorenie kontajnerov v virtuálnom prostredí fabriky

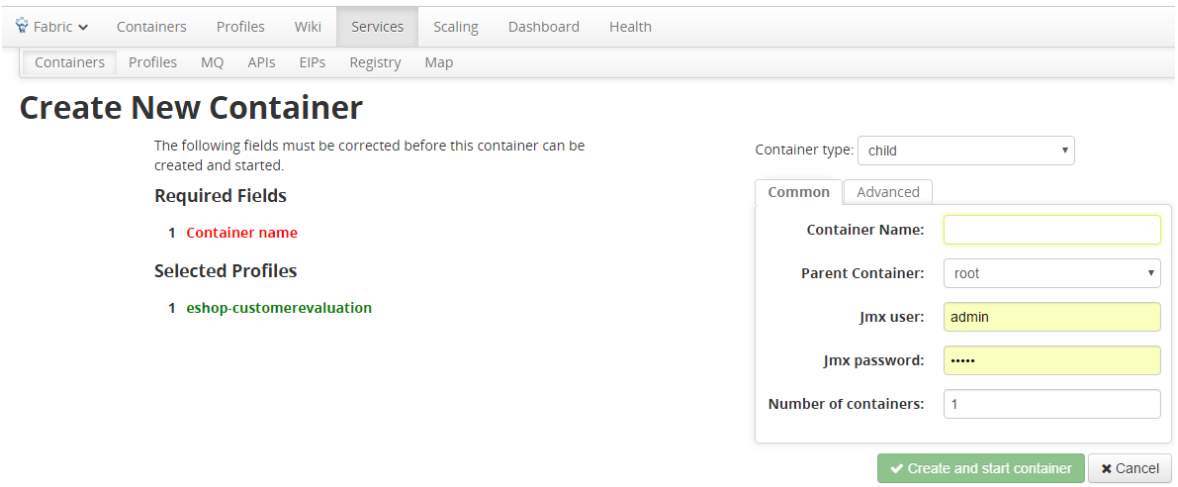
Pre vytvorenie kontajnerov je potrebné spustiť konzolovú aplikáciu JBoss Fuse ktorého inštalácia bola popísaná v kapitole 4.1. V zložke s rozbalením JBoss Fuse prejdeme do zložky bin a pustíme aplikáciu s názvom fuse.bat. Po spustený JBoss Fuse je potrebné spustiť internetový prehliadač kde zadáme adresu http://localhost:8181/hawtio/login. Prihlásime sa pod pred vyplnenými prihlasovacími údajmi teda prihlasovacie meno : admin , heslo : admin. Po prihlásení sa zobrazí prostredie „*hawtio*“, kde je možné spravovať vytvorenú fabriku. Prejdeme teda do záložky „*Profiles*“ ktorá je umiestená v hornom menu. Zobrazí sa stránka kde sú zobrazené modrým ohraničením nasadzované služby, viac Obr.30.



Obr. 31 Výber a nasedenie služieb

Pre funkčnosť podnikovej zbernice je nutné jednotlivé služby nasadiť do kontajnerov. To uskutočníme kliknutím na tlačidlo ktoré je na Obr. 31 ohraničené červenou farbou teda „*deploy*“.

Po kliknutí na tlačidlo deploy sa dostaneme na stránku kde je potrebné vytvoriť nový kontajner. V menu „*Container Type*“ zvolíme možnosť „*child*“, pomenujeme kontajner, zvolíme rodiča kontajnera „*root*“, zadáme prihlasovacie údaje „*admin*“ , „*admin*“. zvolíme počet kontajnerov 1 a potvrdíme tlačidlom „*Create and start container*“ pričom tento postup opakujeme pre každú vytvorenú službu viac Obr. 32.



Obr. 32 Vytvorenie a spustenie kontajnera

6 OVERENIE FUNKČNOSTI PODNIKOVEJ ZBERNICI NA ALIKÁCIÍ NOPCOMMERCE

Po vytvorený návrhu podnikovej zbernice je nutné otestovať jej funkčnosť. Podniková zbernica je schopná prijať správy, spracovať ich a tiež, komunikovať s externými API a databázou. Posielanie správ vykonáva voľne dostupná eCommerce aplikácia nopCommerce.

6.1 Webová aplikácia nopCommerce

Ako už bolo spomínané nopCommerce je voľne dostupne eCommerce riešenie založené na rozhraní ASP.NET MVC verzia 4.0. Taktiež disponuje s databázou typu MS SQL. Kdeže celá podniková zbernica je navrhovaná na platforme Java spojíme tým dve rozličné platformy.

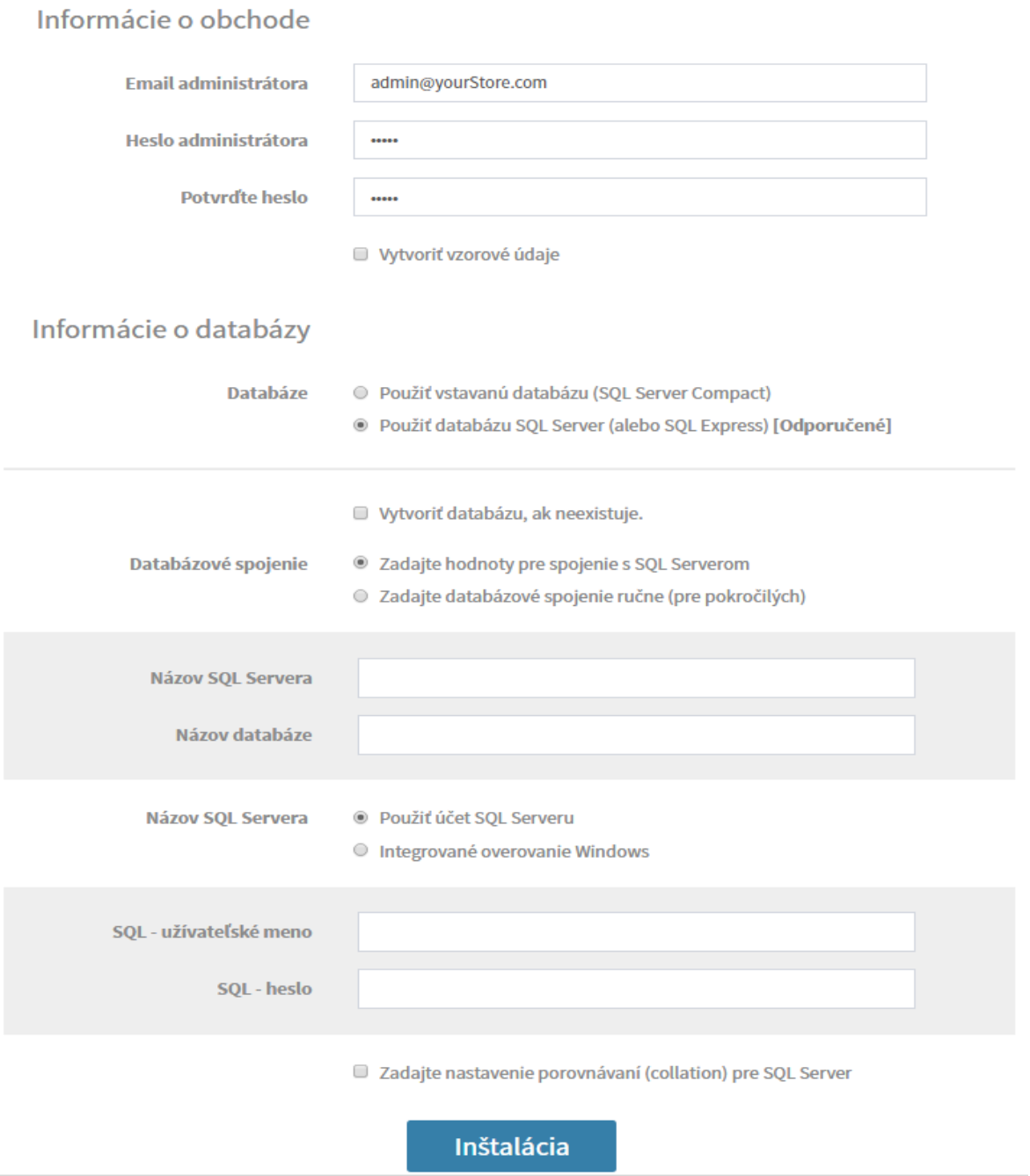
6.1.1 Výhody aplikácie nopCommerce

- Všetky platobné a prepravné moduly sú písane formou pluginu[12], takže jednoducho môžeme nepotrebné moduly odstrániť. V prípade že si vytvoríme vlastný platobný alebo prepravný modul, nie je potrebné zasahovať do aplikácie stačí daný plugin nainštalovať. Funkcionalitu aplikácie môžeme rozšíriť nie len o platobné a prepravné moduly, môžeme tiež vytvoriť ľubovoľnú funkcionalitu a do aplikácie jednoducho pridať.
- Zdrojový kód aplikácie je prepracovaný a vytváraný profesionálnym tímom programátorov, takže nie je dôvod sa obávať neprehľadného alebo nefunkčného kódu.
- Tým nopCommerce je pripravený neustále poskytovať podporu a pomoc pri vyskytnutých problémoch.
- Aplikácia má pripravené API pre nadobudnutie komunikácie cez webové služby.
- Aplikácia poskytuje grafické rozhranie kde je možné pridávať nové produkty, upravovať exitujúce, pridávať nové kategórie a tiež upraviť užívateľské rozhranie aplikácie.
- Jednoduchá inštalácia a prvé spustenie

[12] Zásuvný modul ktorý nepracuje samostatne ale ako doplnkový modul inej aplikácie.

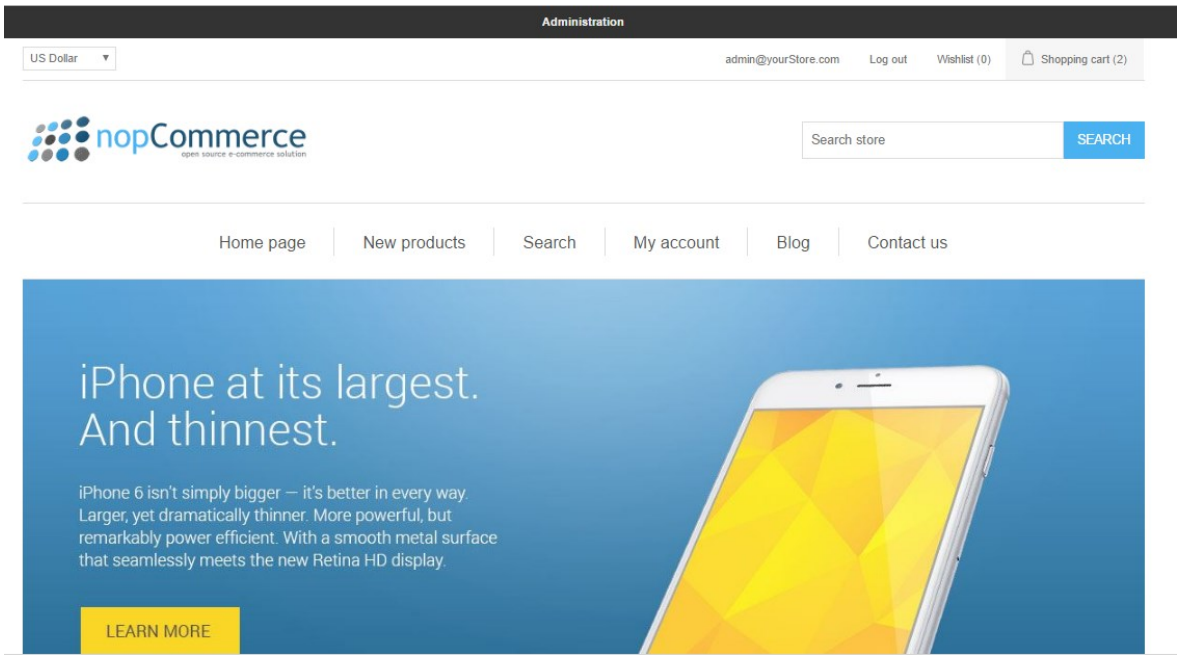
6.1.2 Inštalácia nopCommerce

Aplikáciu nopCommerce je možné si stiahnuť s oficiálnych webových stránok. Po stiahnutí zdrojového kódu je potrebné projekt rozbaliť a otvoriť ho v ľubovoľnom vývojovom nástroji, ktorý podporuje rozhranie ASP.NET MVC. Aplikáciu vo vývojovom prostredí pustíme čo otvorí internetový prehliadač kde sa zobrazí inštalačný formulár, „*nopCommerce instalation*“ ktorý je zobrazený na Obr. 33.



Obr. 33 Inštalácia aplikácie nopCommerce

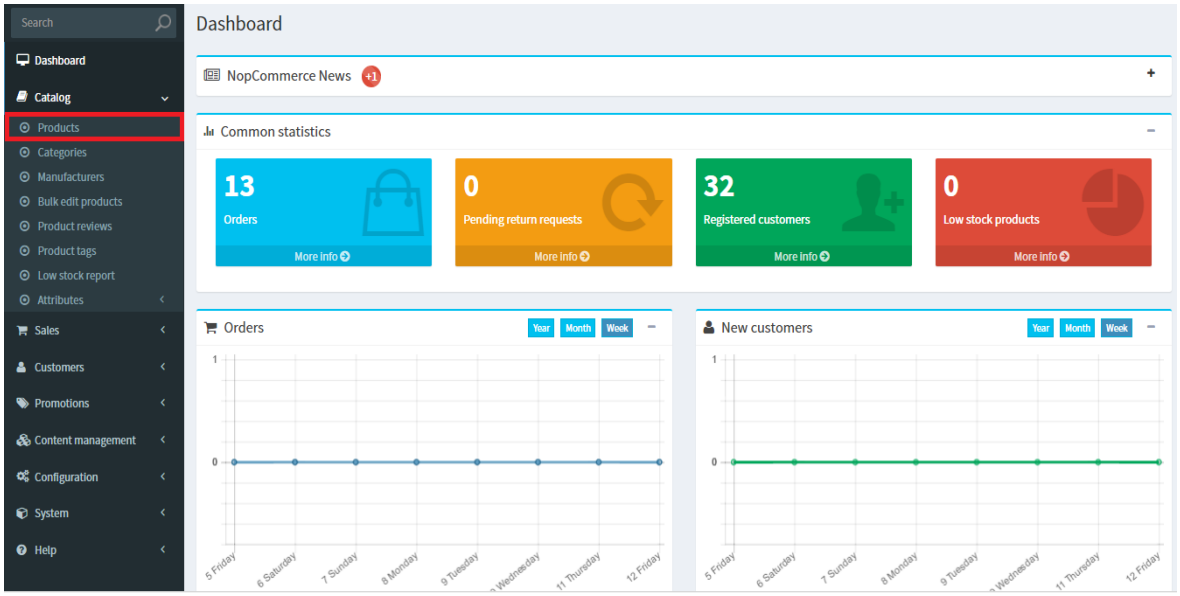
Po vyplnení formulára pre inštaláciu aplikácie stačí kliknúť na tlačidlo inštalácia a aplikácia sa postará o vytvorenie novej databázy v rátane všetkých potrebných tabuliek, procedúr a jediným vytvorením administrátorským účtom. Na Obr. 34 je ukážka prvého pustenia aplikácie nopCommerce.



Obr. 34 Ukáža prvého pustanie aplikácie nopCommerce

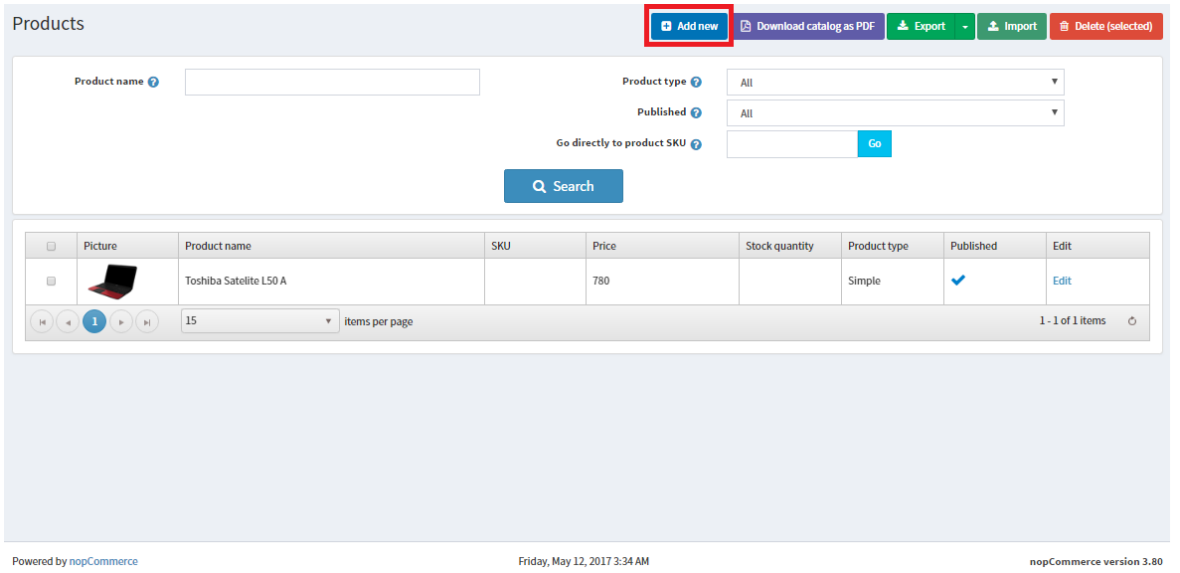
6.1.3 Pridávanie produktov

Pridávanie produktov do aplikácie nopCommerce je možné priamo s aplikácie. To znamená že nie je potrebné upravovať zdrojový kód. Je potrebné byť prihlásení ako administrátor a prejsť do rozhrania pre úpravy internetového obchodu, do ktorého je možné sa dostať kliknutím na odkaz „*Administration*“ umiestený nad hlavičkou aplikácie. Na Obr. 35 je zobrazené rozhranie pre administráciu aplikácie. V rozhraní sa vyskytujú tiež rôzne štatistické údaje ako počet vytvorených objednávok za určité obdobie, počet registrovaných zákazníkov. Taktiež poskytuje možnosť si tieto štatistické údaje prezrieť v grafoch. Sledované štatistické údaje je možné podľa potreby meniť.



Obr. 35 Ukážka rozhrania pre úpravu aplikácie nopCommerce

Pre pridanie nového produktu je potrebné sa dostať do spravovania produktov. Na Obr.35 je v červeno ohraničený odkaz pre presmerovanie do pod sekcie produkty.



Obr. 36 Podsekcia správa produktov

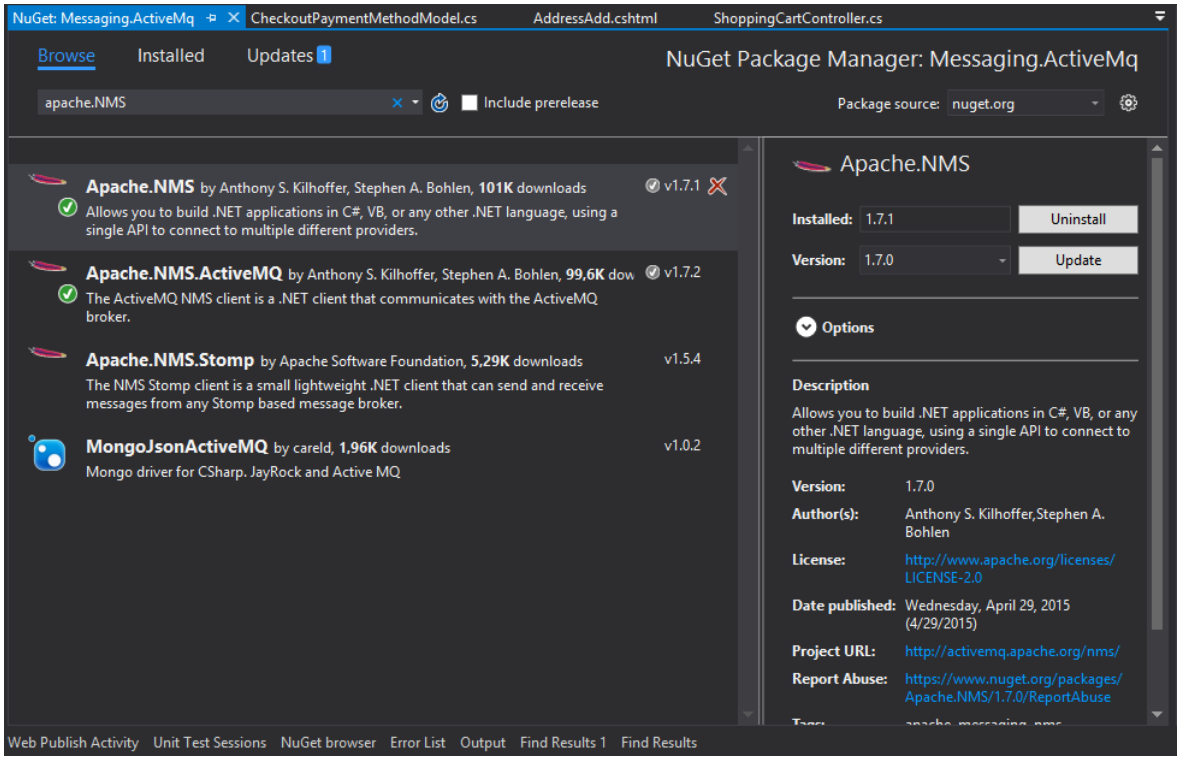
S Obr. 36 je vidieť že v sekcií spravovanie produktov sú k dispozícií okrem základných funkcii ako editácia, vytváranie a mazanie produktov rôzne iné funkcie, ako napríklad export zoznamu produktov do rôznych formátov alebo import produktov s tabuľkového formátu excel. Nový produkt pridáme kliknutím na tlačidlo ohraničené červenou farbou. To nás presmeruje na formulár vyplnenia údajov o produkte.

6.1.4 Pridanie funkcionality pre zasielanie správ na podnikovú zbernicu

V aplikácií nopCommerce nie je zahrnutá funkcionalita, ktorá by dokázala zasielať správy na vytvorenú podnikovú zbernicu platformy JBoss Fuse. Avšak už bolo spomínané táto aplikácie ja ľahko rozšíriteľná preto nebol problém túto funkcionalitu pridať. Aplikácia zašle správu na zbernicu vždy pri registrácií nového užívateľa a pri vytvorení novej objednávky.

To zabezpečuje balíček Apache.NMS, ktorý je poskytovaný platformou .NET. Prostredníctvom tohto balíčka sme schopní prepojiť komponentu Apache ActiveMQ s platformou .NET. To znamená že sme schopní prijímať správy z fronty ActiveMQ a tiež odosielať správy do fronty ActiveMQ[13].

Balíček je voľno dostupný ako nuget to znamená že jeho inštalácia je jednoduchá rýchla. Na Obr. 37 je zobrazená ukážka inštalácie nugetu Apache.NMS vo vývojovom prostredí Microsoft Visual Studio.



Obr. 37 Ukážka inštalacie nugetu Apache.NMS

6.1.4.1 Odosielanie správy pri registrácií nového zákazníka

Ako už bolo spomínané odosielanie správy sa uskutočňuje pri každej registrácií nového užívateľa. To znamená že akonáhle registrácia úspešne prejde validáciou vykoná sa nasledovný kód, ktorý zabezpečí poslanie správy. Odosielaný je objekt „*messageCustomer*“, ktorý obsahuje všetky potrebné informácie. ActiveMQ príjme správu vo formáte, ktorý bol popísaný v predchádzajúcej kapitole konkrétne Kód.7.

```
Uri connectUri = new Uri("tcp://localhost:61617");
IConnectionFactory factory = new NMSConnectionFactory(connectUri);
IMsgConnectionFactory msgConnectionFactory = new MsgConnectionFactory(factory,
"admin", "admin", null);
var connection = msgConnectionFactory.CreateConnection();
IMsgSessionFactory msgSessionFactory = new MsgSessionFactory();
var session = msgSessionFactory.CreateSession(connection);
MessagingInfo messagingInfo = new MessagingInfo
 {
    Name = "queue1",
    MessagingType = MessagingType.Que
 };
IDestinationFactory destinationFactory = new DestinationFactory();
MsgSender msgSender = new MsgSender(msgConnectionFactory, msgSessionFactory, desti
nationFactory);
var messageCustomer = new MessageCustomer
{
  CustInfo = "EshopCustomer",
  custInfoId = customer.Id,
  firstName = model.FirstName,
  lastName = model.LastName,
  age = (int)DateTime.Now.Year - model.DateOfBirthYear.Value,
  address = address_
};
var modelAddress = new CustomerAddressEditModel();
msgSender.SendMessage(messageCustomer, messagingInfo);
```

Kód. 18 Odosielanie správy na ActiveMQ pri registrácií zakazníka

6.1.4.2 *Odosielanie správy pri vytvorený novej objednávky*

Ďalším prípadom odosielania správ na ActiveMQ je prípad kedy zákazník vytvorí novú objednávku. Pričom odosielaná sprava je tak isto vo forme objektu tento krát je to však objekt typu „*MessageOrder*“. Po úspešnom vytvorený novej objednávky sa vykoná nasledujúca časť kódu :

```
Uri connectUri = new Uri("tcp://localhost:61617");
IConnectionFactory factory = new NMSConnectionFactory(connectUri);
IMsgConnectionFactory msgConnectionFactory = new MsgConnectionFactory(factory, "ad-
min", "admin", null);
var connection = msgConnectionFactory.CreateConnection();
IMsgSessionFactory msgSessionFactory = new MsgSessionFactory();
var session = msgSessionFactory.CreateSession(connection);
MessagingInfo messagingInfo = new MessagingInfo
 {
 Name = "queue1",
 MessagingType = MessagingType.Que
 };
IDestinationFactory destinationFactory = new DestinationFactory();
MsgSender msgSender = new MsgSender(msgConnectionFactory, msgSessionFactory, desti-
nationFactory);
IList<MessageOrder> messageOrders =
placeOrderResult.PlacedOrder.OrderItems.Select(orderItem => new MessageOrder()
 {
   OrderInfo = "EshopOrder",
   orderInfoId = orderItem.OrderId,
   orderTime = DateTime.Now.ToShortDateString(),
   productType = orderItem.Product.ProductType.ToString(),
   shortProdInfo = orderItem.Product.ShortDescription,
   productName = orderItem.Product.Name,
   customerId = _workContext.CurrentCustomer.Id
 }).ToList();
_paymentService.PostProcessPayment(postProcessPaymentRequest);
foreach (var messageOrder in messageOrders)
  {
    msgSender.SendMessage(messageOrder, messagingInfo);
  }
```

Kód. 19 Odosielanie srávy na ActiveMQ pri vytvorení objednávky

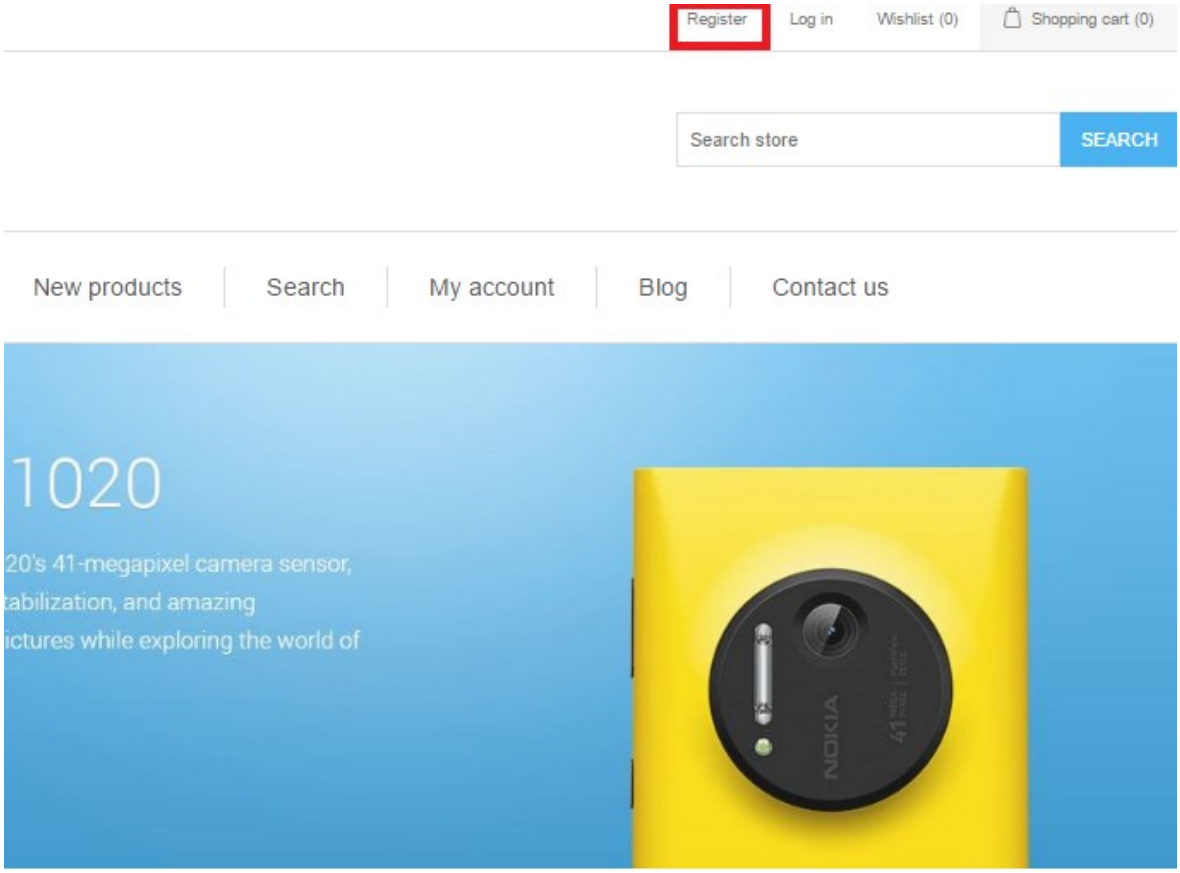
6.2 Overenie funkčnosti podnikovej zbernici služieb

6.3 Zasielanie správ na podnikovú zbernicu služieb

Celý proces integrovania začne zasielaní správ na komponent podnikovej JBoss Fuse ActiveMQ. Aby sme dosiahli zaslanie správ je potrebné sa zaregistrovať do internetového obchodu alebo vytvoriť novú objednávku.

6.3.1 Registrácia nového zákazníka

Ako prvý krok pri registrácií je potrebné sa v aplikácií prekliknúť na stránku s registračným formulárom, dosiahneme to kliknutím na odkaz „*Register*“ umiestnený v hornej lište aplikácie tak ako je znázornené červeným ohraničením na Obr. 38.

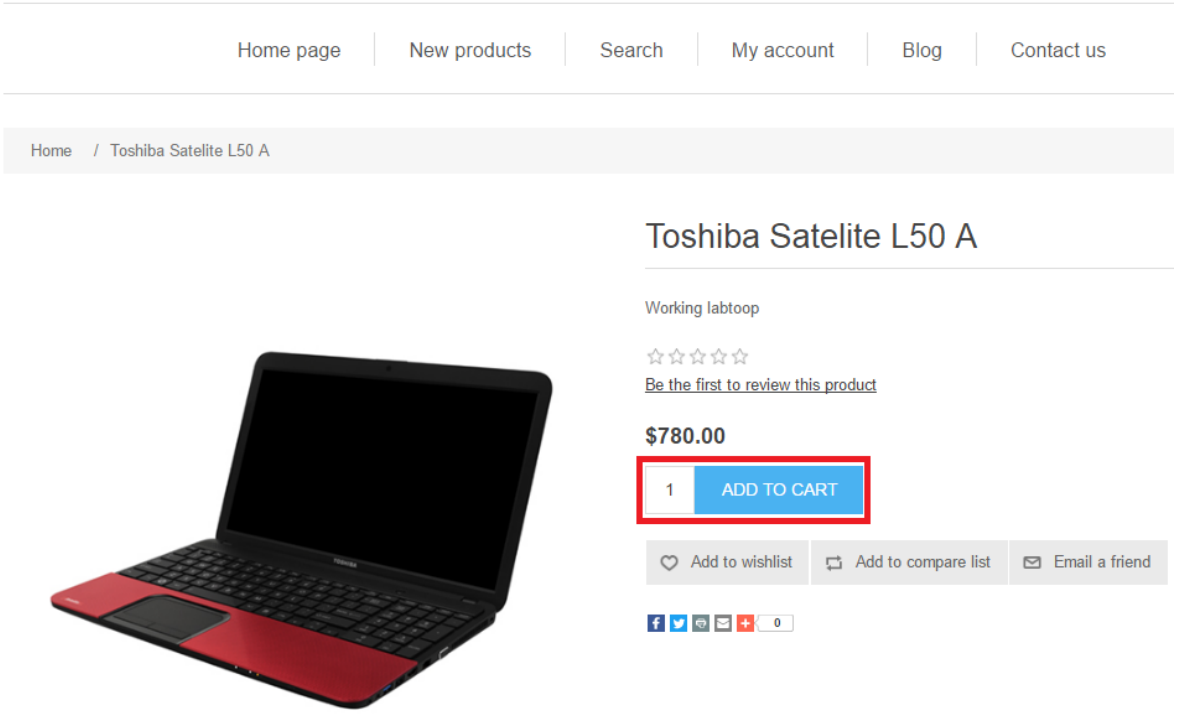


Obr. 38 Presmerovanie na registráciu zákazníka

Po vyplnený základných údajov ako pohlavie, meno, priezvisko, vek zákazníka a adresa zákazníka potvrdíme registráciu kliknutím na tlačidlo „*REGISTER*“. Po potvrdení registrácie aplikácia pošle správu s údajmi zákazníka na adresu ActiveMQ podnikovej zbernice.

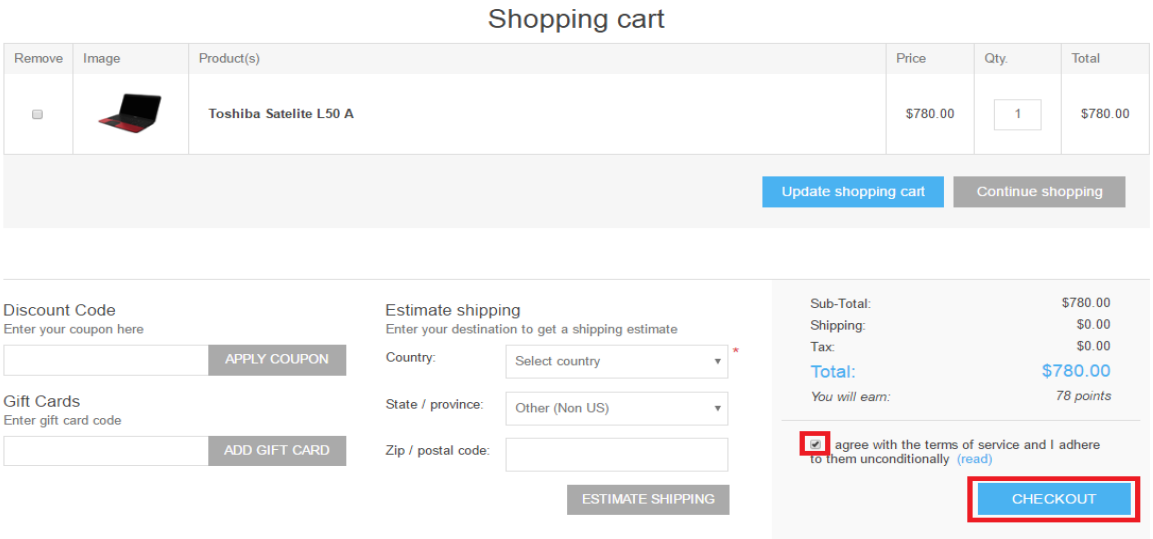
6.3.2 Vytvorenie novej objednávky

Ďalším spôsobom ako odoslať správu na podnikovú zbernicu je vytvorenie novej objednávky zákazníkom. Ako prvé pri vytvorení novej objednávky je potrebné vyhľadať daný produkt a pridať ho do nákupného košíka.



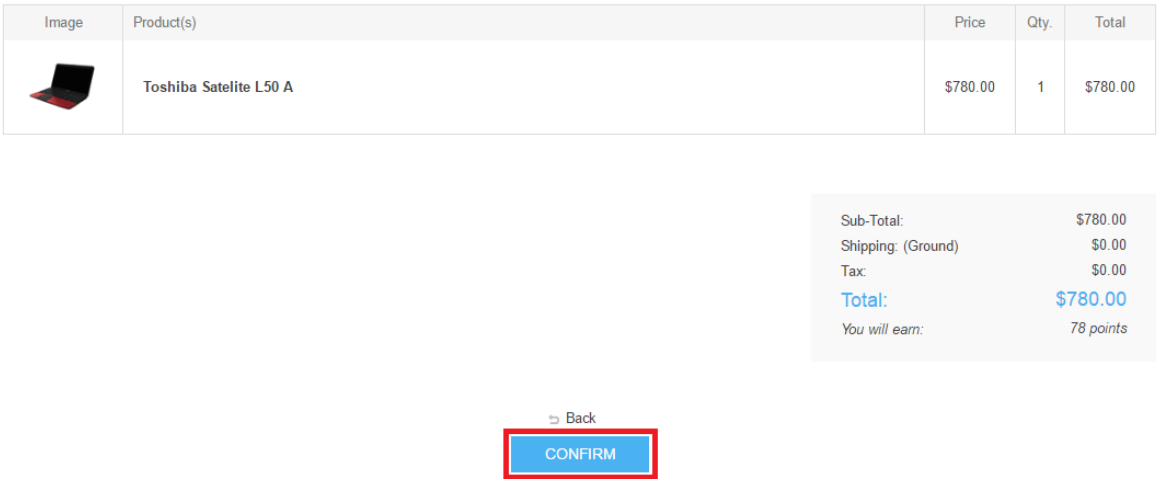
Obr. 39 Vloženie produktu do nákupného košíka

Prejdeme do nákupného košíku kde potvrdíme súhlas s podmienkami a klikneme na tlačidlo „*CHECKOUT*“.



Obr. 40 Nakupný košík

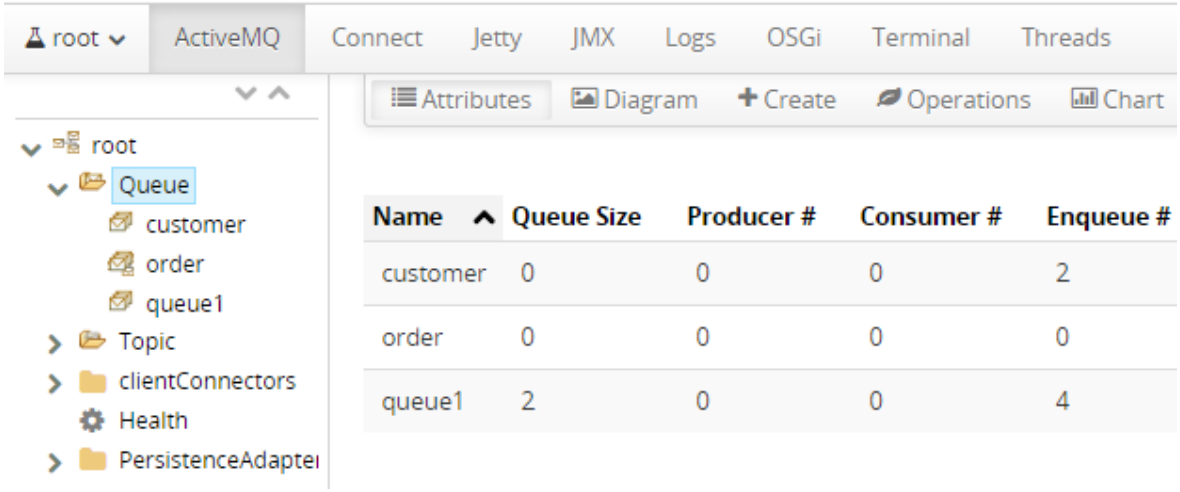
V poslednom kroku vytvorenia novej objednávky potvrdíme adresu, ktorá je definovaná už pri registrácií nového zákazníka , rovnako je nutné potvrdiť ostatné formuláre ktoré sa týkajú druhu platby spôsobu doručenia a podobne. Nakoniec sa dostaneme k potvrdeniu objednávky viac Obr. 41. Po potvrdený sa odošle ďalšia správa na ActiveMQ komponent JBoss Fuse podnikovej zbernici avšak tentokrát s údajmi o vytvorenej objednávke.



Obr. 41 Potvrdenie vytvorenia novej objednávky

6.4 Prijatie správ

Po odoslaní správ s aplikácie je úlohou podnikovej zbernice dané správy prijať a ďalej ich spracovať. Najskôr správy smerujú do fronty s názvom „*queue1*“, vytvorenej pre kontajner služby „*datareciver*“ kde táto služba načúva, kým sa do fronty nedostane správa. To znamená že pokiaľ bude kontajner pozastavený správy sa budú vo fronte zhromažďovať. Na Obr. 42 je táto situácia zobrazená, kedy sa správy zhromažďujú vo fronte „*queue1*“ pokiaľ sa kontajner „*datareciver*“ nespustí.

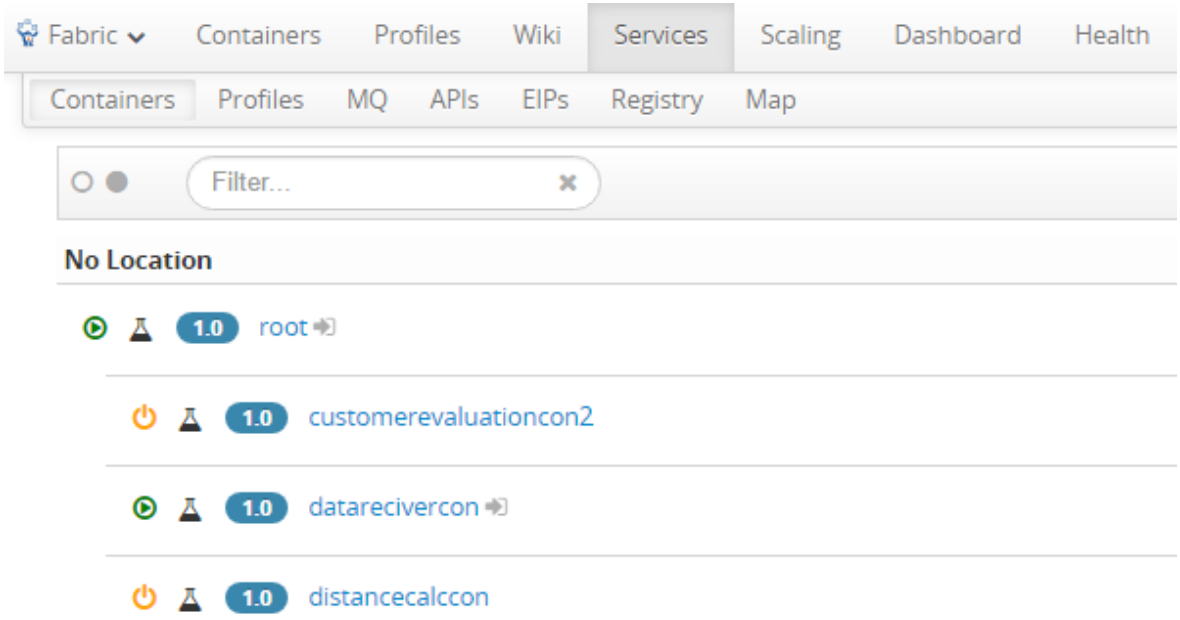


Obr. 42 Stav fronty queue1 pri pozastavenom kontajnery „*datareciver*“

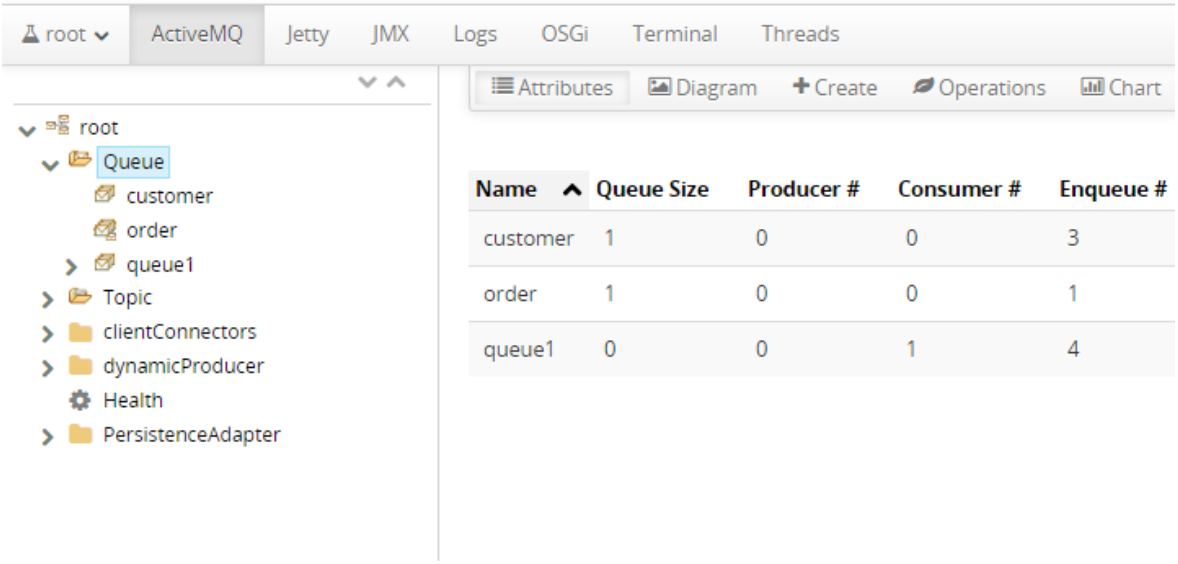
6.5 Spracovanie prijatých správ

6.5.1 Zapnutý Kontajner „*datarecivercon*“

Spracovanie prijatých správ začína rozdelením podľa toho či sa jedná o správu o registrácií zákazníka alebo o vytvorenie novej objednávky. A túto funkcionalitu sa stará kontajner „*datarecivercon*“, v ktorom je nasadená služba „*datareciver*“. Po zapnutí kontajneru (Obr. 43) sa správy rozdelia jedna do fronty pre zákazníka a druhá do fronty pre objednávku. Viac Obr. 44.



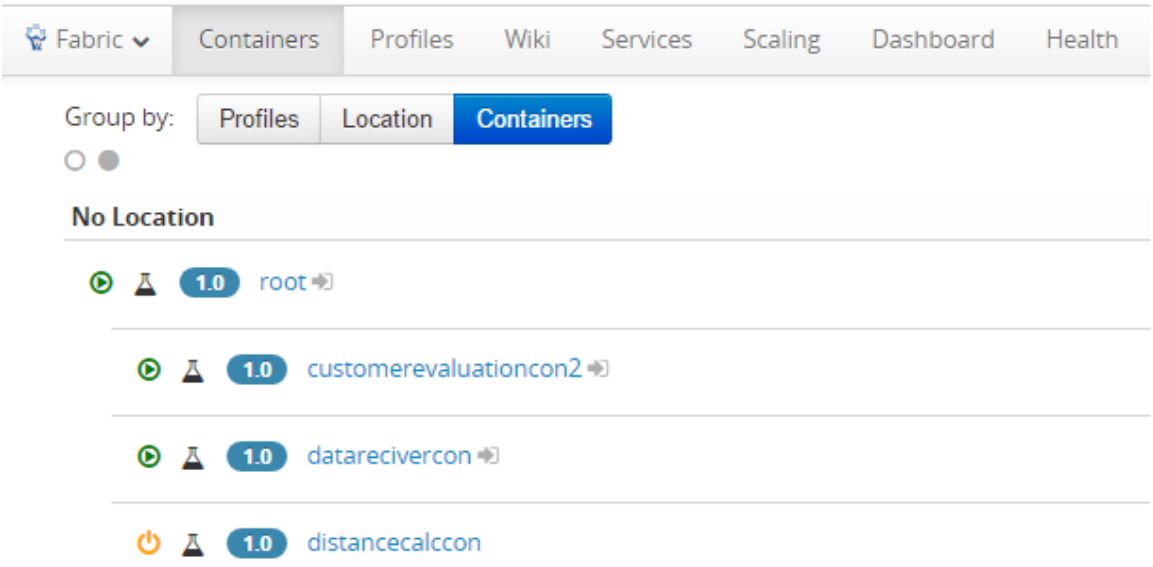
Obr. 43 Zapnutie kontajneru „*datareciercon*“



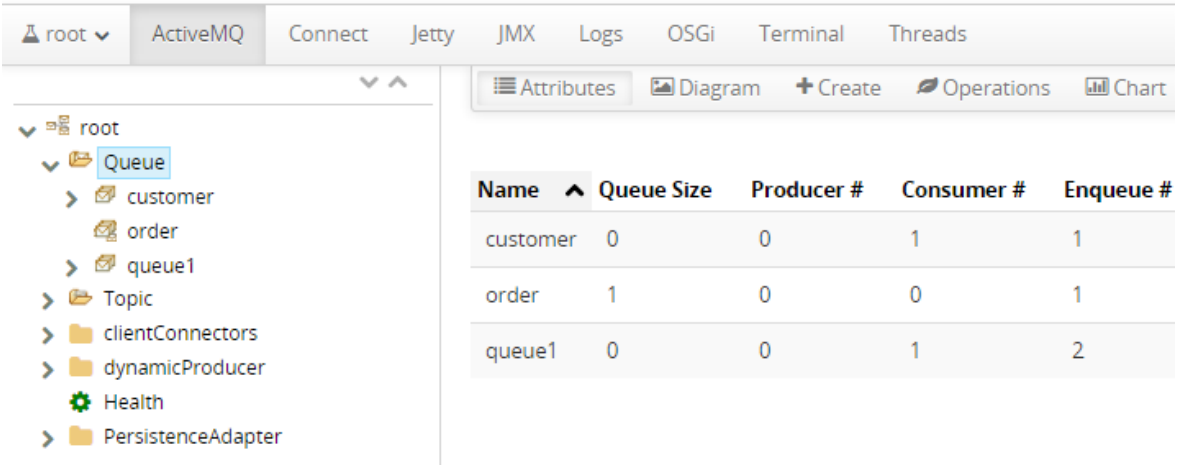
Obr. 44 Stav fronty po zapnutí „*datarecivercon*“

6.5.2 Zapnutie kontajneru „*customerevaliation2*“

Služba ktorá je nasadená v kontajneri „*customerevaluation*“ načúva fronte s názvom „*customer*“. Vo chvíli kedy sa v kontajneri vyskytne správa kontajner ju spracuje tak ako bolo popísane v kapitole 5.3 to znamená že zákazník sa v konečnom dôsledku uloží do externej databázy H2 a tým pádom sa fronta „*customer*“ vyprázdni.



Obr. 45 Spustenie kontajneru "*customerevaluation2*"



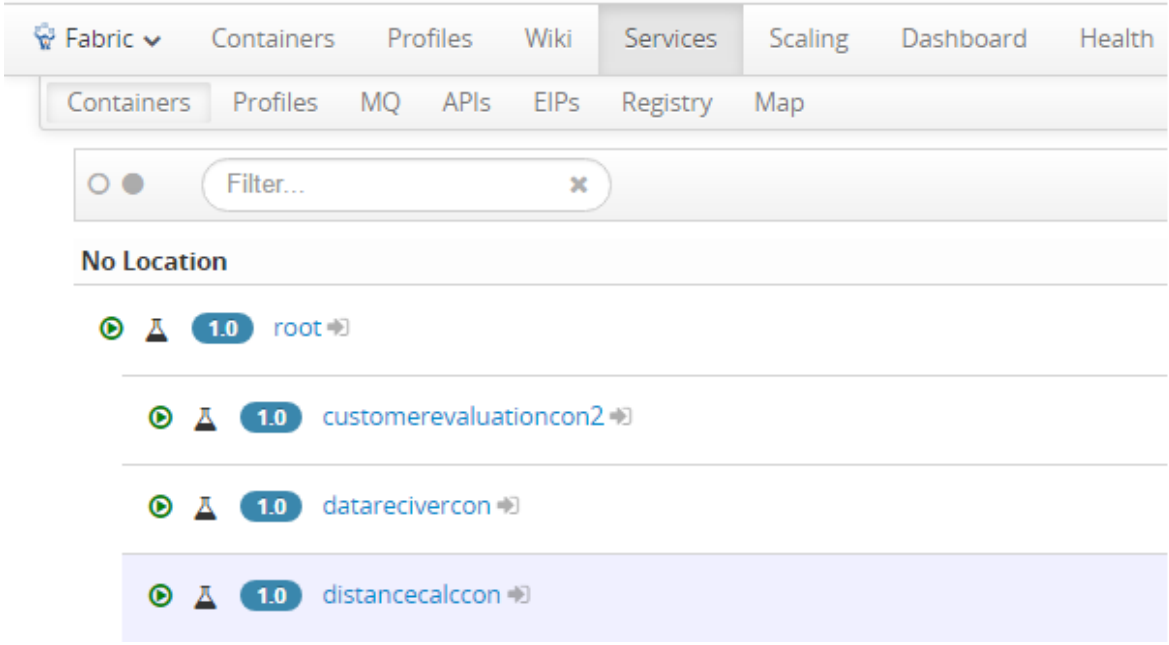
Obr. 46 Obsah fornty po spustený kontaineru „customerevaluation2“

Obr. 47 Obsah tabulky „*CUSTINFO*“v externej databáze H2

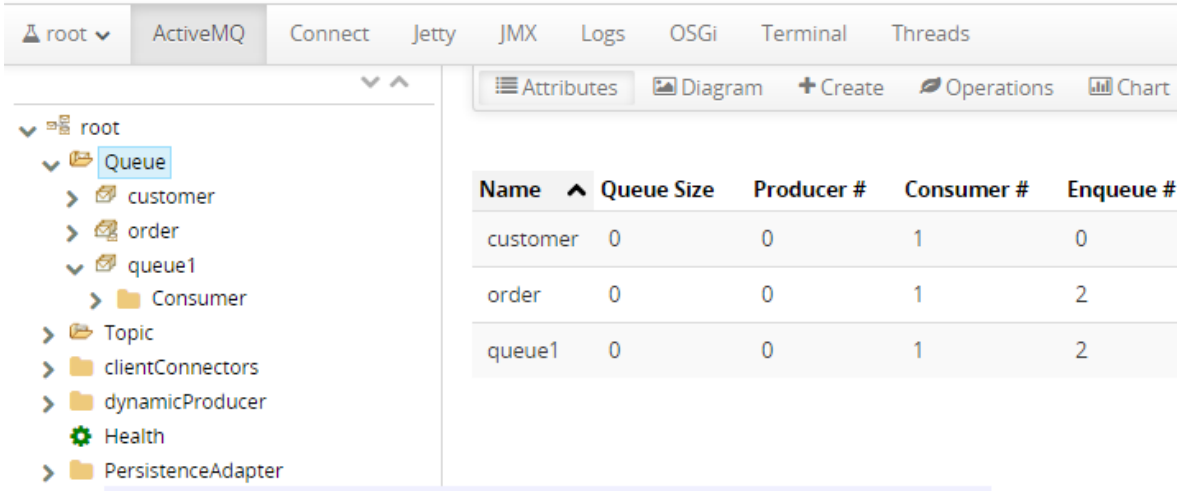
Na Obr. 47 a 46 vidíme že informácie o zákazníkovi sa úspešne spracovali a uložili do databáze.

6.5.3 Zapnutie kontajneru „*distancecalculationcon*“

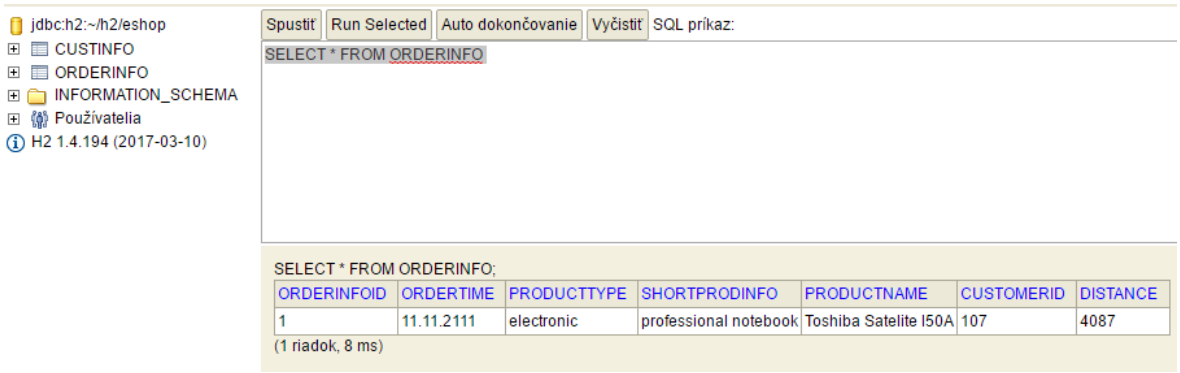
V poslednom kontajnery je nasadená služba ktorá vypočíta vzdialenosť medzi bydliskom zákazníka a najbližšej predajne v okruhu 5 kilometrov. Bližší popis tejto služby je vysvetlený v kapitole 5.4. Vypočítanú vzdialenosť spoločne s objednávkou služba uloží do externej databáze.



Obr. 48 Spustenie kontajneru distancecalculationcon



Obr. 49 Stav fronty po zapnutí konajneru „*distancecalcon*“



Obr. 50 Obsah tabulky "*ORDERNINFO*" v externej databaže H2

ZÁVER

V diplomovej práci som sa zaoberal implementáciou servisne orientovanej architektúry pomocou podnikovej zbernici služieb, ktorá bola vyvíjaná v integračnej platforme JBoss Fuse. Ako prvé som nakonfiguroval systém, tak aby bolo možné podnikovú zbernicu služieb navrhnúť a následne otestovať. Konfigurácia spočívala v nastavený systémových premenných, inštalácií samotného virtuálneho prostredia JBoss Fuse a v inštalácií vývojového nástroja JBoss Developer Studio pričom bolo potrebné, do vývojového prostredia dohľadať a nainštalovať integračný balíček určený pre prácu s JBoss Fuse.

Podnikovú zbernicu som navrhol tak aby rozširovala funkcionalitu internetového obchodu, o výpočet vzdialenosti bydliska zákazníka k najbližšiemu elektronickému obchodu v okruhu 5 kilometrov. Tým podniková zbernica prepojila odlišné programovacie prostredia a zabezpečila komunikáciu s rôznymi API a databázou.

Aplikácia internetového obchodu, ktorá zasiela správy do podnikovej zbernice je vypracovaná na frameworku ASP.NET MVC, pričom jej súčasťou je MS SQL databáza. V podnikovej zbernici pracujem s platformou Java a s SQL databázou typu H2. K výpočtu vzdialeností a zisteniu zemepisných súradníc používam tri rôzne voľne dostupné externé API od spoločnosti google.

Pomocou implementácie servisne orientovanej architektúry, podnikovou zbernicou JBoss Fuse, som všetky tieto komponenty bol schopný prepojiť a zabezpečiť tak medzi nimi komunikáciu a výmenu dát.

Na praktickom príklade som teda dokázal že integrácia rôznych technológií môže byť na základe princípov servisne orientovanej architektúry a jej implementácie cez podnikovú zbernicu uskutočniteľná, ľahko spracovateľná, prehľadná a taktiež rozšíriteľná. To nám prakticky otvára takmer neobmedzené možnosti pri tvorbe informačných systémov.

SEZNAM POUŽITÉ LITERATURY

[1] *Servisně orientovaná architekura*. Kompletní pruvodce. Holandská 8, 639 00 Brno: Computer Press, a.s, 2009. ISBN 978-80-251-1886-3.

[2] RYCHLÝ, Marek. SERVISNĚ ORIENTOVANÁ ARCHITEKTURA A JEJÍ APLIKACE V SYSTÉMECH SLEDOVÁNÍ A ŘÍZENÍ VÝROBY [online]. Božetěchova 2, 612 66 Brno, 2011 [cit. 2017-05-11]. Dostupné z: http://www.fit.vutbr.cz/~rychly/public/docs/arap11.soa-a-jeji-apl-pri-rizeni-vyroby/arap11.soa-a-jeji-apl-pri-rizeni-vyroby.pdf. Ústav informačních systémů, Fakulta informačních technologií, Vysoké učení technické v Brně.

[3] GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ. Podniková informatika: 2., přepracované a aktualizované vydání [online]. 2. vydání. U Průhonu 22, Praha 7: Grada Publishing, 2009 [cit. 2017-05-11]. ISBN 978-80-247-2615-1. Dostupné z: https://books.google.cz/books?id=gWzsFl5QJbYC&pg=PA361&lpg=PA361&dq=SOA+voln%C3%A1+vazba&source=bl&ots=dqIRLIJHQE&sig=9j7Zi1DS4-9rv8-uaUZLvUPwO0Y&hl=cs&sa=X#v=onepage&q=SOA%20voln%C3%A1%20vazba&f=true

[4] *Mulesoft: What is an ESB?* [online]. [cit. 2017-05-12]. Dostupné z: https://www.mulesoft.com/resources/esb/what-esb

[5] *Mulesoft: Eliminating Point-To-Point Integration Pain with Mule ESB - Use Cases* [online]. [cit. 2017-05-12]. Dostupné z: https://www.mulesoft.com/resources/esb/what-esb

[6] *Zato: esb-soa* [online]. [cit. 2017-05-12]. Dostupné z: https://zato.io/docs/intro/esb-soa.html

[7] KEEN, Martin, Susan BISHOP, Alan HOPKINS, et al. *Patterns: Implementing an SOA using an Enterprise Service Bus* [online]. IBM, 2004 [cit. 2017-05-12]. ISBN 9780738490007. Dostupné z: http://www.redbooks.ibm.com/redbooks/pdfs/sg246346.pdf

[8] *Programujte: zaklady-xml-webovych-sluzeb* [online]. [cit. 2017-05-12]. Dostupné z: http://programujte.com/clanek/2005081704-zaklady-xml-webovych-sluzeb/

[9] *Crossvale: jboss-fuse* [online]. [cit. 2017-05-12]. Dostupné z: http://www.crossvale.com/pdfs/jboss-fuse.pdf

[10] *Enterprise Service Bus* [online]. CHAPPELL, David. Posledné. O'Reilly Media, 2004, Chapter 6 [cit. 2017-05-12]. ISBN 978-0596006754. Dostupné z: http://www.onjava.com/pub/a/onjava/excerpt/esb_ch6/

[11] IBSEN, Claus a Jonathan ANSTEY. Camel in action. 2010. ISBN 9781935182368.

[12] *Jboss: chap-Introduction_to_Red_Hat_JBoss_Developer_Studio.* [online]. [cit. 2017-05-12]. Dostupné z: http://docs.jboss.org/tools/latest/en/User_Guide/html/chap-Introduction_to_Red_Hat_JBoss_Developer_Studio.html

[13] *Activemq apache: nms* [online]. [cit. 2017-05-12]. Dostupné z: http://activemq.apache.org/nms/

[14] Meeraj KUNNUMPURATH. JBoss 3.2 Deployment and Administration. 1. Apress, 2004. ISBN 978-1-4302-0819-8.

[15] *JBOSS Fuse: tutorialspoint simply easy learning* [online]. [cit. 2017-05-12]. Dostupné z: https://www.tutorialspoint.com/jboss_fuse/jboss_fuse_tutorial.pdf

[16] *Redhat: documentation* [online]. [cit. 2017-05-12]. Dostupné z: https://access.redhat.com/search/#/documentation

[17] SMYDER, Bruce, Dejan BOSANAC a Rob DAVIES. *ActiveMQ in Action*. 2011. ISBN 9781933988948.

[18] SMYDER, Bruce, Dejan BOSANAC a Rob DAVIES. *Enterprise Integration Patterns: Design, Buildingm and Deloying Messaging Solutions*. 1. ISBN 978-0321200686.

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SOA	Servisne orientovaná architektúra.
IT	Informačné technológie.
SOAP	Protokol jednoduchého prístupu k objektom.
WSDL	Jazyk popisu webových služieb
XML	Rozšíriteľný značkovací jazyk
XSD	Jazyk určení pre definovanie XML schémy
APP	Aplikácia
HTTP	Hypertextový prenosový protokol
HTTPS	Zabezpečený hypertextový prenosový protokol
WS	Webová služba
REST	Architektúra navrhnuté pre distribuované rozhranie
JMS	Distribúcia správ v Jave
STOMP	Protokol s textovo orientovanou správou
PT2PT	Protokol typu bod k bodu
MQ	Fronta správ
IDE	Integrované vývojové prostredie
API	Rozhranie programovania aplikácií
XHTML	Rozšíriteľný hypertextový značkovací jazyk
JSF	Technológia Java Server Faces
CDI	Kontext a Dependency Injection
JDK	Vývojové nástroje pre vývoj na platforme JAVA

SEZNAM OBRÁZKŮ

ZOZNAM TABULIEK